Storing and Querying XML Data in the Nested Relational Sequence Database System

Ho Lam, LAU and Wilfred NG

Department of Computer Science
The Hong Kong University of Science and Technology
{lauhl, wilfred}@cs.ust.hk

Abstract. We developed the Nested Relational Sequence Database System (NRSD System), which is built upon the Nested Relational Sequence Model (NRSM). The NRSM eliminates a substantial amount of redundancy embedded in XML documents and preserves the order of XML data. In this paper, we demonstrate that the storing and querying of XML data are desirable over an NRS relation, which we incorporate an index system into the NRSD System. We introduce a set of NRS operations which allow users to manipulate the XML data effectively and demonstrate how to use NRS operations to query the XML data stored in the NRSD System. Our experimental results confirm that the NRSD System substantially reduces the storage size of data. We also study the performance of the select and project operations having path expressions at different nested levels.

1 Introduction

We have proposed the Nested Relational Sequence Model (NRSM) in [3, 4], which is an extension of the well-established Nested Relational Data Model (NRDM) [2, 7] in order to cater for nesting structure and node ordering of XML data. The NRSM supports composite and multi-valued attributes, which are essential for representing hierarchically structured data objects in XML documents. In addition, the NRSM extends the NRDM to support ordering of XML data elements, which allows nested tuple sequences to be defined in an NRS relation.

We developed the NRS Database System (NRSD System) [3, 4], which is built upon the NRSM. An important feature of the system is that XML data that share the same path expression can be merged together and store in the same *index table*, and thus, eliminates a substantial amount of redundancy in XML documents. Another feature of the NRSD System is that it preserves the following three types of order: the value, sibling and ancestor orders as illustrated in Figure 1. In the sequel we will explain how these features are preserved in the NRSD System by using an indexing schemes. We also present the experimental results concerning the resources needed for storing XML documents and the performance of the *select* and *project* operations having various path expressions.

2 The Nested Relational Sequence Data Model

Like the NRDM, the NRSM supports composite and multi-valued attributes, which are essential for representing hierarchically structured information such as XML data. In the NRSM, XML data are stored in the NRS relations. The representation of NRS relations is similar to the Nested Tables (NTs) of the NRDM, which contains multiple rows and columns. The NRS relations inherit the desirable features of the NTs, such as minimizing redundancy of XML data in a document and having more expressive power than conventional relations. XML documents are represented as *merged data trees* (MDT), where leaf nodes

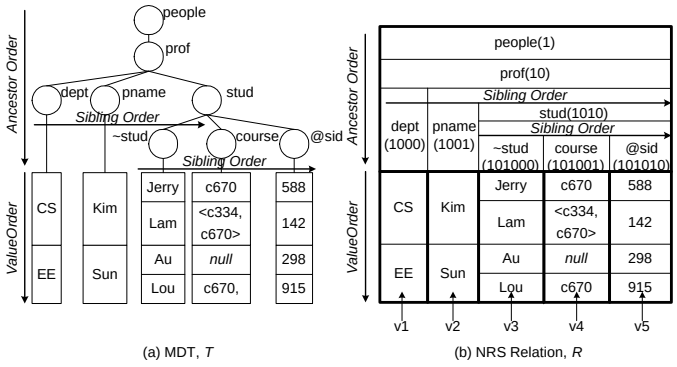

Fig. 1. (a)A MDT and (b)its corresponding NRS relation

are sequences of data values, called the *data nodes*, and non-leaf nodes are the labels of the XML tags and attributes, called the *label nodes*. The MDTs are different from the DOM trees in [9] in the way that information with the same tags are grouped together and the values are stored under the same label node in a MDT.

The NRSM incorporates three types of order: (1) the value order resulting from the sequence of data elements in a data node; (2) the sibling order resulting from the left to right sides for those label nodes which share the same parent; (3) the ancestor order resulting from the tree levels of the label nodes. Figure 1 portrays the three types of orders in a MDT.

The NRSM preserves the original structure of XML documents, in general, they can be retrieved from NRS relations without loss of information. Descriptive information such as *comments* and *processing instructions* can also be stored in an NRS relation. The proposed mapping algorithms [3, 4] between XML documents and NRS relations are straightforward enough to implement on Oracle, as adopted in our NRSD System. If an XML document is not associated with a DTD, we extract an approximated DTD from the input XML document. The extracted DTD provides a basis for constructing the corresponding NRS schema.

A benefit resulting from this approach is that if several XML documents of similar structures are combined into a single NRS relation, we are able to generate an optimized DTD for them. Figure 2 demonstrates the mapping between an XML

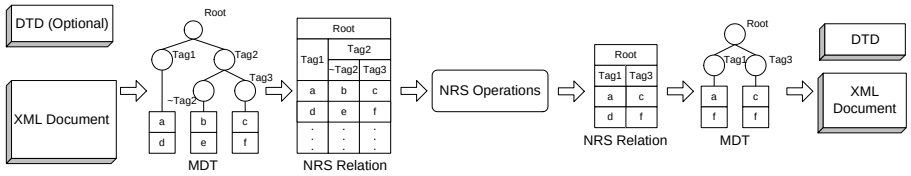

Fig. 2. Mapping an XML document into an NRS relation and retrieve XML data using NRS operations

document and an NRS relation. After we have mapped the XML document into a MDT, we can generate an NRS relation. The XML semantics and the order of the document structure are preserved in both MDTs and NRS relations and the mapping between MDTs and NRS relations is reversible. After an XML document is transformed into its corresponding NRS relation, we can then apply a sequence of NRS operations on the relations.

3 Formulating Queries in the NRSD System

The target of the proposed XML query languages, such as XQuery [9] and XML-QL [1], is to query XML documents and they are not designed for the information stored in XML databases. We introduce a set of operations for the NRSM, called the *NRS operations*, which is a combination of the refinements of existing relation algebra [7, 2] together with some newly defined operations such as the *rearrange* and *swap* operations.

An *NRS expression*, denoted by *QueryExpr*, is defined by the following BNF syntax:

$QueryExpr ::= Op_{[Arugment]}(Range)$
$Op ::= \pi \mid \sigma \mid \eta \mid \mu \mid \uplus \mid \oplus \mid \ominus \mid \odot \mid \omega \mid \kappa \mid \chi \mid \varsigma \mid \Sigma \mid \alpha \mid \perp \mid \top \mid \cup \mid - \mid \times \mid \bowtie$
$PathExpr ::= PathExpr/Tag \mid PathExpr//Tag$
$Tag ::= tag\ labels \mid *$
$Argument ::= PathExpr \mid (ArgumentList) \mid Cast(Argument)$
$ArgumentList ::= PathExpr \mid ArgumentList, PathExpr$
$Cast ::= INT \mid CHAR \mid DATE$
$Range ::= PathExpr \mid BETWEEN\ PathExpr\ AND\ PathExpr$

Since XML data are considered as strings in a document, we use the casting functions, $Cast()$ to convert the values from strings to other data types such as integers or dates. Our query expressions do not have the *For-Each* clause,

instead, the query expressions loop through each tuple specified by the *Range* argument. In the *Range* argument, the expressions *BETWEEN* and *AND* can be used to limit the range of a query in a declarative way. A set of NRS operations for manipulating the XML data stored in the NRSM is introduced in our previous work in [3, 4] . The operations are classified into the following six categories: (1) Nesting operations, (2) Ordering operations, (3) Basic operations, (4) Structure operations, (5) Binary operations and (6) Aggregate operations. Examples are now given to demonstrate how to perform queries using the NRS operations.

3.1 Query Examples

Q_1. Return the names of the students who take the course "c670".

The query Q_1 can be formulated by the XQuery expressions as follows:

```
FOR $s IN document("sample.xml")people/prof/stud
WHERE $s/course = "c670"
RETURN $s/text()
```

We first transform the XML document "*sample.xml*" and its DTD into the corresponding NRS relation, R, which is shown in Figure 1 (b). Then we perform the following sequence of query expressions:

1. $S \leftarrow \pi_{(/people/prof/stud)}(R)$
2. $T \leftarrow \sigma_{(/stud/course="c670")}(S)$
3. $R_{result} \leftarrow \pi_{(/stud/\ stud)}(T)$

stud*		
~stud	course*	@sid
Jerry	c670	588
Lam	<c334, c670>	142
Au	-	298
Lou	c670	915

S

stud*		
~stud	course*	@sid
Jerry	c670	588
Lam	<c334, c670>	142
Lou	c670	915

T

stud*
~stud
Jerry
Lam
Lou

R_{result}

Fig. 3. Intermediate NRS relations for answering the query Q_1

For the sake of clarity, we use the intermediate NRS relations, S and T, to show the results in each processing step, as shown in Figure 3. We first perform the "π" operation "*people/prof/stud*" over R. Then we perform the "σ" operation according to the condition "$/stud/course =$ "c670"" over S. Finally, we perform the "π" operation "*/stud/ stud*" over T, which generates the required results R_{result} and transform it into the corresponding valid XML document as follows:

```
<!ELEMENT stud(#PCDATA)>
<stud>Jerry</stud>
<stud>Lam</stud>
<stud>Lou</stud>
```

Q_2. Create a new XML document with the format given below, where X is the number of professors, Y is the number of students in the department, and Z is the name of the department:

```
<university>
   <dept numP="X" numS="Y">Z</dept>
</university>
```

The query Q_2 is formulated as the following sequence of NRS operations:

1. $S \leftarrow \uplus_{(university(dept(\sim dept, @numP, @numS)}$;
2. $T \leftarrow \oplus_{(university/dept/\sim dept=R/people/prof/dept)}(S)$;
3. $R_{result} \leftarrow \oplus_{(university/dept/@numP=\zeta(/people/prof/pname)(U))}(T)$
   $R_{result} \leftarrow \oplus_{(university/dept/@numS=\zeta(/people/prof/stud/\sim stud)(U))}(R_{result})$;
   $where\ U = \sigma_{(R/people/prof/dept=university/dept)}(R)$.

First, we create a new NRS relation with the schema $university(dept(\sim dept, @numP, @numS))$ using the "$\uplus$" operation. Second, we insert the value of *dept* corresponding to the *dept* of R. Final, we use the "ζ" operation to find out the number of professors and students in the *dept* and insert it into R_{result} using the "$\oplus$" operations. The intermediate NRS relations, S and T, are shown in Figure 4.

University		
Dept		
~dept	@numP	@numS

S

University		
Dept		
~dept	@numP	@numS
CS		
EE		

T

University		
Dept		
~dept	@numP	@numS
CS	1	2
EE	1	2

R_{result}

Fig. 4. Intermediate NRS relations for answering the query Q_2

3.2 Translating XQuery into NRS operations

The flexible and structured facilities of XQuery [9] have been recognized as effective way to query XML data. Currently, we are able to translate the XQuery of the form "FOR-WHERE-RETURN", which is a fundamental form of XQuery expressions, into sequences of NRS operations. We now use the following XQuery expression to illustrate the translation.

```
FOR $b1 IN path1,
    $b2 IN path2,
    ...
    $bn IN pathn,
WHERE condition1, condition2,..., conditionm
RETURN
<tag1>
{
   <tag2>$b1/resultPath1</tag2>
   <tag3>$b2/resultPath2</tag3>
   ...
}
</tag1>
```

Since a query may involve several XML documents, we need to map them into their corresponding NRS relations, R_1, R_2, ..., R_n, before we apply the NRS operations. After the mapping, the following query expressions are performed:

1. $S_1 \leftarrow \pi_{[path_1]}(R_1), S_2 \leftarrow \pi_{[path_2]}(R_2), ..., S_n \leftarrow \pi_{[path_n]}(R_n)$
2. $T_1 \leftarrow \sigma_{[condition_1]}(S_{i1}), T_2 \leftarrow \sigma_{[condition_2]}(S_{i2}), ..., T_m \leftarrow \sigma_{[condition_m]}(S_{im})$
3. $U \leftarrow \uplus_{(tag1(tag2,tag3))}$
4. $R_{result} \leftarrow \oplus_{(tag2=T1/path3,tag3=T2/path4,...)}(U)$

First, we translate the XQuery expression "*FOR b_i IN $path_i$*" into a sequence of "π" operation and store the result into intermediate NRS relations S_1, S_2, ..., S_n. Then, based on the conditions stated in the clauses "*WHERE* $condition_1$, $condition_2$, ..., $condition_m$", we perform the "σ" operation on the involved intermediate NRS relations S_1, S_2, ..., S_n and store the required data in the intermediate NRS relations T_1, T_2, ..., T_m. For example, if "$condition_i$" is equal to "$\$b/name =$ "*abc*"", we know that S_1 is involved and we perform the "σ" operation on S_1. The last step is to return the results in the specified format stated in the "*RETURN*" expression. To achieve this, we perform the "$\uplus$" operation to create a new NRS relation using the schema corresponding to the one stated in the XQuery "*RETURN*" expression and perform the "$\oplus$" operations to insert the corresponding results into the NRS relation. In our current implementation, we can only handle structure of at most two nested levels.

4 Developing the Indexing Scheme in the NRSD System

An NRS relation is stored as a set of *index tables (ITs)* in the NRSD System. The schema of the NRS relation is mapped as a *global index table (GIT)* and the data value are mapped as various *value index tables (VITs)*. The MDT and NRS relation R_{sample} shown in Figure 1 is mapped into the ITs $= \{g, v_1, \ldots, v_s\}$ shown in Figure 5.

In the NRSD System, we capture different kinds of order by representing order using index values, which further reduce the storage size. We use the

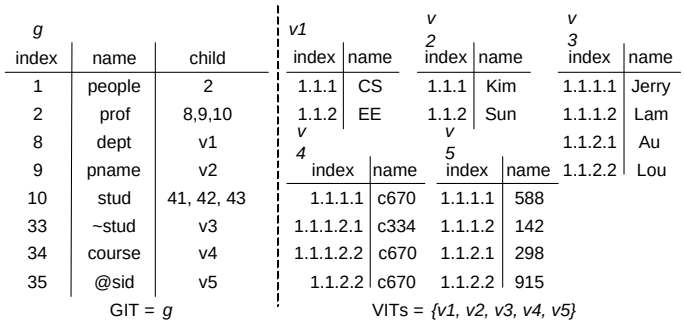

Fig. 5. The ITs corresponding to R_{sample}

prefix matching and the *dot-notation indexing strategies* to handle the orders of XML data in the NRSD System.

Prefix matching indexing is used in the GIT. We assign the root attribute with index 1 and for each child attribute, we assign $log_2 k$ bits to represent its sibling order, where k is the number of its sibling nodes, and concatenate it after the index of its parent. For example, in Figure 1, attribute *stud*, whose index is 1010, has three children, we assign two bits for each of its child attributes. The indexes of its children are 101000(40), 101001(41) and 101010(42) as shown in g of Figure 5. We apply the *longest bit match algorithm* [6] to search for the parent of a sequence of nodes. The essential idea of the algorithm is to find out the *longest common prefix* of the indexes in an incremental manner. In the VITs, the indexes are assigned by using the dot notation indexing scheme. For example, the index for the *stud Lam* in $v3$ is 1.1.1.2. From the GIT, the path expression of $v3$ is "$people/prof/stud/{\sim}stud$". It represents the fact that *Lam* is the second *stud* of the first $prof$, who is the first *people* in the first XML document.

Basically, the implementation of the NRS operations in the NRSD System is translated into the following sequence of actions. (1) Lookup the corresponding child attributes from the GIT by tracing the path expressions. (2) Perform queries over the corresponding VITs stated in the child attribute of the GIT. (3) Return the data required. For example, if we perform the "π" operation on *stud*, $\pi_{[/prof/stud]}(people)$. We lookup the GIT for the path $people/prof/stud$ and know that it has three corresponding VITs with indexes: 40, 41 and 42. Therefore, we join the values from these three VITs according to their longest common prefix and return them into tuples. For example, the two *course* of *stud Lam* are assigned with indexes 1.1.1.2.1 and 1.1.1.2.2, the longest common prefix is 1.1.1.2, which is in the same tuple of *stud Lam*. In order to avoid heavy overhead of the machine arising from referencing the GIT extensively. An in-memory tree is built for storing the global information for performing quick reference. In general, the size of the GIT is less then 0.1% of a given XML document and it does not impose any significant burden on the main memory.

5 Preliminary Experimental Results

In order to show the effectiveness of the NRSD System, we have been running some experiments using real life *DBLP* XML data [8] on the NRSD System. The data has maximum twelve child tags per element and four nesting levels in DBLP XML documents. The experiments are conducted on a computer of *Pentium III 550MHz* with 256MB RAM and 30GB hard disk.

5.1 Results for Storing XML Documents in the NRSD System

In the experiment, we load XML documents having different sizes into the NRSD System. Table 1 shows the results of our experiments, from which we can check that the table space required for storing an XML document is approximately 85% of its original size. With the growth of document size, the number of ITs and the size of the GIT become stable. It is due to the fact that the NRSD System groups and stores the tags which share the same path expression, and represents orders using indexes. We remark that the size reduction is obtained

Table 1. Experimental results of the NRSD System with different XML document sizes

Size of NRS schemas (bytes)	Number of all tables in Oracle DBMS	Table space including schema and table overheads (kilobytes)	Input XML file size (kilobytes)	Percentage of table space required
414	23	1,987	2,404	82.65%
1368	61	8,159	9,318	87.56%
1380	63	12,399	14,251	87.00%
1404	66	16,722	18,817	88.87%
1425	67	24,076	28.791	83.62%

without performing any compression on the database. This work can serve as a starting point for applying existing XML data compression technology [5] on the NRS relations. We also emphasize that we are able to formulate queries in the NRSD System by using a set of algebraic operators, which is difficult to perform in a compressed domain. We are improving the grouping algorithm and trying to further decrease the table space for storing XML data in the NRSD System. The data size reduction in the NRSD System is useful in practice for exporting and exchanging XML database objects on the Web.

5.2 Results for Performing Select and Project Queries

In order to study the query performance of the NRSD System, we run six queries (E_1 to E_6) over three sample DBLP XML documents of sizes 8.8MB, 24MB and 36MB. The objective of the experiment is to test how path expressions affect the performance of the "π" and "σ" operations in the NRSD System, since they

are fundamental in NRS query expressions. By studying their performance, we can understand how to optimize more complex queries. The queries used in the experiment are formulated as follows.

$E_1 = \pi_{/dblp/inproceedings/author}(R)$
$E_2 = \pi_{//author}(R)$
$E_3 = \pi_{//inproceedings/}(R)$
$E_4 = \sigma_{/dblp/inproceedings/year='2000'}(R)$
$E_5 = \sigma_{//year='2000'}(R)$
$E_6 = \sigma_{[//year='2000',//author='Sun']}(R)$

E_1 evaluates the performance of the "π" operation with full path expression. E_2 evaluates the performance of the "π" operation on the recursive path expression ($//$). E_3 evaluates the performance of the "π" operation that require several outer join operations. E_4 evaluates the "σ" operation operation with a condition specified with full path expression. E_5 evaluates the "σ" operation with a condition specified with recursive path expression. E_6 evaluates the "σ" operation with multiple conditions specified with recursive path expressions. We plot

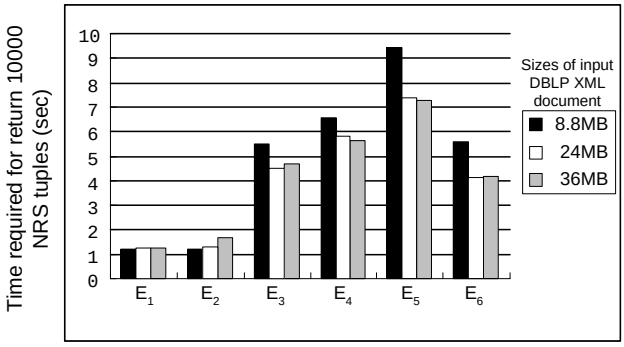

Fig. 6. Performance of the queries E_1 - E_6

the average time required for the NRSD System to output 10000 NRS tuples in Figure 6. From E_1 and E_2, we can see that the number of nesting levels does not affect the running time of the queries used in the experiments. This is because the structure of the XML document stored as an in-memory tree saves the time for looking up the corresponding indexes from the GIT from the database. The NRSD System requires more time to return the results for queries E_3 - E_6. This is due to the fact that the NRS relation are stored in several VITs in the underlying Oracle DBMS, join operations are performed to retrieve the data from the NRSD System , which urges us on further studying the issues of query optimization of the join operation. For the performance of the select operation, we first return the index of tuples that satisfied the conditions and they return the tuple that match the index. Therefore, it require much more time than the project

operation. We notice that from E_4 to E_6, when the document size is small, the ratio of the selecting overhead is high, since the number of tuples returned is small compared with large document. One interesting observation is that, in E_6, we have two conditions and the number of returned tuples is limited, although the fraction of selecting overhead is still high, but the number of join operations required are smaller and results can be returned much more efficiently.

6 Conclusions and Future Work

In this paper, we introduce the storage and manipulation of XML data in the NRSM. The NRSM is useful in transforming and restructuring the XML relations to other forms. It also allows users to manipulate the ancestor order in the schema, the sibling order of tags and the value order in the tuples. We also demonstrate with examples how to formulate XML queries in terms of NRS operations and to translate XQuery into a sequence of NRS operations. We believe that a more user-friendly and high level language similar to standard SQL can be developed on the basis of NRS operations.

We discuss the experimental results of the NRSD System and show that the table space required for storing XML data with the NRSD System is significantly less than the size of the original XML documents, which is mainly due to the fact that NRSM eliminates the redundant data from the XML documents. We believe that compression can be further applied on the NRSM such that the storage size can be further reduced while preserving the querying facilities. We are improving the grouping algorithm on data nodes which have the same labels. We are still investigating the performance of other NRS operations apart from the project and select operations over the NRS databases.

References

1. A. Detsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu. *A Query Language for XML.* In: http://www.research.att.com/ mff/files/final.html, (1998).
2. H. Kitagawa and T. L. Kunii. *The Unnormalized Relational Data Model.* Springer-Verlag, (1989).
3. H. L. Lau and W. Ng. *Querying XML Data Based on Nested Relational Sequence Model.* In: Poster Proc. of WWW, (2002).
4. H. L. Lau and W. Ng. *The Development of Nested Relational Sequence Model to Support XML Databases.* In: Proc. of the International Conference on Information and Knowledge Engineering (IKE'02), pp. 374-380, (2002).
5. H. Liefke and D. Suciu, *XMILL: An efficient compressor for XML data.* In: Proc. of SIGMOD, vol. 29, pp. 153-164, (2000).
6. M. A. Weiss. *Data Structures and Algorithm Analysis in C++.* Addison Wesley, (1999).
7. M. Levene. *The Nested University Relation Database Model.* Springer-Verlag, (1992).
8. M. Ley. *Digital Bibliography & Library Project.* In: http://dblp.uni-trier.de/, (2002).
9. World Wide Web Consortium. In: http://www.w3.org/, (2002).