Managing XML by 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 have established the Nested Relational Sequence Model (NRSM), which is an extension of the Nested Relational Data Model in order to handle XML data. The NRSM incorporates the *value order*, the *sibling order* and the *ancestor order*. In this paper, we demonstrate NRS operations by examples and illustrate how XML queries can be formulated. We discuss the work related to implementing the NRSD System and present our preliminary results concerning the use of the system. We confirm the benefits of reducing the data size in NRSD.

1 Introduction

XML is becoming a standard format for representing and exchanging data on the World-Wide-Web. With the growth of electronic commerce, the quantity of XML documents on the Web has rapidly increased. To cope with the large amount of XML documents, we need a DBMS that is efficient enough to handle the storage, management and retrieval of XML documents. This depends on establishing an effective data model in a formal manner. A data model also serves as a foundation for future XML DBMSs development.

There are some inadequacies in the existing approaches for mapping XML documents into common DBMSs. First, the hierarchical structure of XML does not easily fit into the relational data model. Second, the complex nested tags and multi-value features of XML documents cause the normalization problem. This requires extra storage space and may even introduce errors when updating the data. Third, in some existing approaches like [1, 2], information about a complex object is often scattered over many relations, leading to a non-trivial correspondence between a real-world object and its database representation. Fourth, handling orders in RDBMSs may lead to many additional columns for each relation, thus, increasing the storage cost. Therefore, we have proposed the Nested Relational Sequence Model (NRSM), which is an extension of the Nested Relational Data Model (NRDM) [4, 8] in order to cater for the nesting structure and node ordering of XML documents.

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 NRS

relations are similar to the Nested Tables (NTs) of the NRDM, they contain multiple rows and columns. The NRS relations inherit the desirable features of the NTs, such as minimizing data redundancy and having more expressive power; they are also extended to support ordering of XML data by using sequences. Since ordering is an important feature of XML, nested tuple sequences are allowed in NRS relations.

Fig. 1. The Merged Data Tree and NRS relation showing three types of orders

The NRSM incorporates three types of ordering as follows: (1) *value order*, the order on data values; (2) *sibling order*, the order of tags which share the same parent; and (3) *ancestor order*, the hierarchical order between parent and child tags. Figure 1 portrays the three types of orders in a MDT: *tuple 1* for *Peter Kim* has the value order prior to *tuple 2* for *Ken Li*, which in turn, has the value order prior to *tuple 3* for *Sam Ho*. In the schema level, the ancestor order is from the top to the bottom, *Document* has the ancestor order prior to *student* and *student* has the ancestor order prior to *name* and so on. On level 2 of the schema level, it shows the three children of *student*. The sibling order is $\langle \text{program}, \text{name}, \text{sid} \rangle$. Note that the sibling order only applies to the attributes which share the same parent. On level 3, the attribute $\sim\text{program}$ has the sibling order over @mode , but it does not have the sibling order over the attributes *firstname* or *lastname*, since they have different parents.

An important feature in our model is that any XML data sharing the same path expression can be collapsed into the same node and be regarded as a sequence of data elements. This shows the benefit of using nested data, which eliminates a substantial amount of redundancy in XML documents.

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 and are regarded as data nodes. The proposed method for mapping between XML documents and NRS relations is straightforward enough

to implement on top of many practical DBMSs such as Oracle. If an XML document is not associated with a DTD, then we extract an approximated DTD from the XML data to provide 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.

Fig. 2. Mapping an XML document to an NRS relation and retrieving under NRS operations

Figure 2 demonstrates the mapping between an XML document and an NRS relation. We represent the document as a *merged data tree* (MDT), where leaf nodes 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*.

XML documents are not necessarily associated with DTDs, therefore, two simple and efficient algorithms to map XML documents into MDTs are proposed. After we have mapped XML documents into a MDT, we can generate an NRS relation. An NRS relation allows nesting of data and maintains the order of the values. The XML semantics and the order of the document structure are preserved in both MDTs and NRS relations. The mapping between MDTs and NRS relations is reversible. After an XML document is transformed into its corresponding NRS relation, we can apply a sequence of NRS operations on the relation to formulate useful queries.

We introduce a set of operations called the NRS operations by examples. The NRS operations are combinations of the refinements of existing relation algebra together with some newly defined operations, such as the *swap* and *rearrange* operations, to allow users to manipulate the structure and contents of the NRS relations. An important feature of our query language is that it preserves the nesting and ordering of original XML data in the output answer. The output of NRS operations is always an order-preserving NRS relation, which can be converted back to XML with our proposed retrieval algorithm. The details of the NRS operations can be found in [5, 6]. Herein, our discussion is to focus on the issue of how to implement a database system based on the NRSM. It is a middleware which acts between the user and the DBMSs based on the NRSM as the data model in order to help the user perform queries on the XML data.

The rest of the chapter is organized as follows: Section 2 discusses the storage and retrieval of XML data in the NRSM. Section 3 highlights the main features of NRS operations with examples. Section 4 describes the architecture of the

implementation of the NRS database system (NRSD System) and presents the preliminary experimental results. Finally, we give our conclusions in Section 5.

2 Storing and Retrieving XML Data in NRSM

Throughout this section, the XML document and DTD shown in Figure 3 are used to illustrate some concepts developed in the NRSM. This XML document contains information about the professors in a university. Each professor (*prof*) comprises information about his department (*dept*), name (*name*) and student (*stud*). Each student is given a student id (*sid*) for identification and they can take any number of courses (*course*) during their studies.

<pre> <!ELEMENT people(prof+)> <!ELEMENT prof(dept, pname), stud*> <!ELEMENT dept(#PCDATA)> <!ELEMENT pname(#PCDATA)> <!ELEMENT stud(#PCDATA, course*)> <!ATTLIST stud sid CDATA #REQUIRED> <!ELEMENT course(#PCDATA)> <people> <prof> <dept>CS</dept> <pname>Kim</pname> <stud sid="588">Jerry <course>c670</course> </stud> <stud sid="142">Lam <course>c334</course> </pre>	<pre> <course>c670</course> </stud> </prof> <prof> <dept>EE</dept> <pname>Sun</pname> <stud sid="298">Au</stud> <stud sid="915">Lou <course>c670</course> </stud> <stud sid="611">Ray <course>c630</course> <course>c334</course> </stud> </prof> </people> </pre>
---	--

Fig. 3. The sample DTD (sample.dtd) and XML (sample.xml)

We now give the definition of a merged data tree, which is essentially a tree representation of an XML document in the NRSM.

Definition 1 Merged Data Tree A merged data tree (MDT), T is defined as $T = \langle r, D, L \rangle$, where r is the root, D is a set of data nodes and L is a set of label nodes.

By representing XML documents as MDTs, XML data that share the same path expression can be merged together and stored under the same label node, and thus, eliminating a substantial amount of redundancy. It also preserves the value, sibling and ancestor orders of XML data. Figures 4(a) shows a MDT, T , which corresponds to the DTD and the XML document shown in Figure 3. The root node of T is the node *people* at Level 0, the level value of nodes increments from the top to the bottom as shown in Figure 4(a). Circles are used to represent

label nodes, while rectangles are used to represent data nodes. From now on we simplify “ $\langle a \rangle$ ” (a sequence having one atomic value) as “ a ”.

Fig. 4. The (a) MDT and (b) NRS relation for sample.xml and sample.dtd

Definition 2 (Nested Relational Sequence Relation) An Nested Relational Sequence Relation (an NRS Relation), R , is defined by $R = \langle N, S, O \rangle$, where N , S and O denotes the name, the schema and the occurrence of R respectively. $S = \{a_1, a_2, \dots, a_i\}$ is equal to a set of attributes, a_i . $O = \{t_1, t_2, \dots, t_i\}$ is equal to a sequence of tuples, t_i over S . We use the notation $O[a_i] = \{t_1[a_i], t_2[a_i], \dots, t_i[a_i]\}$ to represent a sequence of atomic value in O which is corresponding to the attribute a_i .

Figure 4(b) shows the table which represents an NRS relation, which is corresponding to the DTD and the XML document in Figure 3. The name N for this NRS relation is *people*, it is the same as the top-level element of the sample XML document. The upper part of the table is the *schema* S and the lower part of the table is the *occurrences* O . Compared with an XML document, the schema can be viewed as a sequence of labels of XML elements and attributes, while the occurrences can be viewed as a sequence of data values.

In the NRSM, three kinds of orderings are maintained; they are the *value*, *sibling* and *ancestor* orders. The value order is the order of tuples in the occurrence level. It follows the sequence of data in the original XML document from top to bottom. For example in Figure 4(b), we can check that t_1 *Kim* has a value order prior to t_2 *Sun*”.

Both the sibling order and the ancestor order are the orders of attributes at the schema level. The sibling order is the order of attributes under the same parent from left to right. In XML documents, the sibling order is equivalent to the order of tags at the same nesting level with the same parent. If tag A appears before tag B in an XML document, A comes before B in the sibling order. In

Figure 4(b), under the attribute *prof*+, the attribute $\wedge(\textit{dept}, \textit{pname})$ has the sibling order prior to the attribute *stud**. For the child attribute of *stud**, their sibling order is $< \sim \textit{stud}, \textit{course}*, @\textit{sid} >$.

The ancestor order is the hierarchical order between the parent and child attributes at the schema level. The parent attribute are prior to their child attributes in terms of the ancestor order. Ancestor order represents the hierarchical order of the nested tags in XML documents. This order is essential to XML data, especially for locating a tag using path expression. For example, in Figure 4(b), we have to write the path expression `“/people/prof+/stud*/course*”` to locate the tag *course* but not the path expression `“/people/stud*/prof+/course*”`.

In XML documents, tags are organized in a hierarchical manner and they are allowed to be nested to any level. The nesting tags represent the parent-child relationships. Since tags are allowed to share the same label, these multi-valued data are represented as a sequence of data under the same column of the NRS relation. In the NRSM, nesting of data is allowed within another sequence, each item in the sequence can be another sequence of data. For example, in the sample XML document in Figure 3, each *prof* may supervise several *stud* and each *stud* may take several *course*. In other words, a *prof* may have a sequence of *stud*, in which, another sequences of *course* are contained. In Figure 4(b), the *course* taken by the *stud* of *prof* “*Kim*” is represented as the sequence $< \text{c670}, < \text{c334}, \text{c670} > >$. This sequence contains another sequence $< \text{c334}, \text{c670} >$, because the *stud* “*Lam*” has taken two courses.

Within the NRSM, we are able to transform a MDT into an NRS relation and vice versa [5, 6]. 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, such as *nest* and *unnest* operations on the relations to manipulate the XML data. Figure 4(b) shows the NRS relations corresponding to the MDTs in Figure 4(a). For details of the mapping, the reader can refer to [5, 6].

3 NRS Operations and Running Examples

We define a set of algebraic operations on NRSM [5, 6], which is employed to formulate a query over an NRS relation. These operations enable users to retrieve XML information from NRS relations. The output result of these operations is an NRS relation, taking one or more NRS relations as input. These operations are introduced in [5, 6]. We now use some examples to show how to perform queries using NRS operations. In this section, the example queries are operating on the sample NRS relation as shown in Figure 4(b). It is the NRS relation corresponding to the XML document and its DTD as shown in Figures 3.

(Q₁) List the names of the students who are taking the course “c670”. This query can be expressed in XQuery 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 has been shown in Figure 4(b). Then we perform the following sequence of operations:

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

For the sake of clarity, we use the temporary NRS relations, S and T , for storing the results at each step. In the first step, we perform a projection on “*people/prof/stud*” over R . Then we perform the selection according to the condition “*/stud/course = “c670”*” over S . Finally, we perform a projection on “*/stud/ stud*” over T , which generates the required results R_{result} . The temporary NRS relations S and T for this query are shown in Figure 5.

S:	suid*		
	~stud	course*	@sid
	Jerry	c670	588
	Lam	c334	142
	Au	-	298
	Lou	c670	915
	Ray	c630	611
		c334	

T:	suid*		
	~stud	course*	@sid
	Jerry	c670	588
	Lam	c334	142
	Lou	c670	915
		c670	

R_{result} :	suid*	
	~stud	
	Jerry	
	Lam	
	Lou	

Fig. 5. Generated NRS relations for answering the query Q_1

We then transform the resulting NRS relation by using its corresponding DTD. The conformed XML document can be represented 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 the name of the department:

```
<university>
  <dept numProf="X" numStud="Y">Z</dept>
</university>
```

First, we create a new NRS relation named *university* with the given schema *university* (*dept* ($\sim dept$, $@numProf$, $@numStud$)) using the operation $New(\uplus)$. Then we insert the value of *dept* corresponding to the *dept* of R . Each *dept* has two attributes, *numProf* and *numStud*. Thus, we use the aggregate function “*Count*(ς)” to find out the number of professors and students in the *dept*. The operations are shown below and the corresponding NRS relations are shown in Figure 6.

1. $S \leftarrow \uplus(\text{university}(\text{dept}(\sim\text{dept}, @\text{numProf}, @\text{numStud})))$;
2. $S' \leftarrow \oplus(\text{university}/\text{dept}/\sim\text{dept}=\text{R}/\text{people}/\text{prof}/\text{dept})(S)$;
For each data value under “university/dept/” :
3. $R_{\text{result}} \leftarrow \oplus(\text{university}/\text{dept}/@numProf=\text{S}(/people/prof/pname)(T)),$
 $\text{university}/\text{dept}/@numStud=\text{S}(/people/prof/stud/\sim\text{stud})(T))(S')$,
where $T = \sigma_{(R/people/prof/dept=\text{university}/dept)}(R)$.

S:	university		
	dept		
	~dept	@numProf	@numStud

S'	university		
	dept		
	~dept	@numProf	@numStud
	CS		
	EE		

R _{result} :	university		
	dept		
	~dept	@numProf	@numStud
	CS	1	2
	EE	1	3

Fig. 6. Generated NRS relations for the query Q_2 .

The resulting NRS relation and its corresponding DTD and XML document can be represented as follows:

```
<university>
  <dept numProf="1" numStud="2">CS</dept>
  <dept numProf="1" numStud="3">EE</dept>
</university>
```

4 Implementation of the NRSD System

In this section, we discuss the architecture, the interface design and other implementational details of the NRSD System and then we give some experimental results to illustrate that the system is desirable.

The NRSM model is deployed by adopting the usual client-server strategy. Figure 7 shows the architecture of the NRSD System. The system acts as a middleware between the users and the DBMS. The server is responsible for translating MDTs into NRS relations, which are managed by the underlying Oracle DBMS [14]. The client is responsible for (1) translating the output results of NRS operations into XML documents in order to answer querying from users and (2) dealing with the requests of different XML display formats and providing guidance for users when imposing queries. There are three main components in

Fig. 7. The architecture of the NRSD System.

the NRSD System: (1) the *Query Guidance and Display Facilities* (QGDF), (2) the *Query Modulator* (QM), and (3) the *NRS Transformer* (NT). The QGDF

is responsible for handling the interaction and display facilities of the NRS System. It provides a simple and user-friendly interface to help users perform queries on the XML data. The QM translates the queries and controls the mapping between XML documents and MDTs. The NT unit performs the mapping between MDTs into the NRS relations. The main function of the NT is to map an NRS relation into its corresponding set of relations in the underlying Oracle DBMS. The flow of a query follows the steps listed below:

1. Before starting a query, the relational tables stored in Oracle DBMS are transformed into a MDT by the NT and send to the QM.
2. The QM passes the MDT to the QGDF. The QGDF constructs the content of the interface and displays the MDT to the user.
3. The system is ready for users to initiate a query.
4. For each NRS operation that the user performs, the QGDF collects the required parameters from the user interface and sends them to the QM.
5. The QM updates the MDT and communicates with the NT about the updated data. Finally, the QM returns the querying result to the QGDF for display.
6. Step 4 is repeated if the user requests to perform another NRS operation.

We have implemented the NRS client interface by using *Java™ 2 Platform Standard Edition v1.4.0* [11] and *Simple API for XML (SAX)* [3] to help users initiate XML queries. The SAX parser provides an event-driven and serial-access mechanism for accessing XML documents. We choose SAX as an XML parser in the NRS System because of its high speed and less memory-intensive mechanism when parsing an XML document. We depict the user interface of the

Fig. 8. The interface of the NRS System.

Query Guidance in Figure 8. The interface consists of three main components: (1) a list of NRS operations on the left, (2) a *main panel* in the middle, and (3) a

text field at the bottom. In the main panel, there are two tabs which allow users to change between the view of the NRS relation and the corresponding XML document. The main panel is divided into two halves: the upper half is the *tree panel*, which is an area displaying the nesting structure of the NRS relation; the lower half is the *relation panel*, which is an area showing the data of each column. In the tree panel, different symbols are used to represent different types of tags. For example, the symbol “*e*” denotes an element, the symbol “@” denotes an attribute and the symbol “*t*” denotes text data. Aided by the interface given by Figure 8, we are able to initiate a query described by the following steps.

1. Select the *NRS Relation* tag from the main panel.
2. Identify the schema of the column from the tree panel. Then the relation panel displays the corresponding part of the NRS relation.
3. Select the attribute in the relation panel. The path of the selected sub-schema is displayed in the text field on the bottom of the interface.
4. Finally, click on a NRS operation button in the list in order to perform the query.

DBMS supports object data types such as a *collections of objects* and *nested tables* [10]. These object data types are useful for deploying an NRS relation, since they can support ordered collections of structured objects from the implementation point of view. The object types provided by *Oracle8i* are similar to the class mechanism supported by *Java* and *C++*. The following example shows how an object type is specified in order to implement the *prof+* object in our sample NRS relation:

```

1. CREATE TYPE DeptPname AS OBJECT
(
  dept   VARCHAR(10),
  pname  VARCHAR(20)
);
2. CREATE TYPE Courses AS VARRAY(99) OF VARCHAR(4);
3. CREATE TYPE Student AS OBJECT
(
  stud   VARCHAR(20),
  course Courses,
  sid    VARCHAR(3)
);
4. CREATE TYPE Prof AS OBJECT
(
  dp      DeptPname
  stud    Student,
);
5. CREATE TYPE Prof_Table AS TABLE OF Prof;
6. CREATE TABLE people
(
  professor Prof_Tables
) NESTED TABLE professor STORE AS professor_table;
```

In the statements 1 - 4, a new type *courses* is created instead of just using *VARCHAR*, since *course** in the NRS relation may be an empty sequence. In the statements 5 - 6, we create a storage table *professor_table*, which is associated with the nested table column *professor*. The column serves as the storage for all the corresponding nested data.

We have been running some experiments using real life data on the NRSD System in order to show the effectiveness of the system. All queries are conducted on a *Pentium III 550MHz* computer with 256Mb RAM and 30Gb hard disk. For the input source, we borrow parts of the XML document from the *Digital Bibliography & Library Project* (DBLP) [9]. The DBLP maintains a server that provides bibliographic information on major computer science journals and proceedings. In the DBLP XML document, the maximum number of child tags per element is twelve and the deepest nesting level is four. In the experiment, we test the NRSD System by loading XML documents of different size. Table 1 shows the results of our experiment. We can see that the table spaces required for storing the XML documents is approximately 85% of the original size of the documents. With the growth of size, the number of tables and size of schema become steady. It is due to the fact that NRSM collapses data of the same tags label into the same data nodes, resulting in very minimal redundancy.

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%

The result of size reduction is obtained without performing any compression on the database, this can serve as a starting point for further XML data compression. Note that one fundamental difference between using NRS relations and imposing usual XML compression is that we are able to query in NRSM based on a set of algebraic operations, 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 of NRS relations in the NRSD System. The benefit of data size reduction is useful for exporting the XML data from one database system to another.

5 Conclusions

We introduced the NRSM and demonstrated with examples how to formulate XML queries in terms of NRS operations. We have discussed the implementational issues of the NRSD System which uses Java and SAX API on top of the

Oracle8i. We also presented the design of an interface using Java Swing Classes to guide the user to initiate queries. The NRSD System is a middleware system between the users and the underlying relational DBMS, it allows users to perform queries and operate on the structure of the XML data. The preliminary results shows 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 are further improving the NRSM in several areas. First, as the results of our experiments are obtained without any compression on data, we believe that a compression algorithm can be applied to the NRSM such that the storage size can be further reduced but we still preserve the querying facilities. Second, we are improving the grouping algorithm on data nodes which have the same labels. Third, we are working on the translation of XQuery into NRS operations. Finally, we are still carrying out experiments concerning querying over the NRS databases, we then will be more clear about the performance of the NRS operations in the NRSD System.

References

1. D. Beech, A. Malhotra and M. Rys. *A Formal Data Model and Algebra for XML*. Communication to the W3C, (1999).
2. D. Florescu and D. Kossman. *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*. In: Proc. of VLDB99, (1999).
3. D. Megginson. *Simple API for XML*. In: <http://www.saxproject.org/>, (2002).
4. H. Kitagawa and T. L. Kunii. *The Unnormalized Relational Data Model. For Office Form Processor Design*. Springer-Verlag, (1989).
5. H. L. Lau and W. Ng, *Querying XML Data Based on Nested Relational Sequence Model*. In: Proc. of Poster Track of WWW, (2002).
6. H. L. Lau and W. Ng, *The Development of Nested Relational Sequence Model to Support XML Databases*. In: Proc. of IKE'02, pp. 374-380, (2002).
7. M. A. Weiss, *Data Structures and Algorithm Analysis in C++*. 2nd edition, Addison-Wesley, (1999).
8. M. Levene. *The Nested University Relation Database Model*. Springer-Verlag, vol 595, (1992).
9. M. Ley. *Digital Bibliography & Library Project*. In: <http://dblp.uni-trier.de/>, (2002).
10. Oracle Corporation. *Oracle9i*. In: <http://www.oracle.com/>, (2003).
11. Sun Microsystems, Inc.. *Java 2 Platform, Standard Edition*. In: <http://java.sun.com/j2se/>, (2003).
12. W. Ng. *Maintaining Consistency of Integrated XML Trees*. International Conference on Web-Age Information Management WAIM'2002. Lecture Notes in Computer Science Vol. 2419, Beijing, China, pages 145 -157, (2002).