

# XMLEase: A Novel Access- and Space-Efficiency Model for Maintaining XML Data in Relational Databases

A. Elçi

Department of Computer Engineering, and  
Internet Technologies Research Center, Eastern  
Mediterranean University, Famagusta, Mersin  
10, Turkey

B. Rahnama

Department of Computer Engineering, and  
Internet Technologies Research Center, Eastern  
Mediterranean University, Famagusta, Mersin  
10, Turkey

*Abstract - Representing XML data in terms of the rows and columns of a relational table is quite inefficient. Similarly, object-oriented databases cater for storage of objects but lack fast access. In these cases, a better technique is required to speedup representation and retrieval of hierarchical data in relational databases. This paper presents XMLEase, a new access- and store-efficient technique to keep XML data in relational databases. XMLEase keeps data node together with its successors down to a pre-defined level of hierarchy. This method has been implemented and evaluated; in comparison to other similar methods it is found to perform better. It shows a fair linear performance independent to the depth of hierarchy with superior data retrieval efficiency. Although it suffers of a bit of extra space usage for storing successor links, however this is much less than that of the other methods used for dealing with XML data. Present paper focuses on the access efficiency of the model in connection with run query processing. This method's best feature is its efficiency in operations to store and retrieve XML data requiring only one-run SQL query.*

Keywords: XML data, Relational database, hierarchical representation, Access and Space Efficiency

## 1.0 Introduction

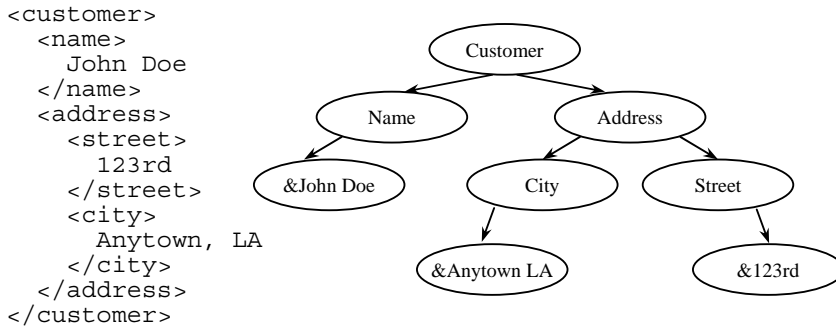
Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML [1]. XML data is in terms of a hierarchical data structure, hence the query languages such as XQuery, YATL, XML-QL and XML-GL were proposed to let the researchers easily write and parse the XML data[2], [3]. In order to facilitate the task of querying XML documents, efficient storage models for storing XML data in relational databases some methods such as Edge, Monet, XRel and XParent were proposed [4]. A comparison studied by Haifeng et al. [Haifeng et al. 2001] shows that the XParent is the best approach in terms of storage efficiency [4] [16] while its access efficiency is poor. This method scatters XML data into three tables, namely, “Text”, “Element”, and “Path”. In that case, to access the XML data efficiently one needs to “join” these three tables. DBXML [12], denormalized relational tables [22], and support of XML data within SQL codes [18] for processing of hierarchical data are common XML database facilities providing ease of dealing with XML data within relational tables. In addition simplified query based techniques such as RelaXML [13], XML data scheme [14] [15], could also help maintaining XML data. Moreover, processing methods for indexing XML data [17] [19] in order to raise the query efficiency such as search queries [20] are other area of interest. However, few of methods could be applied for both access and space efficiency models [16] [23]. Some mapping schemes [11] [14] [21] try to have both requirements together; nevertheless the complexity of model is increased.

In this paper, we examine a database schema called XMLEase as a new approach to XML data representation model. This approach regards an XML file as consisting of terms of hierarchical data structure, each node of which is made up of “Label” and “Data” fields. Each node is then converted to a record in a relational table. To remove the “join problem”, a node is stored in a table together with its redundant pointers to the root of the XML tree. This causes increase in space usage while affording higher access efficiency. This approach benefits from avoiding any requirement for data field processing of relational data as it’s in XParent and allows to run a single SQL query to store or retrieve the XML data.

The rest of the paper is arranged as follows. Section 2 discusses an XML data model in terms of hierarchical data modeling. In section 3 we mention the existing approaches for XML data in relational databases. In the subsequent section we introduce the proposed method XMLEase. In section 5 we discuss common queries and SQL syntax for storing or retrieving of XML data. Section 6 contains analytical and experimental results such that XMLEase is brilliant in both access and space efficiency. And finally, we conclude the paper with findings in section 7.

## 2.0 XML Data File

In this part we show a short XML data and its hierarchical representation. The XML data graph for the following XML document given in Fig. 1.



**Fig. 1.** A small XML Document and its corresponding data graph

In Fig. 1, the root “customer” is regarded as the 0th level of the XML tree (we start counting from the root), the “name” is in the 1st level of hierarchy, etc.

The existing approaches to XML data representation in relational databases are taken up in the following section. XMLEase is further introduced in Section 4.

## 3.0 The Existing Approaches

There are many models dealing with hierarchical data structures that may be utilized in representation of XML data in relational DB tables. Among more common ones, the “Nested Sets Model” [7], [8] nodes are numbered sequentially with two indices for bidirectional search queries. In the “Materialized Path Model” [5], [6], the “address” of a node is stored with it, starting from the root, in the form of a partially-concatenated string of integers. The mapping between XML and relational data are treated in [2], [10], and [14]. Structure-mapping-based and the more interesting one XML Data Mediator [9] methods are also in this category.

Edge, Monet, XRel, RelaXML, and XParent are five out of most existing approaches to store XML data in relational tables [3]. The access efficiency of Edge is better than that of other methods, because it employs a single normalized table. On the other hand, the XParent method based on Materialized Path Model [5], [6] shows the best space usage results. XParent uses numerical tables in connection with the XML data table in order to avoid text processing on the string data kept in the materialized path table [4] [23].

## 4.0 A Novel Method: XMLEase

XMLEase is a novel access-efficient method that introduces a way of converting an XML tree to RDBMS tables. To understand this method better, let’s suppose that redundant edges are introduced into the tree, so that each node is connected to each one of its ancestors, instead of just its parent. This makes it possible to fetch an XML tag data of a sub-class for a specific property (such as if we need all the name spaces in the third level of XML tree) using a single query, without needing a join operation. Thus, Table 1 (“xmltable”) displays the relational modeling of the given XML data and its represented graph given by Fig.1 in Section 2 above.

**Table 1.** XML data graph representation using XMLEase method

(Identifier)	(P1)	(P2)	(P3)
Customer	NULL	NULL	NULL
Name	Customer	NULL	NULL
Address	Customer	NULL	NULL
&John Doe	Name	Customer	NULL
State	Address	Customer	NULL
Street	Address	Customer	NULL
&Anytown, LA	State	Address	Customer
&123 <sup>rd</sup>	Street	Address	Customer

The “*Identifier*” column contains a name for intermediate nodes and the immediate data in the case of leaves of the XML tree in the example. Other columns maintain pointers to ancestors, such as, *P1* is the pointer to immediate parent, *P2* is the link to the grand parent, and so on. For example, the record for “Name” has “Customer” as *P1*, and “Null” for *P2* & *P3*. In this case, the depth of hierarchy supported by a relational database is limited by the number of parent columns. On the other hand, the bigger the number of ancestor columns, depending on the specific XML document, there will be more chances for “Null” pointers, thus wasting storage. It is up to the user to pre-determine the level of hierarchy in order to keep efficiency high vis-à-vis the nature of the XML files in consideration. The user may reset the level at any time.

Note that the scope of hierarchy in XMLease is limited to the supported number of parents (number of pointer columns; let’s call it “a layer”) rather than keeping the complete depth of all the nodes. In order to fetch the nodes at deeper levels than the supported level, the scheme is applied to the successive layers in turn and query results are joined. As far as the model requires a single run for each layer of hierarchy, it is also access (i.e., time) efficient. This is taken up in the next section. It is conceivable that XML data hardly ever has a high number of hierarchies. Choosing the supported level as, say 10, would require just a single layer, thus generally utilized without requiring joins or re-runs.

## 5.0 Access Efficiency in Query Processing

XMLease speeds up query processing for insertion, update, retrieval and deletion of data. This is due to its keeping parent pointers with entities. The sample queries given below may help show this result. The examples in this section refer to the relational table given above.

### 5.1 Data Insertion

In order to insert a new XML tag, say “GateNo”, as another property of “Address”, one can easily run the following query:

```
INSERT INTO xmltable VALUES ('GateNo', 'Address', 'Customer', NULL);
```

The entity <GateNo, Address, Customer, Null> will be inserted. Since the ancestors’ addresses are kept in the pointers of this entity, one does not need to update the parent nodes while inserting a node into an XML tree. Consequently, a parent’s field need not be updated in order to add a successor node. Thus, insertion is completed in single-run query.

### 5.2 Data Update

To update a data value in the relational table, one needs only to run a simple single-run query. For instance, to change the street value to “321st”:

```
UPDATE xmltable SET xmltable.Identifier = '&321st' WHERE  
xmltable.P1='Street' and xmltable.P2='Customer';
```

### 5.3 Data Deletion

In order to delete a tag, one can simply delete its specific record. Since the ancestors’ addresses are kept in the pointers, while deleting a node from an XML tree its ancestors’ nodes need not be updated. However, a node and its sub-tree can be removed with single-run query without requiring looping or recursion. This is done simply by inner select deletion if DBMS supports it, or by deletion of entities explicitly selected through parent pointer being the deleted tag.

### 5.4 Data Retrieval

With the XMLease table representation, the following two major queries need be covered:

a) ‘Fetching all the children of a node, say, “Customer” that are 2 levels down’ can be done with the following query:

```
SELECT Identifier FROM xmltable WHERE P2 = 'Customer';
```

b) To find all of the sub-tree of “Customer”, the following single query can be run:

```
SELECT Identifier FROM xmltable WHERE P1 = "Customer" or P2 = "Customer"
or P3 = "Customer";
```

As seen above, in both cases only a single-run query is required.

The examples given above help elucidate the access efficiency of XMLEase. In the following section, space efficiency of XMLEase will be shown; analytical formulation and experimental comparison against other similar methods will be displayed.

## 6.0 Space Efficiency

### 6.1 Analytical Results

Among all existing methods, the materialized path is superior in terms of access efficiency and space used to keep XML data. Therefore, the XMLEase method is compared against the materialized path.

Let's first express  $Us_{Ease}$ , the space used by XMLEase method. Let's denote the total number of nodes in an XML tree by  $N$ , the level of hierarchy by  $L$ , and the size of the record identifier by  $S$ , then:

$$Us_{Ease} = N \cdot (L + 1) \cdot S \quad (1)$$

Considering indices on the table, space used is expressible as follows, where  $Sp$  is size of pointer,  $n$  is the maximum possible number of nodes at each level, and  $Spi(L, n, i)$  is equal to the size of indices on each column for  $i^{th}$  level parent (non key attribute)

$$Spi(L, n, i) = \frac{n^{L-i+1} - 1}{n - 1} (S + Sp + n^i \cdot Sp) \quad (2)$$

Therefore, the space used taking indices into account is given below, where  $k$  is the supported level of hierarchy (i.e., number of parent columns,  $k \leq L$ ):

$$Usi_{Ease} = N \cdot (L + 1) \cdot S + \sum_{i=1}^k Spi(L, n, i) \quad (3)$$

Whereas the space usage for the materialized path is as follows if  $Ps$  is size of each path entity:

$$Us_{MP} = N \cdot (S + Ps) \quad (4)$$

For a complete tree, where  $S$  is size of each dot-separated integer entity,

$$Ps = \sum_{i=1}^L i \cdot S \cdot n^i \quad (5)$$

Then,

$$Us_{MP} = \left( \sum_{i=0}^L i \cdot S \cdot n^i \right) + N \cdot S \quad (6)$$

On the other hand, considering indices in the table,  $Nic$ , the size of index on key attribute is given as follows:

$$Nic = N \cdot (S + Sp) \quad (7)$$

As  $Usi_{MP}$  is given by the following,

$$Usi_{MP} = 2 \cdot \sum_{i=1}^L i \cdot S \cdot n^i + N \cdot S + N_{ic} + N \cdot Sp \quad (8)$$

Therefore, after inserting  $N_{ic}$  and factoring:

$$Usi_{MP} = 2 \cdot \sum_{i=1}^L i \cdot S \cdot n^i + 2 \cdot N \cdot (S + Sp) \quad (9)$$

The comparison of formulas (10) & (11) that do not include indices is straightforward: XMLEase is superior to the materialized path method. Let's consider the case with indices. In order to simplify the calculations, let's assume the size of each entity is one as follows:

$$Usi_{MP} = 2 \cdot \sum_{i=1}^L i \cdot S \cdot n^i + 2 \cdot N \cdot (1 + Sp); S = 1; \quad (12)$$

$$Spi(L, n, 1) = \frac{n^L}{n-1} \cdot (1 + Sp) + n \cdot Sp; N = \frac{n^{L+1} - 1}{n-1}$$

Thus,

$$Usi_{MP} = 2 \cdot \left[ \sum_{i=1}^L i \cdot n^i + (1 + Sp) \cdot \frac{n^{L+1} - 1}{n-1} \right] \quad (13)$$

And, to simplify the space usage formula for XMLEase considering indices, let's assume the supported level of hierarchy  $k$  (i.e., the number of parent columns, where  $k \leq L$ ) is taken as  $k = L$ :

$$Usi_{Ease} = N \cdot (L + 1) + \sum_{i=1}^k \left( \frac{n^L}{n-1} \cdot (1 + Sp) + n \cdot Sp \right) \quad (14)$$

To compare the space used by both methods, we do as follows:

$$\Delta Usi = Usi_{MP} - Usi_{Ease} \quad (15)$$

It can be shown that  $\Delta Usi$  is always positive [23]. Thus, XMLEase method uses less space than the materialized path method.

## 6.2 Experimental Results

In the experiment reported below, space usages for different methods are being compared. As the first test case, let us consider tables without taking indices into consideration. A simple program was written for working with MySQL DB in order to fill the relational tables by the same XML data. The figure given below displays the space used versus levels of hierarchy by different methods without considering indices.

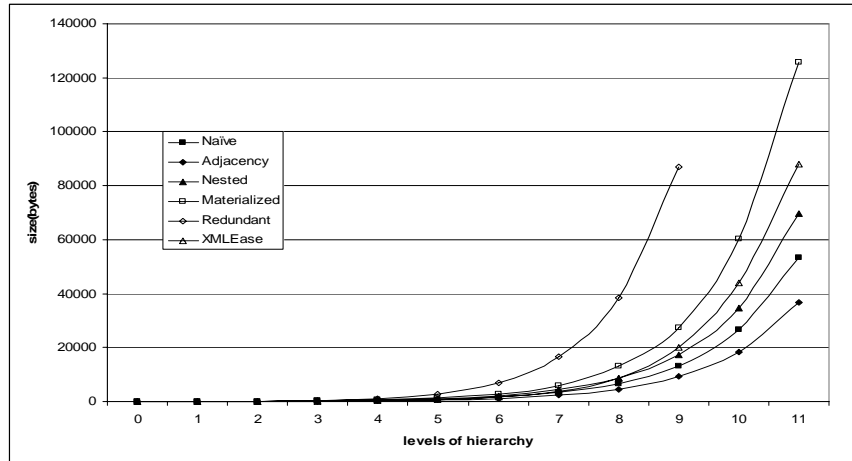


Fig. 2. Space used (size) versus levels of hierarchy by different methods without considering indices

It is clear that the experimental values follow the analytical estimations: XMLEase method uses less space than materialized path and redundant arcs methods. However, it consumes more space than naive, adjacency list and nested sets methods but these are not as access efficient as XMLEase.

If indices on the tables are taken into consideration, the experimental results show that XMLEase is better than all methods except the naïve one in agreement with analytical estimations. However, naïve method is access inefficient as already indicated.

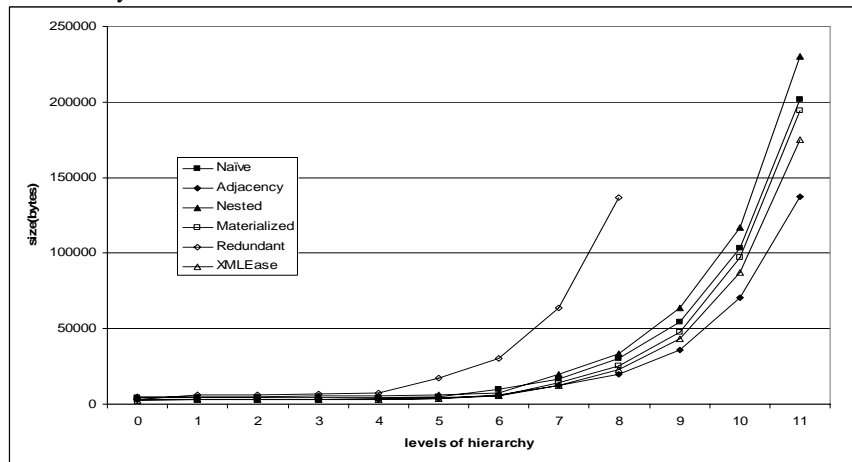


Fig. 3. Space used (size) versus levels of hierarchy by different methods considering indices

## 7.0 Conclusion

Representing XML data in terms of the rows and columns of a relational database table is quite inefficient. Similarly, object-oriented databases cater for storage of objects but lack fast access. In these cases, a better technique is required to speedup representation and retrieval of hierarchical data in relational databases. This paper presented XMLEase, a new access- and store-efficient technique to keep XML data in relational databases. Models of representing XML data in relational databases are considered; their performance is compared against that of XMLEase.

The best of the methods using existing models such as Nested Sets and the Materialized Path Models is the XParent. Yet, the XParent is known to suffer from low performance for path interpretation.

The XMLEase method introduces redundant links between a node and its ancestors so that the children of a given “Label” at any level below the node in the XML data file can be retrieved efficiently. The XMLEase places these links as pointers in separate columns. In this approach, the XML data tags refer to their ancestors instead of successors as represented by the XParent method. It is shown that the fundamental DB ML operations such as insert, update, delete, and retrieve are carried out efficiently with a single-run query by XMLEase. Addi-

tionally, deleting a node does not require updating the previously stored parents. The small extra space used for pointers appears to be the only drawback of this approach. Yet, wasted storage is much less than that of the other methods used for dealing with XML data. Furthermore, it is clear that the XML data servers value 'access efficiency' and speed more than they would mind using a little extra storage space.

Experimental results follow the analytical expectations. XMLEase method is more efficient than redundant arcs and materialized path methods in terms of space usage. It performs better than all others in the case of access efficiency.

## 8.0 References

1. W3C Organization, Extensible Markup Language, <http://www.w3.org/XML/>.
2. Katz, H., Chamberlin, D., Draper, D., XQuery from the Experts: A Guide to the W3C XML Query Language, Addison-Wesley Professional, 1st edition (2003).
3. Bonifati, A., Ceri, S., Comparative analysis of five XML query languages, vol. 29(1), SIGMOD Record, (2000) 68–79.
4. Jiang, H., Lu, H., Wang, W., Xu Yu, J., Path materialization revisited: an efficient storage model for XML data, Proceedings of the Thirteenth Australasian Conference on Database Technologies, Australian Computer Society, Inc., Melbourne, Victoria, Australia (2002) 85–94.
5. Celko, J., Trees in SQL, [http://www.intelligententerprise.com/001020/celko.jhtml?\\_requestid=123193](http://www.intelligententerprise.com/001020/celko.jhtml?_requestid=123193).
6. Tropashko, V., Nested Sets and Materialized Path, <http://www.dbazine.com/tropashko4.shtml>.
7. Celko, J., Trees and Hierarchies in SQL for Smarties, Morgan Kaufmann (2004).
8. Furnas, G., Zacks, J., Multitrees: Enriching and Reusing Hierarchical Structure, Human Factors in Computing Systems, Boston, Massachusetts USA, (1994), 330–336.
9. Bohannon, P., Freire, J., Roy, P., Simeon, J., From XML schema to relations: a cost-based approach to XML storage, Proceedings of the 18th International Conference on Data Engineering, IEEE Proceedings (2002) 64–75.
10. Chaudhuri, S., Chen, Z., Shim, K., Wu, Y., Storing XML (with XSD) in SQL databases: interplay of logical and physical designs, IEEE Transactions on Knowledge and Data Engineering, vol. 17(12), (2005), 1595–1609.
11. Maghamez, A., Hu, G., Multi-resolution indexing for XML data, 3rd ACIS International Conference on Software Engineering Research, Management and Applications, (2005), 206–211.
12. Tzvetkov, V., Xiong W., DBXML - Connecting XML with Relational Databases, CIT 2005, Proceedings of the 5th International Conference on Computer and Information Technology, (2005), 130–135.
13. Knudsen, S.U., Pedersen, T.B., Thomsen, C., Torp, K., RelaXML: bidirectional transfer between relational and XML data, Proceedings of the IDEAS 2005, 9th International Database Engineering and Application Symposium, (2005), 151–162.
14. Fujimoto, K., Dao Dinh Kha, Yoshikawa, M., Amagasa, T., A Mapping Scheme of XML Documents into Relational Databases using Schema-Based Path Identifiers, Proceedings of the WIRI '05, International Workshop on Web Information Retrieval and Integration, Proceedings (2005), 82–90.
15. Xing, G., Guo, J., Wang, R., Managing XML documents using RDBMS, Proceedings of the SNP/D/SAWN 2005, Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, and First ACIS International Workshop on Self-Assembling Wireless Networks, (2005), 186–191.
16. Qin, J., Zhao, S., Yang, S., Dou, W., Efficient storing well-formed XML documents using RDBMS, Proceedings of the 2005 International Conference on Services Systems and Services Management, vol.2, Proceedings of ICSSSM '05 (2005), 1075–1080.
17. Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, E., Naughton, J., Tatarinov, I., Special section on advanced XML data processing: A general technique for querying XML documents using a relational database system, ACM SIGMOD Record, vol.30(3), ACM Press (2001).
18. Nicola, M., Linden, B., Industrial session: XML support in relational system: Native XML support in DB2 universal database, Proceedings of the 31st International Conference on Very Large Data Bases VLDB '05, VLDB Endowment (2005).
19. Amer-Yahia, S., Du, F., Freire, J., XML processing: A comprehensive solution to the XML-to-relational mapping problem, Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management, ACM Press (2005).
20. Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C., Research sessions: XML I: Storing and querying ordered XML using a relational database system, Proceedings of the 2002 ACM SIGMOD international conference on Management of data SIGMOD '02, ACM Press (2002).
21. Weigel, F., Schulz, K., Meuss, H., XML data management and web discovery: Exploiting native XML indexing techniques for XML retrieval in relational database systems, WIDM '05, Proceedings of the 7th annual ACM international workshop on Web information and data management, ACM Press (2005).
22. Balmin, A., Papakonstantinou, Y., Storing and querying XML data using denormalized relational databases, The VLDB Journal, The International Journal on Very Large Data Bases, vol.14(1) Springer-Verlag New York, (2005).
23. Rahnama, B., A New Method to Represent Hierarchical Data in Relational Database Systems, Master of Science in Computer Engineering, Eastern Mediterranean University (2005)