# Supporting Efficient Streaming and Insertion of XML Data in RDBMS

Timo Böhme, Erhard Rahm

University of Leipzig, Germany {boehme, rahm}@informatik.uni-leipzig.de http://dbs.uni-leipzig.de

**Abstract.** Relational database systems are increasingly used to manage XML documents, especially for data-centric XML. In this paper we present a new approach to efficiently manage document-centric XML data based on a generic relational mapping. Such a generic XML storage is especially useful in data integration systems to manage highly diverse XML documents. We focus on efficient insert operations, support of streamed data and fast retrieval of document fragments. Therefore we introduce a new numbering scheme called DLN (Dynamic Level Numbering) and several variants of it. A performance evaluation based on a prototypical implementation demonstrates the high efficiency of DLN.

# **1** Introduction

Today there is a high demand to manage XML data with relational database systems. All relational database vendors have added XML support to their products. Currently, two main kinds of mappings for integrating XML data into the relational model are supported [1]. The first is a document-centered one, and stores documents as a whole which poses restrictions on queryability and intra-document updates. The second is a data-centered one and requires a XML DTD or schema to map XML data to application-specific tables and attributes. While this approach supports application-specific SQL operations over the data it loses information such as element sibling order and leads to an expensive reconstruction of document fragments.

Both mappings are not able to efficiently manage large scale document-centric XML data<sup>1</sup> which can be updated and accessed via standard XML interfaces like XPath, XQuery or DOM<sup>2</sup>. These requirements can be met by a third kind of mapping: the structure-centered storage. It does not require a XML DTD or schema and maps the tree or graph structure of XML documents generically into predefined relations.

<sup>&</sup>lt;sup>1</sup> In [2] a general technique for translating XML queries to SQL for data-centered storage with preserved tree order information is proposed. However the queries cannot be executed efficiently since e.g. ancestor-descendant relationships have to be computed in a recursive way and document reconstruction needs several join and sort operations.

<sup>&</sup>lt;sup>2</sup> Document Object Model: http://www.w3.org/DOM

Such a generic XML storage is especially useful in data integration systems to manage highly diverse XML documents.

Early work on generic relational storage of XML data [3] showed that storing only parent child relationships results in poor performance for determining ancestor-descendant relationships and for document reconstruction. These problems can be solved by applying numbering schemes which represent the order of XML nodes and keep information about subtree containment. Except for a very recent proposal called ORDPATH [13], the known numbering schemes [1,6,7,8,9,10,11] either optimize only sequential processing of input data but not updates like inserting new subtrees, or are unsuited for streamed XML data, e.g. requiring a DOM representation.

We therefore developed a new numbering scheme called DLN (Dynamic Level Numbering) for generic (structure-centered) storage of XML data in relational databases. The main objectives of DLN include:

- support for sequential processing of large XML documents (streaming),
- efficient updates, especially the insertion of complex subtrees,
- fast reconstruction of documents or document fragments and
- no manual definitions for data insertion.

The rest of the paper is organized as follows. Next we discuss related work. Section 3 introduces the new DLN numbering scheme and some variants. Section 4 presents a comparison with ORDPATH and a performance evaluation. Finally Section 5 concludes the paper.

### 2 Related Work

Different relational mappings for generic storage of XML documents are proposed. In [3] the tree-like node structure of XML documents is represented as parent-child relationships, but this approach is inefficient for reconstructing documents. [4] encodes the document tree in binary relations, but also has performance difficulties for reconstructing document fragments. A multidimensional mapping using document id, value and path surrogate is published in [5]. However the approach does not deal with update operations and its path coding restricts the number of child elements per node.

A key approach to improve query and retrieval performance is the use of semantically meaningful node-ids when mapping XML data into nodes according to the Document Object Model (DOM). Therefore several numbering schemes have been proposed. One of the first numbering schemes supporting ancestor-descendant relationships was published in [6]. It labeled each tree node with a pair of preorder and postorder position numbers. So for each pair of nodes *x* and *y*, *x* is an ancestor of *y* if and only if *preorder*(*x*)<*preorder*(*y*) and *postorder*(*x*)>*postorder*(*y*). A similar scheme was chosen in [7]. While this numbering scheme is easy to compute and can be used for streamed XML data it is highly inefficient when new nodes are inserted. Here each node in preorder traversal coming after the inserted node has to be updated.

The update problem has been addressed by the *extended preorder* numbering scheme introduced in [8] and adopted in [9] as *durable node numbers*. They also use a

pair of numbers for each node. The first number captures the total order of the nodes within the document like the preorder traversal but leaves an interval between the values of two consecutive nodes. The second number is a range value. As with the preceding scheme the ancestor-descendant relationship between node *x* and *y* can be determined from *x* is an ancestor of *y iff*  $order(x) < order(y) \leq order(x) + range(x)$ . With the sparse numbering insert operations will not necessarily trigger renumbering of following nodes if the difference of the order value of preceding and following node is larger than the number of inserted nodes. However inserting new sub trees with a substantial number of nodes requires renumbering as well.

In [1] a numbering scheme called *simple continued fraction* (SICF) is proposed. It numbers the nodes from left to right and top-down. Each node number can be expressed as a sequence of integer values – adding an integer per tree level – or a fraction. This approach reduces the update scope<sup>3</sup> after a node insertion to the right siblings and their descendants of the inserted node. However this still can be a large number. Furthermore SICF fails if a certain tree depth is reached.

Another approach with left to right and top-down numbering is published in [10]. The so called *unique element identifiers* (UID) are based on a tree with a fixed fan-out of k. If a node has less than k children virtual nodes are inserted. The UID allows the computation of the parent node id and the id of child i. This approach has two main drawbacks: 1) fixed fan-out is problematic with irregular structured documents, 2) node insertion requires updates of all right siblings and their descendants.

Some of the UID drawbacks were tackled in [11] with the definition of recursive UIDs (rUID). Here the tree is partitioned in local areas allowing different fan-outs and reducing updates after insertion of nodes. However it needs access to the whole tree in order to compute the identifiers which prevents the streaming of data to be inserted.

Theoretical findings for labeling dynamic XML trees are given in [12]. The described schemes determine labels which are persistent during document updates and contain ancestor information. Furthermore lower bounds for the maximum label length are presented. However no sibling order of the XML nodes is maintained and therefore it is not suitable for general XML document management.

A recent paper proposed the so-called ORDPATH numbering scheme [13]. It is based on a hierarchical labeling scheme using a prefix-free encoding and supports insertion of new nodes without relabeling existing nodes. This is similar to our approach however differs conceptually in the construction of the node id and the insert semantics. Furthermore our numbering scheme can take advantage of information about the number of node siblings to minimize the length of node ids. We will compare both schemes in Section 4.

# **3** Dynamic Level Numbering Scheme (DLN)

Based on the study of the previous numbering schemes with their pros and cons and the goals stated in Section 1 we developed a new numbering scheme. It aims at supporting a wide range of XML documents, especially

<sup>&</sup>lt;sup>3</sup> the set of nodes whose numbers (potentially) have to be updated



Fig. 1. Dewey order encoding with update scope after insertion of middle chapter element

- irregular documents having nodes with low fan-out as well as high fan-out and
- large documents which can only be sequentially inserted.

Moreover, it should be stable under update operations and support efficient queries. In particular, the numbering should

- explicitly express the total nodes order to allow node clustering for high retrieval performance of document fragments using sequential scans,
- reduce the necessity for renumbering after insert operations,
- assist in the efficient processing of XPath queries, e.g. containment queries and
- in order to use this numbering scheme with a conventional RDBMS it should be exploitable by the indices and query optimizer of the database system.

Since simple preorder numbering has the mentioned drawbacks for insert operations, order encoding based on decimal classification (DC) like the Dewey system looks promising [14]. DC ids are composed of a sequence of numeric values separated by a dot. The root node is assigned a single numeric value. Child node ids start with the id of the parent node appended by a dot and a numeric value which we call the *level value*. The level value of a left sibling from a node A must be less than the level value of A. As illustrated in Fig. 1, this approach restricts the update scope after a node insertion to the right sibling nodes and their descendants. Besides this property the encoding has further advantages: 1) the parent can be computed from the id, 2) the ancestor-descendant relationship can also be determined using only the id value and 3) the ids can be sequentially assigned.

Unfortunately the DC encoding also has some shortcomings: 1) the id length depends on the tree depth, 2) a binary or string comparison of the ids may deliver wrong results with respect to the total node order, e.g. comparing 1.9 and 1.10 and 3) inserting a node as a child of a parent with a high fan-out may still result in a large number of nodes to be updated.

In [14] two solutions for the second shortcoming were proposed. The simple approach uses a fixed number of digits for each level number. Thus our example may be translated to 0001.0009 and 0001.0010 which now delivers a correct result using string compare. However this solution restricts the maximum fan-out of a node to a fixed value and on the other hand uses too much storage space for child nodes with





Fig. 2. DLN with adjusted number of digits

Fig. 3. Subvalues after insertion of left chapter subtree and middle section node in chapter 1.2

few siblings resulting in long path ids. Therefore the second approach uses UTF-8<sup>4</sup> encoding for the level numbers where small values can be represented by single bytes and larger values by two or more bytes. This encoding results in smaller path ids and permits binary compare operations.

With the UTF-8 encoding we still need at least one byte per level value. Given that most nodes of a XML document have a low fan-out this seems to be a waste of storage space. As a consequence of this and the still unsatisfying update behavior we developed a new numbering scheme called Dynamic Level Numbering (DLN) which is based on DC.

### 3.1 Basics of DLN

DLN contains solutions for the stated problems two and three of DC encoding and has an efficient binary representation which tackles problem one. In order to obtain comparable ids we took the simple approach with a fixed number of digits for level values (fixed length). However we altered this requirement so that only the level values of sibling nodes need to have the same lengths. Hence the number of digits per level value can be dynamically adjusted according to the number of sibling nodes. In the example of Fig. 2, the siblings at the second level use 1 digit, while the descendants of node *1.1* at the third level use 3 digits per level value.

DLN reduces the renumbering effort after insert operations by the introduction of *subvalues*. The idea is that between two consecutive level values *a* and *b* we can have further values by adding a suffix to *a*. Yet the resulting ids need to be larger than all ids of children nodes of *a*. This is accomplished by inserting a special character between *a* and the suffix which is greater than the dot separating the level values.

The application of subvalues is shown in Fig. 3. Nodes 1.0/1 and 1.2/1 could be inserted without renumbering the existing nodes. It is important to note that the inserted chapter node must not get level value 0 (or id 1.0). Otherwise we would have no possibility to add further nodes via subvalues to the left of it. Subvalues can be used recursively. For instance to insert a node between nodes with ids 1.1/1 and 1.1/2 we can add a further subvalue level and assign 1.1/1/1 to the new node. The only disadvantage of subvalues is the increased id length.

<sup>&</sup>lt;sup>4</sup> Unicode Transformation Format-8, defined in RFC 2279

In our early tests we coded DLN ids as strings of variable length. However we experienced bad scalability for query execution. It seemed that the query optimizer could not take into account metadata about statistical distribution of the string ids for its query plan. Consequently we chose to use a binary representation of the ids which can be processed efficiently and used by the optimizer.

For the binary representation of a DLN id the level values are binary coded using the same number of binary digits for all sibling nodes as required before. To separate the level values and to distinguish between values of the next level and subvalues we use only one bit. '0' means the following value belongs to the next hierarchy level whereas '1' depicts a subvalue. The binary representation of DLN is assumed in the sequel of the paper.

Before we discuss the properties of our encoding we give some examples to demonstrate the transformation into binary representation. If we use 2 bits for the first level and 4 bits for the second level we would encode our examples 1.09 as  $01\ 0\ 1001$  and 1.10 as  $01\ 0\ 1010$ . To insert a node between both siblings without renumbering we have to use a subvalue resulting in 1.09/01 or in binary notation  $01\ 0\ 1001\ 1\ 0001$ . The length of a subvalue should be identical to the length of the level value. With this property we can minimize the metadata needed to calculate the ids of following nodes and the parent node id.

#### 3.2 Properties of DLN

With our encoding we obtain a preorder numbering when comparing the ids left aligned and padded to the right with zeros. This can be easily deduced from the following observations: 1) if no subvalue is used sibling order is maintained by increasing values using a fixed length for the current level value 2) if *a*, *b*, *c* are following sibling nodes with ids id<sub>a</sub>, id<sub>b</sub>, id<sub>c</sub>, respectively, and id<sub>a</sub> and id<sub>c</sub> have consecutive level values and id<sub>b</sub> was created using id<sub>a</sub> and a subvalue then id<sub>a</sub><id<sub>d</sub> because they have the same prefix in the length of id<sub>a</sub> and id<sub>b</sub> has at least one 1-bit at the next position and id<sub>b</sub><id>c because id<sub>c</sub> is greater in the length of id<sub>a</sub>, 3) children append at least one 1-bit to its parent id leading to a greater id value, 4) a following sibling *s* has a greater id than all descendants of the current node *c* because *s* either has an increased level value compared to *c* and therefore is already greater in the length of id<sub>c</sub> or *s* uses a subvalue with a prefix identical to the id of *c* with a following separator 1-bit which is greater than the separator 0-bit of the descendants of *c*.

The update scope after a node insertion is in the worst case equal to the one of the standard DC encoding. However the concept of subvalues largely reduces the need for renumbering nodes. Even if no subvalues between two siblings are left because a maximum id length is reached renumbering the next sibling with its descendants will be sufficient in most cases. Starting at the document root element the number of possible subvalues decreases logarithmically if there is a maximum id length. This is the intended behavior since renumbering at higher levels should be avoided.

Beside the total node order DLN supports the computation of ancestor-descendant relationships between two nodes using the length l of the id from the node n with the smaller id value. If the first l bits are identical n is an ancestor of the other node. The parent id of a node n can be computed using l and the length of its level value.

#### 3.3 DLN and Streamed Data

The adaptable length of level values can be used to adjust to varying node fan-outs and to produce short ids. However determining the optimal length requires a-priori knowledge about the sibling number. At insertion time this is the case if the complete document tree is available in a DOM-like representation. Though this does not hold for streaming XML data requiring sequential processing.

With the following algorithm it is possible to sequentially load streaming XML data with unknown fan-out and to dynamically adapt the number of bits. It combines the bits of multiple subvalues which are added after a certain number of sibling nodes have been inserted. When the first child node c of an existing node is processed, a minimal number of bits, n, is used for the level value. After the lower n-1 bits for siblings of c have been used ( $2^n-1$  nodes in total), a subvalue is added for the next sibling. Now we set the highest bit of the level number to 1 and use the concatenated remaining bits crb of the level value and the subvalue. Again only crb-1 bits are used for the next siblings to be inserted and the algorithm repeats recursively.

Table 1 shows the bit usage and resulting number of sibling node ids in the current level for the described algorithm with a 4 bit level value. The 'X' in the bit pattern denotes the usable bits and the single '1' character is the delimiter bit which separates level value and subvalues.

# subvalues	bit pattern	number of ids		
0	OXXX	17		
1	10XX 1 XXXX	871		
2	110X 1 XXXX 1 XXXX	72583		
3	1110 1 XXXX 1 XXXX 1 XXXX	5844679		

Table 1. DLN Streaming algorithm: bit usage and resulting number of ids

This algorithm is very space-efficient. It produces short node-ids for irregularly structured documents compared to using a fixed number of bits for all nodes accommodating the maximal fan-out, and compared to the use of subvalues only if level value bits are exhausted.

With a small modification the algorithm can be enhanced to more efficiently use bit ranges. After a new subvalue is added and before the highest 0-bit is set to '1' the bit range of the new subvalue can be exploited for the next numbers. For instance between 0111 and 1000 1 0000 we have 0111 1 0001 to 0111 1 1111. So with one subvalue we can now label 86 sibling nodes instead of 71.

### 3.4 Variants of DLN

With the highly varying structure of XML documents with respect to depth (average, maximal) and fan-out and the possible parameters for DLN there is no setting which is best for all documents (cf. Section 4). However we can find variants tailored to important application cases which are differentiated by 1) streaming vs. DOM and 2) fixed vs. adjustable number of bits.

The *streaming DLN* uses the algorithm described in the preceding section. It is the most general approach and can be used for all kinds of XML data, especially data which can only be read sequentially. Since it has no knowledge about the number of siblings it uses a fixed number of bits for all level values and subvalues. With the finding in [15] that the distribution of element fan-out follows a power law, i.e. most elements have only few children, it might be advantageous to use smaller number of bits for level values than for subvalues.

If the number of siblings is known in advance (like with data represented in a DOM) one can use the DOM variant of DLN which adjusts the number of bits for the level value according to the number of siblings. However this simple approach might not be the best one. If an element has a large fan-out all children will use a level number with the same big number of bits. This has two disadvantages: 1) to calculate parent id or range of children ids one needs the length of the id and the number of bits used in the last level number, which consumes valuable space for each id, 2) if a node has to be inserted using subvalues the subvalues will use the same big number of bits.

To overcome the disadvantages of the simple DOM DLN we defined two variants. The first one, *fixed DOM DLN*, uses a small fixed number of bits for level values and subvalues and adds as many subvalues as needed to reach the necessary number of bits. Compared to streaming DLN it can use all bits resulting in shorter ids. Since we can calculate the id length and know the number of bits we can save on these values resulting in a partly better overall bit usage compared to simple DOM DLN.

The second variant, *restricted DOM DLN*, uses an adjustable number of bits for level values but restricts the maximum number of bits. If more bits are needed subvalues will be used like with the fixed DOM DLN. To calculate the id length we only need to know the length of the last level value and the position of the last 1 bit in the last level value or subvalue. Since we restricted the maximum number of bits only a few bits are needed. In Section 4.1 we give a quantitative comparison of the mentioned DLN variants.

### 4 Evaluation

We first evaluate the DLN variants with respect to the maximal and average id lengths for several large XML documents. We also compare the results with the ORDPATH numbering scheme. In 4.2, we present DLN performance results for insert and retrieval operations.

### 4.1 Comparing DLN variants and ORDPATH

Given the dependence of numbering schemes on the path lengths, an important quality measure is the maximal and average id length for a given document set. For our evaluation, we use several document collections from the XML data repository of the University of Washington and other publicly available documents. The set includes data-centric XML (mostly converted from other formats) and document-centric XML, e.g. Shakespeare's works, a novel and religion books. In Table 2 some metrics of the documents are given showing the different structural properties.

document	max depth	avg depth	max fan-out	avg fan-out	90% fan-out	#nodes
Nasa	8	5,5	2435	2,8	3	530528
Cities	4	3,6	364	5,0	5	21028
Dictionary	8	3,2	163826	3,9	8	1545406
Novel	4	3,9	75	26,9	75	220
Pop. Places	3	2,9	164045	14,0	13	2952811
Religion	6	4,8	289	25,1	44	48259
Shakespeare	6	4,8	434	5,5	10	179689
Sigmod	6	5,4	89	3,7	4	15263
Treebank	36	7,9	56384	2,3	4	2437667
WFB	7	4,9	260	4,1	9	347868
Courses	5	4,0	2112	4,2	7	66735

Table 2. Depth, fan-out and number of nodes of document collection

In Table 3 we compare the *maximum* and *average* length of a node id for the DLN variants and the two variants of the ORDPATH numbering scheme [13]. We added the last two here since they have some conceptual similarities with DLN. For a better overview we chose the fixed streaming DLN with 4 bits as a reference and show for the other columns how the id lengths differ from it. For instance, looking at the Nasa document we have a maximal id length of 64 bits for streaming DLN with 4 bits, while the maximum is reduced by up to 9 bits (to 55 bits) for DOM DLN and by 1 bit (to 63 bits) for ORDPATH. We used the enhanced algorithm for streaming DLN as described in Section 3.3.

	Streaming DLN								DOM DLN					ORDPATH					
	fixed	num	ber o	f bi	ts	less bits for level				sim	ple	fix	ted	restr.					
						number (LN)													
doc	4 bit	3 bit 5 bit		LN: 3 bit LN:			4 bit			4 bit		max		A		В			
						SV:	4 bit	SV:	5 bit					4	bıt				
Nasa	64 45,3	-1	-1,0	7	-0,9	2	-2,2	5	-0,1	-8	-3,0	0	-6,6	-9	-9,6	0	-3,0	-1	-6,1
Cities	39 31,1	4	-0,1	2	-0,3	1	-1,3	4	-0,8	-1	3,1	-5	-4,2	-5	-4,0	5	-0,6	7	0,1
Dict.	69 39,2	2	4,0	8	0,6	-1	0,1	1	-1,0	-13	-0,3	-5	-4,3	-12	-4,5	9	6,2	-6	-4,6
Novel	24 22,6	3	-1,6	5	3,8	6	-1,7	1	0,8	6	6,2	0	0,4	-2	-1,6	3	0,7	-1	-3,5
Pop. Plac.	44 39,6	3	3,4	-3	-1,7	2	-0,2	0	-0,6	-5	-0,9	-10	-6,3	-9	-5,3	7	5,2	-1	-3,9
Religion	54 33,7	-3	0,4	-1	2,1	-1	2,0	-1	1,8	-8	3,8	-10	1,3	-10	1,1	-7	-1,1	0	1,8
Shakesp.	44 31,8	3	-0,2	3	0,4	5	1,4	4	1,2	-2	3,0	-5	-3,7	-4	-4,4	1	-0,3	5	-0,7
Sigmod	44 34,3	3	0,6	9	5,2	9	1,3	3	1,7	0	3,5	0	-0,4	-2	-3,1	2	1,2	7	1,0
Treebank	199 59,6	-20	-1,4	34	4,1	-21	-3,8	-2	-1,2	-64	-9,4	-5	-6,6	-71	-16,8	-3	2,0	-57	-13,6
WFB	49 35,8	2	0,4	4	-0,7	0	0,8	4	0,6	-3	2,7	0	-1,3	-4	-2,0	3	0,1	5	-0,5
Courses	44 34,3	3	2,4	3	-0,2	0	1,5	-2	-1,5	-3	2,0	-5	-5,2	-7	-5,8	-2	-1,4	0	-2,1

Table 3. Maximum (with dark gray background) and average id length

Except for the Treebank data, the streaming DLN with 4 bits has good overall values especially compared to the 3 bits and 5 bits variants. Since the Treebank document is very deep and has a small fan-out the streaming DLN with a smaller number of bits per level value are better suited. The values for DOM DLN include the bits needed to calculate the id length which is why simple DOM DLN has not smaller values in every case. Generally the restricted DOM has the smallest values in



Fig. 4. Relational schema for XML mapping with DLN

nearly every case. The ORDPATH values are similar to the streaming DLN variants. Especially type B performs well in nearly all cases. However they cannot take advantage from knowledge about the number of siblings like the DOM DLN. In the case of the *average* node id length the streaming DLN with 4 bits again shows good overall values. Only the DOM DLN variants are superior, especially the restricted one.

#### 4.2 DLN performance for database operations

In order to evaluate the processing performance of DLN we executed several database operations influenced by the numbering scheme. All tasks run on an Intel Pentium III 800 processor with 512 MB main memory under Windows 2000 and the database system MySQL 4.0. We implemented a DLN prototype in Java which exports a proprietary API for data access and manipulation as well as a subset of XPath. Before we present the results we describe the database schema used in our DLN prototype.

### 4.2.1 Database Schema using the DLN Scheme

The XML to relational mapping scheme we chose is partly based on the DOM. In Fig. 4 the corresponding relational database schema is shown. Each node of the document tree from this model is stored as a data set in a node table. We use a separate table for attributes as well as for large text chunks. For increased query performance and reduced storage size the element and attribute names are indexed in a separate table. The parent and *rightSibling* attributes are used for fast evaluation of the child and sibling axes.

#### 4.2.2 Results

We selected three areas of interest for evaluating the DLN performance: 1) loading/inserting streamed data, 2) querying for documents or document fragments and 3) queries using the descendant axes.

Table 4 shows the insertion performance for bulkloading different kinds of documents and three insertion modes (deactivated indices, all indices activated except the fulltext index, all indices activated). Since XML data can have a highly varying element per kilobyte ratio and we store every element in a single row it won't be informative to give a MB/s value. Instead we measure insertion performance in nodes (elements, attributes and text nodes) per second. The selected documents capture data-

centric XML (Cities, Sigmod) as well as document-centric XML (Bible, XMach-1). The XMach-1 data is a set of 10.000 documents from the XML database benchmark XMach-1 [16]. One can see that even for a large number of documents the insertion performance scales well. As one would expect the number of nodes inserted per second drops with a lower nodes per kilobyte ratio. The exception of XMach-1 is a result from better memory handling with smaller file sizes and already running applications. The performance from the insertion with all indices activated can also be expected when an XML fragment is inserted into an existing document since no extra effort for renumbering is needed.

document #nodes		n/ŀB	Inserti	on (nodes pe	Reconstruction		
document	#HOUCS	II/ KD	w/o idx	w/o txtidx	with txtidx	(nodes per second)	
XMach-1	$2,2*10^{6}$	11	2.500	-	-	-	
Cities	$3,5*10^4$	44	2.380	1.729	1.699	10.310	
Sigmod	3,8*10 <sup>4</sup>	38	2.180	1.579	1.522	9.860	
Bible	$2,5*10^4$	8	1.899	1.405	1.152	8.525	

Table 4. Insertion and reconstruction performance

Retrieving documents or fragments with shredded data is normally time consuming because a large number of join and sort operations are needed. However with a preorder numbering scheme it can be reduced to a simple range query. Therefore we have good performance as shown in the reconstruction column of Table 4.

Queries using the descendant axes like a//b can be optimized as range queries using the node id of a and the maximum child id of a. The latter one can be calculated with a user defined function. We evaluated these queries on multiple documents and found that in most cases queries using the descendant axes are faster than queries with a complete path expression. For instance with the Sigmod document the query *//issue[.//author='Michael Stonebraker']]/volume* took 40 ms whereas */SigmodRecord/issue[articles/article/authors[author='Michael Stonebraker']]/volume* took 451 ms to complete (both queries needed 190 ms to build the XML result).

### 5 Conclusion

We have developed a new numbering scheme, called DLN, for generic structurecentered storage of XML data in relational database systems. The numbering scheme allows insert operations without the need for renumbering existing nodes. Furthermore it can be used for streamed data and benefits from additional structure information in order to reduce the id length.

We have shown parameterized variants of DLN which can be used to select an optimal numbering for a specific document structure. The performance evaluation of our prototypical DLN implementation demonstrated that inserting, retrieving and querying of XML data are efficiently supported by the numbering scheme.

Our numbering scheme constitutes an important building block for efficient generic XML data management in relational database systems. In future work we will

address open issues such as efficient relational access to the generically stored XML data and synchronized updates in multi-user environments.

# References

- Kuckelberg, A.; Krieger, R.: Efficient Structure Oriented Storage of XML Documents Using ORDBMS. In Bressan, S. et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 131-143, Springer-Verlag, 2003
- Shanmugasundaram, J.; Shekita, E. J.; Kiernan, J.; Krishnamurthy, R.; Viglas, S.; Naughton, J. F.; Tatarinov, I.: A General Techniques for Querying XML Documents using a Relational Database System. In SIGMOD Record 30(3), pp. 20-26, 2001
- 3. Florescu, D.; Kossmann, D.: Storing and Querying XML Data using an RDBMS. In IEEE Data Engineering Bulletin 22(3), 1999
- Schmidt, A.; Kersten, M. L.; Windhouwer, M.; Waas, F.: Efficient Relational Storage and Retrieval of XML Documents. In WebDB (Selected Papers) 2000, pp. 137-150, 2000
- 5. Bauer, M. G.; Ramsak, F.; Bayer, R.: Multidimensional Mapping and Indexing of XML. In Proc. of German database conference BTW 2003, pp. 305-323, 2003
- 6. Dietz, P. F.: Maintaining order in a linked list. In Proc. of the 14th Annual ACM Symposium on Theory of Computing, pp. 122-127, California, 1982
- Shimura, T.; Yoshikawa, M.; Uemura, S.: Storage and Retrieval of XML Documents using Object-Relational Databases. In Proc. of the 10th Intern. Conf. on Database and Expert Systems Applications (DEXA'99), LNCS 1677, Springer-Verlag, pp. 206-217, 1999
- 8. Li, Q.; Moon, B.: Indexing and Querying XML Data for Regular Path Expressions. In Proc. of the 27th VLDB Conf., Roma, Italy, 2001
- Chien, S.; Tsotras, V. J.; Zaniolo, C.; Zhang, D.: Storing and Querying Multiversion XML Documents using Durable Node Numbers. In Proc. of the Intern. Conf. on WISE, Japan, pp. 270-279, 2001
- 10. Lee, Y. K.; Yoo, S.; Yoon, K.; Berra, P. B.: Index Structures for Structured Documents. Proc. of the 1st ACM International Conference on Digital Libraries, pp. 91-99, 1996
- Kha, D. D.; Yoshikawa, M.; Uemura, S.: A Structural Numbering Scheme for XML Data. In Chaudhri, A. B. et al. (Eds.): EDBT 2002 Workshops, LNCS 2490, pp. 91-108, Springer-Verlag, 2002
- 12. Cohen, E.; Kaplan, H.; Milo, T.: Labeling Dynamic XML Trees. In Proc. of PODS 2002
- O'Neil, E.; O'Neil, P.; Pal, S.; Cseri, I.; Schaller, G.; Westbury, N.: ORDPATHs: Insert-Friendly XML Node Labels. ACM SIGMOD Industrial Track, 2004
- Tatarinov, I.; Viglas, S.; Beyer, K. S.; Shanmugasundaram, J.; Shekita, E. J.; Zhang, C.: Storing and querying ordered XML using a relational database system. In Proc. of SIG-MOD Conf., pp. 204-215, 2002
- Mignet, L.; Barbosa, D.; Veltri, P.: The XML Web: a First Study. In Proc. of the 12<sup>th</sup> Intern. WWW Conference, Budapest, 2003
- Böhme, T.; Rahm, E.: XMach-1: A Benchmark for XML Data Management. In Proc. of German database conference BTW 2001, pp. 264-273, Springer-Verlag, Berlin, 2001