XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases

Zakaria Bousalem*, Ilias Cherti
Faculty of Sciences and Technology, University Hassan I, FSTS, Settat, Morocco.

* Corresponding author. Tel.: +212661539264; email: zakaria.bousalem@gmail.com
Manuscript submitted July 13, 2015; accepted September 24, 2015.
doi: 10.17706/jsw.10.12.1389-1401

Abstract: Due to Its simplicity, its flexibility and its expansion possibilities, XML can be adapted to multiple domains. Its self-described structure and nesting, allows XML to become the dominant standard for storing and transferring data through the World Wide Web. In addition the relational database systems are mature and extremely powerful. Therefore, many researches have been done to propose an efficient approach to store and query an XML document in Relational Database. In this paper we present a new approach to store and query an XML document using relational databases, which decompose an XML document into three tables without using any XML schema or DTD. Our approach supports efficiently the structural modifications to the XML tree, and achieves lower storage consumption. Also, we propose two powerful algorithms for mapping XML data to relational databases and from relational database to XML data.

Key words: XML, RDB, mapping, labeling scheme, algorithm.

1. Introduction

XML is a cross-platform, software and hardware independent tool for transmitting, storing and exchanging information [1]. XML is a widely recognized standard to ensure the portability of data using structural and semantic markup. The fact that contains both the data itself and information about their structure (self-describing structure) and its nesting provides a simple, yet, flexible means for applications to model and exchange data in a way that is readable by humans and easily verified, transformed, and published.

There are three approaches to storing XML data [2]. The first is a native XML repository that supports XML data models and query languages directly. The second, storing the data in relational tables via relational databases and finally, storing the XML data in object-relational databases [3], [4]. The first approach suffers [5] from several limitations (like access control information, returns the data only as a XML format…..). To overcome these limitations many approaches [2], [5]-[12] have been proposed, for taking advantages of the mature technologies that are provided by current relational DBMSs (like indexes, triggers, data integrity, security and query optimization by SQL), and also in order to exploit the full power of this new technology. These approaches can be separated into two main categories:

Schema-based mapping: This approach uses the structure of a schema for that document (DTD or XML Schema) to generate the relational schema for storing an XML document [8], [9], [11], [13], [14]. The structure of the model of the relational database will be generated using the foreign key and the primary key that points to the parent-child relationship in the XML Tree (extracted from DTD or XML Schema).

Generic mapping: The fundamental concept of this approach is to use a regular schema for storing the

XML documents in relational database without the existence of DTD or XML Schema. The principle is to catch the tree structure of the XML document [6], [7], [10], [12].

The first approach gives advantages over the second. Firstly the mapping can be independent of the schema of XML document (DTD or XML Schema). Secondly it has a unified method of query mapping. On the other hand the Schema-Based XML mapping has efficient performance of query processing.

According to (Atay et al., 2007) [15], the process of storing the XML documents in a relational database is composed of four essential steps:

1) **Schema mapping:** this first step consists for generating the relational schema corresponding to XML structure from XML schema (DTD or XML Schema) or without it.
2) **Data mapping:** the goal of this step is to save the XML data in the relational database generated in the first step, as relational tuples.
3) **Query mapping:** the aim is to convert the XML queries (XPATH or XQuery) into SQL queries and to be executed after in the relational database.
4) **Reverse data mapping:** The result of the query mapping will be turn relational data. This data will be translated into XML data in the structure of the input XML query.

The main idea of this paper is to present a new approach to carry out the Schema mapping and Data mapping steps. Our approach will benefit from the advantages of the Path-Based Approach [7], [16] and also benefit from the advantages of the OrdPath approach. Our approach supports efficiently the structural modifications (Insertion, update and delete) to the XML tree, and achieves lower storage consumption.

The remaining of the paper is organized as follows: in Section 2, we briefly introduce the related works. Section 3 describes the principle of the Ordpath labeling scheme. In Section 4, we present our XML Data model. In Section 5, we describe the relational schema design used in our approach and we compare our approach with other methods. Section 6 proposes an implementation of our approach. Finally, Section 7 concludes our contribution.

2. Related Works

Many studies focus on the area of the storing and retrieving XML document in relational database: **Edge** [6], **XRel** [7] and **XParent** [16]. The **XParent** approach consists of storing an XML document in a four tables:

LabelPath(ID, Len, Path)
DataPath(Pid, Cid)
Element(PathID, Did, Ordinal)
Data(PathID, Did, Ordinal, Value)

LabelPath table stores label-path in the path attribute. Each path identified by ID, the Len attribute records the number of edges for each label-path. The DataPath table stores the parent-child relationship information between nodes in XML DATA Tree. The Pid is the ID of the parent and Cid is the ID of the child. The Data table stores information about these nodes in XML DATA Tree. The Element table is like the Data table, but it does not record the values of the nodes.

The Edge approach records all information of nodes in XML DATA Tree in one table:

Edge(Source, Ordinal, Target, Label, Flag, Value)

An edge is identified by two nodes identifier: Source and Target. Ordinal attribute stores the ordinal of an edge. Label attribute records the node name. The Flag attribute stores a sign to indicate if this edge is point to object (ref) or to a value (val). If the value of the Flag attribute is: val, the Value attribute stores the value of this object else, this attribute will be empty.

The XRel approach is a four table database schema as follows:

Path(PathID, Pathexp)
Element(DocId, PathID, Start, End, Ordinal)
Text(DocId, PathID, Start, End, Value)
Attribute(DocId, PathID, Start, End, Value)

This approach introduces to a novel concept of region to store the parent-child relationship by using two attributes: Start and End. The Path table records the distinct paths, Table Element stores information about all element nodes in the XML DATA Tree, Text table records information about the leaf elements. Attribute table stores the attribute nodes in the XML DATA Tree. The use of region concept reduces the performance of the queries execution (effect of Theta Join).

In our approach we don't use path in string format to reduce the storage consumption. Also we don't use type of the node attribute; it is deduced from other attributes. We use Ordpath labeling schemes to support efficiently the structural modifications.

3. ORDPATH Labeling Scheme

Several labeling systems of the nodes have been proposed. Besides its effectiveness in insertion and compression, ORDPATH [17] is conceptually similar to the Dewey technique. And as long as the most labeling schemes are designed for static XML data structures, insertion in an XML document will be a great challenge.

ORDPATH provides the possibility of effective insertion at any position in an XML document, and also supports execution of native XML queries with high performance.

The changes in the structure of an XML tree can be done in several ways:

1) New sub-trees can be inserted.
2) Sub-trees can be removed.
3) Sub-trees can be moved in the tree.

On the contrary, for existing labeling systems, simple insertion of a node can be very expensive because it requires a large number of labeling of existing nodes for each new inserted node.

ORDPATH materializes the parent-child relationship in the ID attribute. The concept is to set the child node by the label of the parent ORDPATH plus the local ID of the Child (e.g. 1.3 is the parent node ID of 1.3.5).

In initial load only odd, positive integers are assigned. The other integers (even-numbers and negatives) are reserved for later insertion.

For inserting a new node X between two existing nodes, we use even numbers for example, to insert a new node between two nodes: 1.3.5 and 1.3.7, the new id is 1.3.6 .

4. An XML Data Model

In this paper we adopt the XPath data model [18] to represent our XML document. In the XPath data model, to represent an XML document, the nodes are organized as an ordered tree. There are 7 types of nodes. For simplicity we content ourselves with four types: root, element, attribute and text. Figure 1 is an example XML document describing "**breakfast menu**" information.

Fig. 2 illustrates the XML Data Tree corresponding to our XML document in Figure 1 based on Ordpath [17] labeling scheme. In this figure we represent the root element by black round and the regular elements by yellow round. All the yellow triangle nodes are attribute nodes. Finally we represent the string value by

white rectangle.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<menu>
  <food>
    <name>Belgian Waffles</name>
    <price currency="$">5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories fat="7.7" total="310"/>
    <vitamins>
      <a>7</a>
      <c>0</c>
    </vitamins>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price currency="$">7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories fat="18" total="700"/>
  </food>
</menu>
```

Fig. 1. An example XML document.

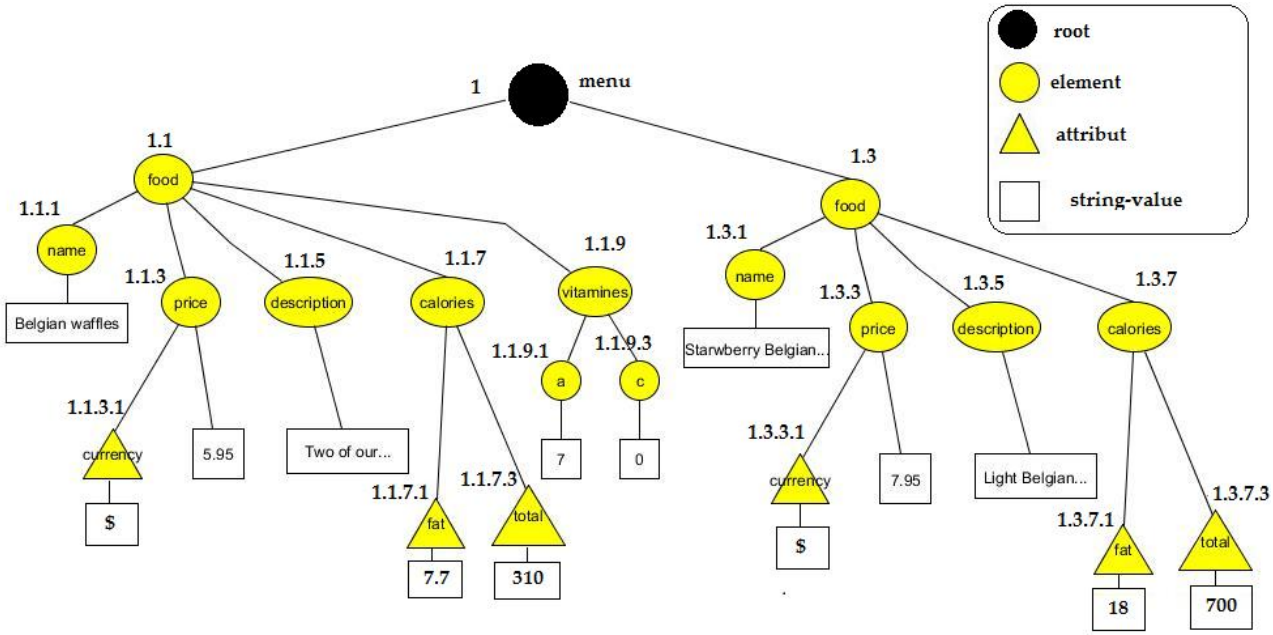

Fig. 2. XML data tree of our XML document in Fig. 1.

5. XMap: Our Proposed Approach

5.1. Presentation of the Approach

In this section, we describe our proposed relational database schema used to store XML documents. The structure of this approach is as follows:

Data Table: this table stores the related value of the node if one exists, otherwise it is NULL. It has the following structure:

Data (ORDPATH, value, order, number Elements, number Attributes, pathId) where**:**

ORDPATH: Stores ID of the node, **reflects** a numbered tree edge at successive levels down the path from the root. This attribute stores both the ID of the node and its parent. For example **1.1.9.1** is the ID of one

node, and **1.1.9** is the ID of his parent.

value: maintains the value of the **node**.

order: preserves the order of the **node** among all nodes in document.

numberElements: Stores the **number** of the elements childs of a node. It is used in step of reconstructing of an XML document.

numberAttributes: Stores the number of the attributes of a node. It is used in step of reconstructing of an XML **document**.

pathId: Maintains the path from the root to each node. This column is used in searching for path expressions in the query. This attribute is a foreign key of the ID in table **Path**

Vertex Table: this table is used for storing all nodes name in the XML document with a unique ID. It has the following structure:

Vertex (id, name) where**:**

id: Stores ID of the node **name**.

name: Maintains the name **of** the node.

Path Table: this table is used **for** storing all paths existing from the root to each node in the XML document with a unique ID. It has the following structure**:**

Path (id, path) where:

id: Stores ID of the **path**.

name: maintains the path that is **constructed** from the ids of all node used in this path from the root to each node. In this attribute we use the numerical ID instead of using the string to reduce the storage space.

ordpath	value	order	numberElements	numberAttributes	pathId
1		1	2	0	1
1.1		2	5	0	2
1.1.1	Belgian Waffles	3	0	0	3
1.1.3	5.95	4	0	1	4
1.1.3.1	$	5	0	0	5
1.1.5	Two of our famous ...	6	0	0	6
1.1.7		7	0	2	7
1.1.7.1	7.7	8	0	0	8
1.1.7.3	310	9	0	0	9
1.1.9		10	2	0	10
1.1.9.1	7	11	0	0	11
1.1.9.3	0	12	0	0	12
1.3		13	4	0	2
1.3.1	Strawberry Belgian	14	0	0	3
1.3.3	7.95	15	0	1	4
1.3.3.1	$	16	0	0	5
1.3.5	Light Belgian waffle..	17	0	0	6
1.3.7		18	0	2	7
1.3.7.1	18	19	0	0	8
1.3.7.3	700	20	0	0	9

Data Table

id	name
1	menu
2	food
3	name
4	price
5	currency
6	description
7	calories
8	fat
9	total
10	vitamins
11	a
12	c

Vertex Table

id	path
1	1
2	1.2
3	1.2.3
4	1.2.4
5	1.2.4.5
6	1.2.6
7	1.2.7
8	1.2.7.8
9	1.2.7.9
10	1.2.10
11	1.2.10.11
12	1.2.10.12

Path Table

Fig. 3. Database tables.

5.2. Comparison with Other Appraoches

Table 1. Comparison with Other Approaches

Approach	Comparison	
Edge	This approach allows to store all information in one table. That will cause to create a difficulty of execution of the query for evaluation of the path through the creation of several joints to test the "label path". To avoid this problem, XMap uses a "path" table that allows storing all possible path, which can accelerate the assessment of path expressions (this idea is available in the XRel approach).	By using the OrdPath labeling schemes. Our approach supports efficiently the structural modifications (Insertion, update and delete) to the XML tree, and achieves lower storage consumption, which is not supported by all these approaches.
XRel	The main advantage of our approach in comparison with XRel is XRel uses the region concept. It maintains a containment relationship to save the ancestor-descendant relationships. This principle uses the theta-join, which is slowing the execution of queries. However in our approach we use the parent-child relationships by storing the id of the parent in the first part of Ordpath for every node. Then we use the equijoins for evaluation the parent-child relationships.	
XParent	In our approach we use the idea of the table "PATH" already used in the XRel method to accelerate the assessment of the path expressions. On the other hand the XParent approach uses two tables "LabelPath" and "DataPath". This causes a very costly execution of queries by increasing number of joints and also by using the SELF JOIN.	

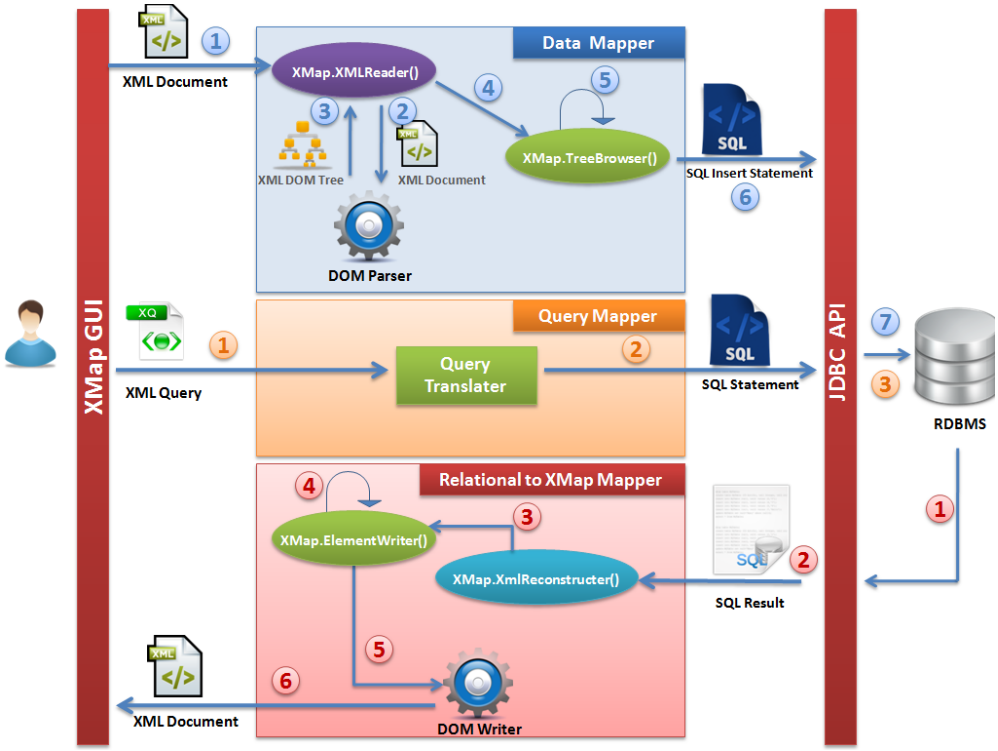

Fig. 4. The proposed model for mapping process of XMap.

6. Implementation

In this section we represent the implementation of our system. XMap is written in Java and we use MySql 5.5.16 to store our relational data. In the following we present the proposed model for Mapping Process of

XMap, the XML to Relational mapping Algorithm and the Relational to XML Relational mapping Algorithm.

6.1. Proposed Model for Mapping Process of XMap

In order to obtain the aim of our system we propose a model for Mapping Process of XMap as shown in Fig. 4. Our model consists of three components.

Component one maps XML document into Relational database using the "**DOM Parser**" that gives as result a "**XML DOM Tree**". The function "**XMap.TreeBrowser**" iterates the "**XML DOM Tree**" and executes the SQL insert statement to store the XML data in RDB , The rule of Component two is translate the XML Query to SQL statement and launch it in the database engine to obtain the results. Component three reconstructs XML document from RDB by transforming the SQL Result to hierarchical data format and return back to the user as a XML document.

6.2. XML to Relational Mapping Algorithm

Fig. 5 presents the algorithm for mapping an XML document to relational database. The algorithm works by recursively traversing the XML DOM Tree.

We use the "**DOM Parser**" to extract the "**XML DOM Tree"** from the XML document (line 5), and we assign the value "1" to the information associated to the root (id, order and idPath) (lines 8-10), then we add the name of the root element and the path "1" to the "**STACKs**" to store it in the database at the end of the algorithm. We use the **treeBrowser** function (line 17) that allows traversing each item passed in parameter. The first thing to do is take the list of sub-elements (line 18) and traverse it(line 20), If there are any nodes of "ELEMENT" type, we increment the **nbrElm** variable (number of sub-elements) and the order variable (lines 21-23) , then we assign the "ID" of this element using the "**getId"** function that generates "ID" from the id of the parent and from the ID of "**PreviousSibling**", and also we affect the "**PathID**" of this element by the index of his "**path**" in the "**stackPath**" (lines 24 -26), effectively we add after the name of the element node in "**stackVertex"** if it does not already exist , and the same for the "**path**" in the "**stackPath**" (27-28 lines), then we continue in the same way for the attributes of each element (lines 29-37).

In the line 38 and 44, we store the information of the attributes and elements respectively in the database, and after traversing all nodes of "**XML DOM tree**", we store in the database all existing "**path**" (line 14) and "**vertex**" (line 15).

The Fig. 3 shows an example for result of the XML to Relational mapping Algorithm of the XML document in Fig. 1.

6.3. Relational to XML Mapping Algorithm

Fig. 6 presents the Relational to XML mapping Algorithm. Firstly we create a root element of the document "**root**" from the first tuple of **resultSet** (line 7), and we append that element to our XML document (line 8).

We traverse the resultSet to associate the "root" by these attributes using the attribute "numberAttributes" of the first tuple of resultSet (9-11 lines) while the order of the document is preserved

We add other elements to the document, but we skip the attributes indices of root (line 12). For that, we use the recursive function **elementWriter()** that has two parameters:

1) The first is the parent element that we want to associate with these sub-elements
2) The second is the number of these sub-elements (we find this number in the numberElements attribute of the resultSet (line 13))

The function elementWriter() traverse resultSet to link the sub-elements to these parents.

```
01 Algorithm XML_to_RDB Mapping
02 Input : XML Document
03 Output : XML data stored in relational database.
04 function XMLParser() {
05        Document doc = domParser.loadDocument(XMLDocument)
06        Element root =doc.getRootElement()
07        globale order ;
08        root.setId("1")
09        root.setIdPath("1")
10        root.setOrder("1")
11        stackVertex.add(root.getNodeName())
12        stackPath.add("1")
13        treeBrowser(root)
14        insertPATH()//to insert infomation of all distinct path
15        insertVERTEX()to insert infomation of all distinct vertex
16 }
17 function treeBrowser(Element elm) {
18        ChildNodes = elm.getChildNodes()
19        nbrElm = 0
20        foreach node in ChildNodes do {
21        if ( node type == ELEMENT_NODE) {
22        nbrElm++
23        order++
24        node.setId(getId(PreviousSibling, ParentId))
25        node.setOrder(order)
          // id is deduced form the parent id and the PreviousSibling id
26        node.setIdPath(stackPath.indexOf(path) + 1))
27        if (!stackVertex.contains(nodeName)) then stackVertex.add(nodeName)
28        if (!stackPath.contains(path))       then stackPath.add(path)
   //path is deduced from  path of parent and nodeNameId
29        if (e.hasAttributes()) {
30          int numAttrs = attributes.getLength()
31          for (int j = 0 j < numAttrs j++) {
32          order++;
33          path =stackPath.get(ParentIdPath-1)+"."+(stackVertex.indexOf(attributeName) +1)
34          attribute.setId(ParentId + "." + ((j * 2) + 1))
            //in initial load we use only positive odd integers
35          attribute.setIdPath((stackPath.indexOf(path) + 1))
          //path is deduced from  path of parent and attributeNameId
36        if (!stackVertex.contains(attributeName)) then stackVertex.add(attributeName)
37        if (!stackPath.contains(path))         then stackPath.add(path)
38        Store attributeId, attributeValue, order, 0, 0, attributeIdPath INTO DATA table
39        }
40        }
41        readXML(e)
42        }
43        }
44        Store elementId, elementValue, elementOrder, nbrElm, elementAttributesLength, elementIdPath  INTO DATA
          table
45 }
```

Fig. 5. XML to relational mapping algorithm.

On line 18 we create an element and associate it with these attributes (lines 20-23) and then set this element with his text value (lines 24-26), and finally attach it with his parent element (line 28). We continue in the same way for the other elements using the same function recursively (29-30 lines).
To fill the resultSet used in our algorithm we execute the query in Fig. 9.

6.3.1. Complexity

The complexity of this algorithm is equal to **O (n)**, it's a linear complexity, where **n** is number of elements and attributes in the XML document. Indeed in our algorithm a search is performed by traversing once the **resultSet** using a loop statement (ligne17). The first loop statement "**while"** (ligne17) will be executed "**ne"** times according to the number of element nodes in XML document. The second loop "**for**" (line 20) will be

executed "**na**" times depending on the number of attribute nodes in the XML document. Knowing that **na + ne = Total number of elements and attributes**. Consequently we deduce that the complexity equals **O (n).**

```
01 Algorithm RDB_to_XML Mapping
02 Input : SQL ResultSet
03 Output : XML Document
04 function XMLReconstracter() {
05        global i=0
06        global document
07        Element root = document.createElement(resulSet[0][name])
08        document.appendChild(root)
09        for (int j = 1; j <= resulSet[0][numberAttributes]; j++) {
10                root.setAttribute(resulSet[j][name], resulSet[j][value])
11        }
12        i = i + resulSet[0][numberAttributes]
13        elementWriter(root,resulSet[0][numberElements])
14 }
15 function elementWriter(Element parent, int numberElements) {
16        int j = 0
17        while (j < numberElements) {
18                int localIndice=i
19                element = document.createElement(resulSet[localIndice][name])
20         for (int k = localIndice + 1; k <= localIndice + resulSet[localIndice][4]; k++) {
21                element.setAttribute(resulSet[k][name], resulSet[k][value])
22                i++
23         }
24        if (resulSet[localIndice][numberElements] == 0) {
25                element.createTextNode(resulSet[localIndice][value])
26        }
27        i++
28        parent.appendChild(element)
29        if (resulSet[localIndice][numberElements] > 0) {
30                elementWriter(element,resulSet[localIndice][numberElements])
31        }
32        j++
33        }
34 }
```

Fig. 6. Relational to XML mapping Algorithm.

6.4. Translating XPath Queries into SQL Queries

Our relational database schema is modeled to benefit both from the advantages of **paths** to accelerate research and also to minimize the storage of these **paths**. For that we choose to store the **path** as shown in Fig. 3, and also we implement MySQL functions to convert the **path** used in the XPath queries into format of **paths** used in our Path table.

- **Function** getOrdPath**()**

This function (Fig. 7) takes in parameter the **path** in text mode and then convert it to the format used in our approach. This function uses another function **getVertexId().**

- **Example of use**

```
SELECT getOrdPath("/menu/food/price/currency");
```

- **Result :**

```
1.2.4.5
```

```
DELIMITER $$
CREATE function getOrdPath( parmPath TEXT )
RETURNS VARCHAR(160)
READS SQL DATA
BEGIN
  DECLARE varId VARCHAR(10);
  DECLARE varIds VARCHAR(160);
  DECLARE varValue VARCHAR(60);
  DECLARE instrValue INT;
  SET varIds = '';
  SET varId  = '';

  REPEAT
    SET instrValue = INSTR(parmPath,'/');
    IF instrValue <> 0 THEN
      SET varValue = SUBSTRING(parmPath, 1, instrValue-1);
      SET parmPath = SUBSTRING(parmPath, instrValue+1);
      SET varId = concat( getVertexId(varValue), '.');
    ELSE
      SET varValue = parmPath;
      SET varId = getVertexId(varValue);
    END IF;
    SET varIds = CONCAT(varIds, varId);
  UNTIL instrValue = 0
  END REPEAT;
  RETURN varIds;
END
```

Fig. 7. Get ordpath function.

```
DELIMITER $$
CREATE function getVertexId( varValue TEXT )
RETURNS VARCHAR(10) READS SQL DATA
BEGIN
  DECLARE done  INT DEFAULT FALSE;
  DECLARE varId VARCHAR(10);
  DECLARE getId CURSOR FOR
    SELECT id
    FROM vertex
    WHERE name = varValue;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done  = TRUE;
  SET varId = '';
  OPEN getId;
  REPEAT
    FETCH getId INTO varId;
  UNTIL done = TRUE
  END REPEAT;
  CLOSE getId;
  RETURN varId;
END
```

Fig. 8. Get vertexid function.

- **Function getVertexId () :**

The role of this function (Fig. 8) is to select from the database the **ID** of each element of the **path**.

- **Function getParent () :**

We used this function to extract the parent **ID** from an ordpath **ID**.

  - **Example of use**

```
SELECT getParent("1.5.1");
              1.5
```

6.4.1. XPath to SQL

In this section we treat translation of the XPath queries into SQL queries. For this we will treat three types of queries.

Table 2. XPath Queries

Query number	XPath query
Q1	/menu
Q2	/menu/food[price>4]/name
Q3	/menu/food[calories/@total =700]/price

Q1 : /menu

This query selects all nodes in the XML document. This query is used to fill the **resultSet** of the Relational to XML mapping algorithm. The corresponding SQL statement to this XPath query is:

```
SELECT ordpath,name,value, numberElements, numberAttributes
FROM data,path,vertex
WHERE  data.pathId=path.id
AND vertex.id= cast(SUBSTRING_INDEX(  path.path,  '.', -1 ) AS UNSIGNED)
ORDER BY order ASC
```

Fig. 9. SQL query to select all nodes in the XML document.

Q2 : /menu/food[price>4]/name

This query selects the names of "**food**" in which the price is greater than 4. In order that the result will be correct, we must test if the price is greater than 4 and that the element "**price**" and the element "**name**" are for the same parent. The corresponding SQL statement to this XPath query is:

```
SET @ pathName =getOrdPath("/menu/food/name");
SET @pathPrice =getOrdPath("/menu/food/price");
SELECT d2.Value
FROM Data d1, Data d2, Path p1, Path p2
WHERE  p1.path = @pathPrice
AND p2.path =@pathName
AND cast(d1.Value AS DECIMAL(10,2)) > 4
AND d1.pathId = p1.id
AND d2.pathId = p2.id
AND getParent(d1.ordpath) = getParent(d2.ordpath)
```

Fig. 10. SQL query to select the names of "food" in which the price is greater than 4.

Q3 : /menu/food[calories/@total=700]/price

This query selects the prices of "food" in which the total of calories equal 700. In order that the result will be correct, we must test if "total" equal to 700, and parent of the "price" element and grandparent of the "total" element are the same. The corresponding SQL statement to this XPath query is:

```
SET @pathPrice =getOrdPath("/menu/food/price");
SET @pathTotal =getOrdPath("/menu/food/calories/total");

SELECT d1.Value
FROM Data d1, Data d2, Path p1, Path p2
WHERE  p1.path = @pathPrice
AND p2.path =@pathTotal
AND cast(d2.Value AS DECIMAL(10,2)) = 700
AND d1.pathId = p1.id
AND d2.pathId = p2.id
AND getParent(d1.ordpath) = getParent(getParent(d2.ordpath))
```

Fig. 11. SQL query to select the prices of "food" in which the total of calories equal 700.

7. Conclusion and Perspectives

In this paper we proposed a novel approach to store and query XML document over relational database. In this approach we used three tables to store all information about an XML documents. The benefit of this approach is that supports efficiently the structural modifications using the Ordpath labeling schemes and achieves lower storage consumption. In perspective, we envisage to compare and test our approach with other existing approaches to assess its performance. Another promising direction for future work is to provide a powerful Framework for XQuery to SQL translation with all queries types (select, insert, update, and delete).

References

[1] Introduction to XML. Retrieved, from http://www.w3schools.com/xml/xml_whatis.asp

[2] Tian, F., DeWitt, D. J., Chen, J., & Zhang, C. The design and performance evaluation of alternative XML storage strategies. University of Wisconsin, Madison.

[3] Kudrass, T., & Conrad, M. (2002). Management of XML documents in object-relational databases. *Proceedings of the EDBT 2002 Workshops XML-Based Data Management and Multimedia Engineering* (pp. 210-227).

[4] Naser, T., Alhajj, R., & Ridley, M. J., (2008). Reengineering XML into object-oriented database. *Proceedings of the IEEE International Conference on Information Reuse and Integration.*

[5] El-Aziz, A. A., & Kannan, A. (2012). Relational storage for XML rules. *International Journal on Web Service Computing*, *3(4)*, 33.

[6] Florescu, D. E. K. D. (1999). Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin*, *22*, 3.

[7] Yoshikawa, M., Amagasa, T., Shimura, T., & Uemura, S. (2001). XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, *1(1)*, 110-141.

[8] Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., & Zhang, C. (2002). Storing and querying ordered XML using a relational database system. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (pp. 204-215).

[9] Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D. J., & Naughton, J. F. (1999). Relational databases for querying XML documents: Limitations and opportunities. *Proceedings of the 25th International Conference on Very Large Data Bases* (pp. 302-314). Morgan Kaufmann Publishers Inc..

[10] Suri, M. P., & Sharma, D. (2012). A model mapping approach for storing XML documents in relational databases. *International Journal of Computer Science Issues*, *9(3)*.

[11] Suri, P., & Sharma, D. (2013). Schema based storage of XML documents in relational databases. *International Journal on Web Service Computing*, *4(2)*, 23.

[12] Fakharaldien, M. A. I., Edris, K., Zain, J. M., & Sulaiman, N. (2012). Mapping extensible markup language document with relational database management system. *International Journal of Physical Sciences*, *7(25)*, 4012-4025.

[13] Du, F., Amer-Yahia, S., & Freire, J. (2004). ShreX: Managing XML documents in relational databases. *Proceedings of the Thirtieth International Conference on VERY large Data Bases* (pp. 1297-1300).

[14] Ramanath, M., Freire, J., Haritsa, J. R., & Roy, P. (2003). Searching for efficient XML-to-relational mappings. *Database and Xml Technologies*.

[15] Atay, M., Chebotko, A., Liu, D., Lu, S., & Fotouhi, F. (2007). Efficient schema-based XML-to-relational data mapping. *Information Systems*, *32(3)*, 458-476.

[16] Jiang, H., Lu, H., Wang, W., & Yu, J. X. (2002). Path materialization revisited: an efficient storage model

for XML data. *Australian Computer Science Communications*, *24(2)*, 85-94.

[17] O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G., & Westbury, N. (2004). ORDPATHs: Insert-friendly XML node labels. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 903-908).

[18] Clark, J. and DeRose, S. (1999). XML path language (XPath). Retrieved 16 November, from http://www.w3.org/TR/xpath.

Zakaria Bousalem is a PhD student in the Department of Mathematics and Computer Science, FSTS, University Hassan 1st, Settat, Morocco. His research interest are in XML processing, relational database management system (RDBMS), and query processing and optimization.

Ilias Cherti is a full professor in the Department of Mathematics and Computer Sciences, University Hassan 1st, Faculty of Sciences & Technology, Settat, Morocco. His research interests are intelligent systems, partial and differential equations. He has organized various national and international events.