STORING SEMISTRUCTURED DATA INTO RELATIONAL DATABASE USING REFERENCE RELATIONSHIP SCHEME

B. M. Monjurul Alom, Frans Henskens and Michael Hannaford
School of Electrical Engineering & Computer Science, University of Newcastle, Callaghan, NSW 2308, Australia

Keywords: XML, Reference Relationship, RDBMS, FCL, PT, CRT.

Abstract: The most dominant data format for data processing on the Internet is the semistructured data form termed XML. XML data has no fixed schema; it evolved, and is self describing which results in management difficulties compared to, for example, relational data. This paper presents a reference relationship scheme that encompasses parent and child reference relations to store XML data in a relational view, and that provides improved of storage performance. We present an analytical analysis that compares the scheme with other standard methods of conversion from XML to relational forms. A relational to XML data conversion algorithm that translates the relational data into original XML data form is also presented.

1 INTRODUCTION

Semistructured data is becoming more and more prevalent for use in performing simple integration of data from multiple sources (Abiteboul, Quass et al. 1996). By semistructured, we mean that although the data may have some structure, this structure is not as rigid, regular, or complete as the structure required by traditional database management systems (Abiteboul 1997). The emergence of XML (Harold 2004), which is a data format for semi-structured data, has increased the use of semistructured data, assisted by the fact that attribute names are stored with the data itself, making it self-describing (Deutsch, Fernandez et al. 1999).

XML is the dominant data exchange format for Internet-based business applications. It is also used as the data format for automated tasks such as information extraction, natural language processing, and data mining (Senellart and Abiteboul 2007). When in XML form, data is neither table-oriented as in a relational database, nor is it strictly typed as in an object database (Abiteboul 1997). Rather, XML data comprises hierarchies that have no fixed schema. While XML form supports Internet transport and certain data processing tasks, it causes issues for other common activities such as querying and updating. Techniques exist for querying XML data (e.g. Lorel (Abiteboul, Quass et al. 1996), UnQL(Buneman, Davidson et al. 1996), XQuery (Pal, Cseri et al. 2005), XML-QL(Deutsch, Fernandez et al.), XPath(Harding, Li et al. 2003)); however these are more complicated to use and often less efficient than queries on the equivalent relational data using languages such as SQL. The promising approach of building XML database management systems above underlying Relational Data Base Management Systems (RDBMS) is described in (Florescu and Kossman, 1999; Shanmugasadaram, Tufte et al., 1999; Shanmugasadaram, Kiernan et al., 2001; Du, Amer et al., 2004; Pal and Cseri, 2004; Balmin and Papkonstatinou, 2005; Josifovskil, Fontoural et al., 2005). These systems represent the data in relational form for processing that works better in that form, and similarly represents the same data in XML form when that is more suitable. Techniques are required to translate the data between forms when necessary, alternately the data is stored in both forms simultaneously. In the quest to achieve query efficiency, three questions have arisen (Balmin and Papakonstantinou 2005):

1. How can XML data be stored in an equivalent RDBMS?
2. How is an XML query translated into an equivalent SQL query?
3. How is the result of an SQL query translated back into an XML result?

This paper addresses the issue of converting data in XML form into an equivalent relational database that preserves any parent/child or other relationships

M. Monjurul Alom B., Henskens F. and Hannaford M. (2008).
STORING SEMISTRUCTURED DATA INTO RELATIONAL DATABASE USING REFERENCE RELATIONSHIP SCHEME.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 118-126
DOI: 10.5220/0001875501180126

implicit in the XML hierarchy. We present a reference relationship scheme in which the XML data (including all element paths and attributes) are stored in a more space-efficient way than is provided by other existing schemes.

The paper also addresses the issue of translation of data from the RDBMS into equivalent XML documents *without loss of any information*. In particular, translations are achieved without loss of attributes and element paths so that an XML document can be translated into relational form, and then back into XML form with the distinction between element paths and attributes intact.

The remainder of this paper is organized as follows: Related work is described in section 2. The new parent child relationship structure is presented in section 3. Analysis of storage requirements, the translation algorithm for conversion from RDBMS to XML form, and the estimated query time is described in section 4. The paper concludes with a discussion and final remarks in section 5.

2 RELATED WORK

The Object Exchange Model (OEM) defines a way of representing XML data (Abiteboul 1997). An instance of OEM can be thought of as a graph, with objects as the vertices and element paths described using labels on the edges. Each object has a unique object identifier (oid). (Abiteboul 1997) also addresses issues to do with querying and reconstructing semistructured data. An RDBMS may be used as a basis for storing and querying XML data (Florescu and Kossman 1999). The XML document is viewed as an ordered and labelled directed graph. A node in the graph represents each XML element; the node is labelled with the *oid* of the XML element. Element-sub-element relationships are represented by edges in the graph and labelled by the name of the sub-element. The order of sub-elements is defined by ordering outgoing edges from nodes in the graph. Values (e.g. Strings) in an XML document are represented as leaves in the graph. All edges of the graph are stored in a relational table called the edge table and all the values (represented as leaves) are stored in separate value tables. While the format assists in performance of XML queries, this graphical form does not differentiate between element paths and attributes, or between element paths and references. Therefore this form of representation is a simplification, and some information may be lost. As a consequence it may be impossible to exactly reconstruct an original XML document from the relational data form.

XQuery supports XML views of Relational Data (Shanmugasadaram, Kiernan et al. 2001), providing a general framework for processing arbitrarily complex queries. The query language provides a view composition mechanism that eliminates the construction of all XML fragments, and an intensive computation that reduces an XQuery query to SQL for efficiency of RDBMS manipulation.

ShreX (Du, Amer et al. 2004) provides generic (mapping-independent) functions for loading shredded documents into relations and for translating XML queries into SQL. In this approach, the annotation processor parses an pre-annotated XML schema, checks the validity of the mappings and creates the corresponding relational schema.

Storing and querying XML data using de-normalized relational databases is described in (Balmin and Papakonstantinou 2005), which elaborates a formal framework for XML schema-driven decomposition that encompass de-normalized tables and binary-coded XML fragments. The key performance focus of this approach is the response time for delivering the first results of a query. At present this approach does not work on more complex queries, because the schema model is based on directed acyclic graphs (DAGs). It is expected that the technique will be improved when it is further developed to be based on arbitrary graphs.

XML data can be stably stored as a byte sequence (BLOB) in columns of tables to support the XML model (Pal, Cseri et al. 2004). This form of storage introduces new challenges for query processing. So-called ORDPATH is used to preserve structural fidelity, and to allow insertion of nodes anywhere in the XML tree without the need for re-labelling existing nodes.

XISS/R is a system based on an extended pre-order numbering scheme, which captures the nesting structure of XML data and provides the opportunity for storage and query processing that is independent of the particular structure of the data (Harding, Li et al. 2003). The system includes a web-based user interface which enables stored documents to be queried via a query language named Xpath. The user interface utilizes the Xpath query engine, which automatically translates Xpath queries into more efficient SQL statements.

Document Type Descriptors (DTDs) may be used as a tool in converting XML into Relational database form (Shanmugasadaram, Krishnamurthy et al. 2001). After the desired relational schema for storing XML documents is defined, an XML

shredder object, that can accept an XML document and shred it into rows, is used to populate the tables of the generated relational schema. A reconstructed XML view can be produced from the schema (Shanmugasadaram, Krishnamurthy et al. 2001). While this technique is less space efficient than others, the reconstructed XML view is identical to the original.

A mapping scheme between semistructured data model and the relational data model, expressed in a query language called STORED (Semistructured to Relational Data) is described in (Deutsch, Fernandez et al. 1999). When a semistructured data instance is given, a STORED mapping can be generated automatically using data-mining techniques. A relational schema is chosen, then the STORED mapping translates the semistructured data instance into that schema. The mapping is always lossless: parts of the semistructured data that do not fit the schema are stored in an "overflow" graph. STORED is more restrictive than other query languages for semistructured data; it doesn't have joins or regular path expressions.

3 FRAMEWORK OF REFERENCE RELATIONSHIP SCHEME (XML TO RDBMS)

Well formed XML data (i.e. in which end tags match start tags) may be used as source data for conversion from XML into RDBMS form. In this new reference relationship scheme, each distinct child (sub element path) of the root path (from XML Document) is represented by a separate relational table called a *parent table*. Each XML document is represented by a separate database. The root path name used in the XML document becomes the name of this database. The children (sub-element and attributes) of the sub-element path (of the root element) are assigned the name of the attributes (i.e. the *column* name) in the parent table, and the content of each distinct child (sub-element or attribute) is the same as the content of each distinct tuple in the parent table. If there is no sub-element Id in the XML document, the presented method creates a distinct Id for each tuple in the Relational Table. If the distinct children (sub-element paths) of the root path are represented by *SE* and the children (element path and attributes) of *SE* are represented by $C_{hl}SE$, the relational schema of the parent table is *PTSchema (Id, all* $C_{hl}SE$ *).*

If any $C_{hl}SE$ consists of sub-elements rather than of direct values, the scheme creates a reference for that particular attribute in the parent table, and a new table (called the Child Reference table) using that reference name is created for all types of ($C_{hl}SE$) children. The children (attributes, sub-element paths and the descendants of $C_{hl}SE$) of $C_{hl}SE$ form the attributes in the Child Reference relation. In the Child Reference relation there is a column named 'Parent Id' that is used to maintain the relationship with the parent table (in which 'Parent Id' is the primary key). An ancestor column (termed *FCL*) is included in the child table to store information that can later be used in reconstruction of the source XML document. Hence the schema of the child reference relation is *CRTSchema(Parent Id, FCL (Flag column), all children of* $C_{hl}SE$*).* If it is found that any attribute in the child table has no direct value (i.e. it consists of nested element(s)), the scheme creates a pnull for that particular attribute rather than creating a new table.

During the reversion process from RDBMS form to XML, this pnull attribute could become a parent element of any attribute except parent Id. Each parent relation can have at most one child reference table or relation. To distinguish between sub-element and attributes, the system creates an attribute dictionary for all attributes in the XML data. This dictionary records the name of the attribute, its value, its parent Id and an parent/child indicator. Thus the general schema for the attribute dictionary is *ADSchema(Parent Id, Name of attribute, value of the attribute, Relation type).* In general, when XML data is converted to relational data, there is no difference between attributes and sub-elements because they are both stored as attributes in the relational table. When it is necessary to reconvert the relational data to XML, it is necessary to distinguish between sub-elements and attributes.

The attribute dictionary is important in achieving this distinction. A null value for Parent Id in the attribute dictionary indicates the existence of the attribute in the parent table rather than in the child table, since the parent table has primary keys but no Parent Ids. Thus it is not always necessary to check the table type in the dictionary. The algorithm is given in Figure 1. To understand the algorithm the following data structures are necessary:

- *PT*: Parent Table.
- *CRT*: Represents the child reference table of the corresponding parent table.

- *null*: It is used in both PT and Child Table, for those attributes that have no nested element path and no value in XML.
- *FCL*: Flag Column used to store ancestor information.
- *Pnull*: Is used for particular attributes in the Child table which have nested sub elements instead of direct values.
- *SE*: Sub element Path of root tag (from XML document).
- $C_{hl}SE$: Child of SE.
- C_{hl}: Child.
- $A_{tt}SE$: Attribute of SE.
- A_{tt} : Attribute.
- *NSE*: Nested SE [If any $C_{hl}SE$ has C_{hl} (element path) which has also C_{hl} (element path) then C_{hl} of $C_{hl}SE$ is called NSE **]**.
- $C_{hl}NSE$: Child of Nested SE.
- $A_{tt}NSE$: Attribute of Nested SE.
- $RefA_{tt}$: Reference Attribute.

```
Algorithm XML_to_RDBMS()
Begin
    Identify all the distinct SE from
    original XML doc and create PT
    according to each distinct SE;
    Identify the distinct ChlSE and
    AttSE    from XML doc and create
    separate column for each AttSE and
    ChlSE in the PT;
If there is no Id in XML document
create a column in PT for id as a
primary key
  Begin
    Store all the contents of each AttSE
    and ChlSE to each tuple of the
    corresponding column in PT;
  End;
If (any AttSE and ChlSE) has no value
and no nested path
  Begin
    Put a null to the corresponding
    tuple of AttSE and ChlSE in PT;
  End;
If (any ChlSE ) has NSE then
  Begin
     Put a reference CRT according to
     its PT name to the tuple of that
     ChlSE column and create a new table
     according to that reference CRT
     name;
     Create distinct columns for all
     the Chl and Att of NSE  in the CRT;
     Create a FCL to keep information
     of different ChlSE for which the
     CRT is created;
     Create a column to store the
     parent Id from PT;
     Put a pnull to the corresponding
     tuple of specific column of any
     NSE and ChlNSE (which has no direct
     value);
  End; // if any ChlSE
    Create a dictionary for all
    attributes from XML document (as
    well as from PT and CRT), with
    their names, values, Parent Id and
    relation type;
End. //main
```

Figure 1: Algorithm XML to RDBMS.

3.1 Explanation of the Translation Scheme

The relational database FamilyInfo (comprising Tables 1, 2 and 3) has been created from the XML document shown in Figure 2. Personal (Table 1), CRTpersonal (Table 2) and Attribute Dictionary (Table 3) are the relations of the FamilyInfo database. In the child reference table (Table 2) the Parent Id is the same as the primary key in the parent table (Table 1). In Table 1, S_S_NO (social security number) is the primary key; in the child reference table (Table 2) it is represented as a Parent Id which provides the ability to connect with the parent table as well as to enable re-creation of the exact relationship between child and parent or ancestor.

```
</FamilyInfo>
      <Personal>  <S_S_No =1, age=33 >
      <address> Met st </address>
      <Sex> Male <\sex>
      <Son>
      <Personal> <Sex=Male>
      <address> Union St </address>
      <S_S_No> 3 </S_S_No>
      <Status>well </Status>
      <Job>Nurse </Job>
      </Personal>
      </Son>
      <Daughter> <Sex=Female>
      <Personal>
      <S_S_No>4</S_S_No>
      <address> Nevil Av  </address>
      <Status> Disable <\Status>
      </Personal>
      </Daughter>
      </Personal>
      <Personal><S_S_N0=2, age=21 >
      <Fname> Turner </Fname>
      <Job> Teacher </Job>
      <Address> Rax st </Address>
      <Sex>Female </Sex>
      </Personal>
</FamilyInfo>
```

Figure 2: Example of XML Document with nested elements.

The XML document depicted in Figure 2 shows that Son and Daughter are two $C_{hl}SE$ that have no direct values, rather they contain nested sub-elements of type Personal (which does have children (address,

S_S_NO, Status, Job)). This structure is expressed in relational form as shown in tables 1 to 3. Table 1 includes columns for Son and Daughter; the reference identifier CRTPersonal is inserted as the value in that column for the first tuple. This value identifies the name of the child reference relation as CRTPersonal; the relation itself is shown in Table 2.

Table 1: Personal.

S_S_NO	address	Sex	Son	Daughter	Fname	Job	age
1	Met st	Male	CRTPersonal	CRTPersonal	null	null	33
2	Rax st	Female	null	null	Turner	Teacher	21

Table 2: CRTPersonal (Child Reference Table for Personal).

S_S_NO	Parent_Id	Personal	Status	address	Job	Sex	FCL
3	1	Pnull	well	Union st	Nurse	Male	Son
4	1	Pnull	Disable	Nevil Av	null	Female	Daughter

Table 3: Attribute Dictionary.

Attribute Name	Value	Parent_Id	Relation Type
S_S_NO	1	Null	PT
age	33	Null	PT
Sex	Male	1	CRT
Sex	Female	1	CRT
S_S_No	2	Null	PT
age	21	Null	PT

The FCL column in Table 2 shows, respectively, the SE Son and Daughter relationships with the parent. When the RDBMS is converted back to an XML document, this flag column plays an important role in allowing retrieval of ancestor information. As shown in Figure 2, the person having S_S_NO #1 has no Fname, so a null is inserted for that attribute of the corresponding tuple in Table 1. Also Figure 2 shows that NSE Personal has no direct value but has nested elements (address, S_S_NO, Status, Job). Thus pnull is inserted in the Personal column of Table 2, rather than creating a new reference table. When the RDBMS is converted back to an XML document, this NSE value of pnull acts as parent for all other attributes except Parent Id. Six tuples are created in the Attribute Dictionary corresponding to the six attributes in Figure 2. As shown in Table 3, six tuples and four columns are created as the attribute dictionary.

By way of comparison, consider Figure 3, an XML document for which the single relational table WorldPopulation Database is given in Table 4. Since there are no attributes or nested element paths in Figure 3, the WorldPopulation database is created in a simple way without having any child relations or attribute dictionary.

```
<World Population>
  <Info>
        <Name>  David </Name>
         <S_S_No> 1<S_S_No>
        <Zone> ASPC </Zone>
        <DOB>5/11/1969 </DOB>
        <Country> AUS </Country>
        <State> NSW </State>
        <City> Newcastle </City>
  </Info>
  <Info>    <Name>  Raul </Name>
        <S_S_No> 11<S_S_No>
        <Zone> AME </Zone>
        <DOB> 10/1/1970 </DOB>
        <Country> USA </Country>
        <State>Dalas</State>
        <City> Dalas </City>
  </Info>
  <Info>    <Name> Diana </Name>
        <S_S_No> 21<S_S_No>
        <Zone> EURO </Zone>
        <DOB> 7/2/1974 </DOB>
        <Country> UK </Country>
        <State> Oxford </State>
        <City> Oxford </City>
  </Info>
  <Info>    <Name>  Xu </Name>
        <S_S_No> 17<S_S_No>
            <Zone> ASIA </Zone>
        <DOB> 1/1/1960</DOB>
        <Country> China </Country>
  </Info>
</World Population>
```

Figure 3: Example of XML Document.

Table 4: Info.

Zone	S_S_N	Name	DOB	Count	State	City
ASPC	1	David	5/11/69	AUS	NSW	Newc
AME	11	Raul	10/1/70	USA	Dalas	Dalas
EURO	21	Diana	7/2/74	UK	Oxford	Oxford
ASIA	17	Xu	1/1/60	China	null	null

3.2 Search Time Analysis of System

Let k be the total number of distinct parent tables, n be the maximum number of tuples in a parent table, p be the maximum number of tuples in a child table, and m be the number of distinct child reference tables. A binary search technique may be applied to find a search key value in the parent and child relations. The searching time to find the data in any parent relation is $\Omega(k * \log_2^n)$. When it is

required to search both parent and child relations, the required time is $O(k*\log_2^n + m*\log_2^p)$. Attribute search time is $O(\log_2^j)$; where j is the total number of attributes in the attribute dictionary. It is required to sort data before attempting a binary search, and the required sort time is $O(n*\log_2^n + p*\log_2^p)$.

4 ANALYSIS OF STORAGE SPACE, RECONSTRUCTION OF XML AND QUERYING THE DATABASE

Analysis of storage capacity for different methods, the translation algorithm for conversion from RDBMS to XML form, and the estimated query time are presented in section 4.1, 4.2 and 4.3 respectively.

4.1 Analysis of Storage Capacity for Different Methods

Let K be the total number of distinct parent tables, n be the maximum number of tuples in a parent table, X be the number of attributes, and Y be the average required bytes for each attribute in the parent table. Since each tuple requires $X*Y$ bytes, for n tuples and K distinct tables the memory requirement would be $=K*n*X*Y$ bytes.

Let p be the maximum number of tuples in a child table, M be the number of distinct child reference tables, S be the number of attributes and T be the average required bytes for attributes in the child table. For child tables, then, the required memory would be $=M*p*S*T$ bytes. Similarly the required memory for the Attribute dictionary is $r*U*V$; where r is the number of tuples, U is the number of attributes, and V is the average number of bytes required for each attribute. Hence the total required memory of the scheme presented in this paper is given by:

$$S_{PCRR} = [K*n*X*Y + M*p*S*T + r*U*V] \quad (1)$$

According to the mapping scheme in (Florescu and Kossman 1999) the memory requirements are as follows: Total number of tuples

$$NT = \sum_{i=1}^{n} (no\_of\_child) \quad (2)$$

Where n=no_of_oid from graph; each tuple requires (at least):

$$S_{NT} = \sum([no\_attr\_in\_edge\_table]* [bytes\_required\_each\_attribute]) \quad (3)$$

The total required memory (using equations [2] & [3]) is at least, but probably (it only includes the edge table, and some memory is required for other tables) greater than:

$$S_{MAP} = NT * S_{NT} \quad (4)$$

According to the OEM method (Abiteboul 1997), if the graph is represented as a linked list where each node consist of three fields (such as Object Id, Contents of each Object, edge name), then the total required memory is:

$$S_{OEM} = [\sum(N) * (B_{Enode})] \quad (5)$$

N is the total number of nodes or children in the OEM graph, B_{Enode} is the required bytes to represent those nodes or children. According to the general technique in (Shanmugasadaram, Krishnamurthy et al. 2001) the required memory is S_{GEN}, calculated as the sum of the required memory for the DTD graph and for corresponding Relational table. Therefore:

$$S_{GEN} = [\sum(N_{DTD}) * (B_{Enode})] + [(N_{Att}) * (B_{Att}) * (T_{NR})] \quad (6)$$

Where N_{DTD} is the total number of nodes in DTD graph, B_{Enode} is the required bytes to represent each node, N_{Att} is the number of attributes in the relational table, B_{Att} is the required bytes for each attribute, T_{NR} is the total number of tuples in the relational table.

According to the memory requirement analysis in equations (1), (4), (5) & (6), the graphical representation of different methods is presented in Figure 4. The storage comparison is analysed by considering millions of element paths and attributes. We see from Figure 4 that the scheme presented in this paper is more space efficient than the other standard methods. A tabular form of the storage analysis is presented in Table 5.

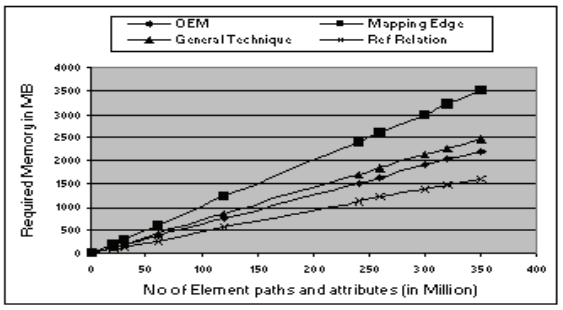

Figure 4: Storage Comparison analysis.

Table 5: Required Memory Using Different Methods.

Paths & Attribute(Million)	OEM Method (MB)	Mapping Scheme (MB)	General Technique (MB)	Our Proposed Scheme (MB)
1	6.31	10	7.05	4.63
20	126.2	200	141	92.6
30	189.3	300	211.5	138.9
60	378.6	600	423	277.8
120	757.2	1200	846	555.6
240	1514.4	2400	1692	1111.2
260	1640.6	2600	1833	1203.8
300	1893	3000	2115	1389
320	2019.2	3200	2256	1481.6
350	2208.5	3500	2467.5	1620.5

4.2 Reconstructing XML from RDBMS

To reconstruct an XML document from its RDBMS representation, the following issues are considered:

- All attributes from the parent relational table are sub-elements or attributes in the XML document.
- The content of each tuple from the parent table is enclosed within a sub-element bracketed by start and end tags.
- The existence of an entry in the attribute dictionary indicates that this an attribute rather than an element path in the XML document.
- Any attribute with a null value in a tuple can be excluded in a subelement for that tuple. For example, in Table 1 the first tuple has null for the Job attribute, indicating that there is no Job element path specified for the corresponding XML element.
- If more than one attribute has pnull in the corresponding tuple of a child reference table, the last attribute with value pnull (according to left to right ordering) is the parent element for all those attributes (except parent_Id) in the XML document. The contents of the Flag column from the child reference table indicates the ancestor element for all attributes and sub-elements of tuples. The corresponding algorithm is given in Figure 5.

```
Algorithm RDBMS_to_XML ( )
Begin
    Identify all distinct PT name as a
    SE in XML document;
    Store all the attributes (except
    the attributes in Dictionary) of
    the PT as the ChlSE and store the
    contents of tuple from PT to the
    content of ChlSE ;
For (each attribute name in Attribute
Dictionary as a PT type)
    Begin
       Set as a AttSE and store their
       values, in XML document
       according to Parent_Id;
    End;
For (Any reference CRT existing) to the
value of any particular attribute of PT
Begin
    Search the corresponding CRT
    relation to collect all the
    attributes from CRT (except the
    attributes in Dictionary) and set
    them as ChlNSE or NSE;
    Store the value of the attributes
    of each tuple in CRT as the content
    of ChlNSE or NSE;
  For (each attribute name in Attribute
  Dictionary as a CT type)
    Begin
       Set as a AttNSE or AttchlNSE and
       store their values, in XML
       document according to Parent_Id;
    End;
  For ((each pnull) value of the
  attribute in CRT)
    Begin
       Store as a parent among all
       NSE/attributes;
    End;
  For (each tuple of FCL)
    Begin
      Set as a grandparent among all
      the NSE/attributes
    End;
End; // For any Refer CRT
End. // For Main
```

Figure 5: Converting RDBMS to XML.

4.3 Querying the Database

In section 3.2, estimated times were calculated by using search time analysis for millions of tuples in the parent and (CRT) child relations.

In Figure 6 the estimated search time is shown for examples where the search data is found in the PT. Figure 7 shows the estimated searching time when it is necessary to search both the PT and child relation table. (Josifovskil, Fontoura et al. 2005) presents results of querying an XML stream; the evaluation time (XMark Query) is given in Figure 8 and includes the parsing times for both XSQ (one of the most complete XML stream processing systems, written in C++) and TurboXPath (also a path processor, written in Java). Although parsing in Java is slower that parsing in C++, TurboXPath is still much faster than XSQ.

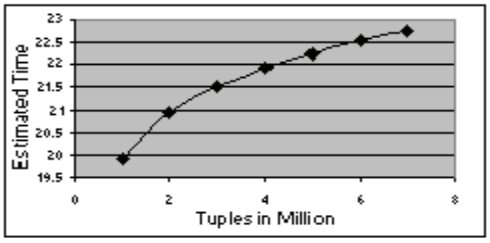

Figure 6: Estimated Searching time only in PT.

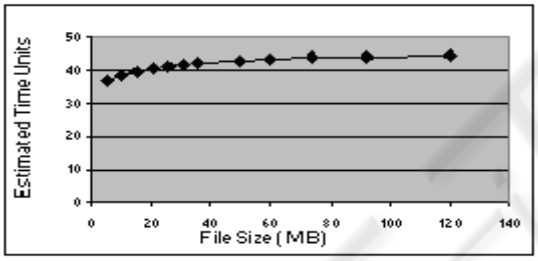

Figure 7: Estimated Querying time in PT & CRT.

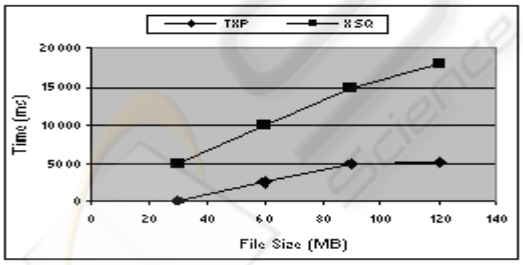

Figure 8: Evaluation time (XMark Query).

The results shown in Figure 6 and Figure 7 make it clear that the required time increases when the number of tuples increases and/or the file size increases. File size depends on both the number of tuples in the relational database and the number of paths in the source XML document. This is consistent with the experimental results obtained by (Josifovskil, Fontoura et al. 2005).

5 CONCLUSIONS

XML is a convenient, semistructured data format for information exchange and some data processing tasks. Other activities, particularly searching and sorting of data, are better supported if the data is represented in a more structured form, such as that used by relational databases. This paper presents a translation scheme that converts XML data into the relational table form that supports efficient pursuit of such other activities. The new technique is significant because it provides more efficient use of storage capacity than other similar schemes, while also supporting highly accurate transition from relational back to XML form when required. Algorithms are presented for conversion from XML to relational form, and for reconstruction of the original XML form from its relational representation. The latter conversion is significant because it uses an attribute dictionary to support accurate definition of element paths and attributes.

Analysis of query performance on the relational form of the XML data suggests that the representation scheme scales well, providing for efficient data processing of both small and large documents.

REFERENCES

Abiteboul, S., 1997. Querying Semistructured Data. ICDT, *The International Conference on Database Theory.* Delphi, Greece.

Abiteboul, S., I. Manolescu, et al., 2006. A Framework for Distributed XML Data Management. *EDBT,* Springer-Verlag Berlin Heidelberg.

Abiteboul, S., D. Quass, et al., 1996. The Lorel Query Language for Semistructured Data. *ftp://db.stanford.edu//pub/papers/lorel96.ps.*

Abiteboul, S. and P. Senellart., 2006. Querying and Updating Probabilistic Information in XML. *The Conference on Extending Database Technology*. Munich, Germany.

Balmin, A. and Y. Papakonstantinou., 2005. A Storing and Querying XML Data using Denormalized Relational Databases. *The journal on Very Large Databases (VLDB)* 14: 30-49.

Buneman, P., S. Davidson, et al., 1996. A query language and optimization techniques for unstructured data. *Inernational Conference on Management of Data (SIGMOD).* San Diego, USA.

Chen, Y., S. Davidson, et al., 2003. RRXS: Redundancy reducing XML storage in relations. *The 29th International Conference on Very Large Databases (VLDB*). Berlin, Germany.

Deutsch, A., M. Fernandez, et al. XML-QL: A Query Language for XML. *http://www.w3.org/TR/NOTE-xml-ql.*

Deutsch, A., M. Fernandez, et al., 1999. Storing Semistructured Data with STORED. *International Conference on Management of Data (SIGMOD)*. Pennsylvania, USA.

Du, F., S. Amer, et al., 2004. ShreX: Managing XML Documents in Relational Databases *The 30th International Conference on Very Large Databases (VLDB)*. Toronto, Canada,

Florescu, D. and D. Kossman., 1999. Storing and Querying XML Data using an RDMBS. *The IEEE Data Engineering Bulletin* 22(3)( ): 27-34.

Halverson, A., V. Josifovski, et al., 2004. ROX: Relational Over XML. *The 30th International Conference on Very Large Databases.* Toronto, Canada.

Harding, P. J., Q. Li, et al., 2003. XISS/R: XML Indexing and Storage System Using RDBMS. *The 29th International Conference on Very Large Databases (VLDB).* Berlin, Germany.

Harold, E. R., 2004. *The XML Bible*, Hungry Minds.

Josifovskil, V., M. Fontoura, et al., 2005. Querying XML Streams. *The journal on Very Large Databases (VLDB)* 14: 197-210.

Pal, S., I. Cseri, et al., 2005. XQuery Implementation in Relational Database System. *The 31st International Conference on Very Large Databases*. Trondheim, Norway.

Pal, S., I. Cseri, et al., 2004. Indexing XML data Stored in a Relational Database. *The 30th International Conference on Very Large Databases.* Toronto, Canada.

Senellart, P. and S. Abiteboul., 2007. On the Complexity of Managing Probabilistic XML Data. *The 27th Iinternational Conference on Principles of Database Systems.* Beijing, China.

Shanmugasadaram, J., J. Kiernan, et al., 2001. Querying XML Views of Relational Data. *The 27th International Conference on Very Large Databases (VLDB)*. Roma, Italy.

Shanmugasadaram, J., R. Krishnamurthy, et al., 2001. A General Technique for Querying XML Documents using a Relational Database System. *The journal (SIGMOD)* 30(3): 20-26.

Shanmugasadaram, J., K. Tufte, et al., 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities. *The 25th International Conference on Very Large Databases(VLDB)*. Edinburg, Scotland.