Design & Implementation Of XML-RDBMS Interface

Vibhore Kumar
(B.Tech III yr.)
Department of Computer Science & Engineering,
Institute of Technology, BHU, Varanasi, India
Ph: +91-522-369867
e-mail: vibhore_k@hotmail.com

Abstract

XML has now established itself as an extremely versatile language capable of labeling the information content of diverse data sources such as relational databases, object repositories, structured and semi structured documents. It has developed into a standard for exchanging data on the World Wide Web. However, relational databases will continue to act as storage for most of our data. Consequently, if XML is to fulfill its potential, some mechanism is needed to publish relational data as XML documents. It also becomes equally important to devise a method that can serve to condense an XML document into relational database.

The aim here is to design and implement algorithms that can serve as XML-RDBMS interface. The standard approach of dealing with XML data relies on semi structured query engines in which XML documents are treated as data sources. What we propose here is a development of an interface that accepts XML documents and stores them into relational database. Besides providing this facility, the interface is also be capable of generating results of a query in the form of XML document that conforms to a provided DTD. To this end, we have developed algorithms and implemented them as a prototype that can make relational tables out of a given DTD, accept XML documents conforming to the DTD, and store them into relational tables. The system is also capable of generating XML documents conforming to a given DTD when a query is fired onto the relational database. The system has been evaluated using a wide variety of test data and it was found to work for the well-formed XML documents and valid DTDs. The interface returns suitable error messages for illegal inputs.

Index Terms – Schema, DTD, Parser, SAX, DOM

ADVANCES IN DATA MANAGEMENT 2000
Krithi Ramamritham, T.M. Vijayaraman (Editors)
Tata McGraw-Hill Publishing Company Ltd.

1.Introduction

Extensible Markup Language (XML) has now well established itself for the purpose of data transfer over the web. The reason for this wide acceptance of XML documents is the fact that XML data is self-describing. This means the program receiving an XML document can interpret it in multiple ways, can filter it and can structure it to suit its needs.

XML has been primarily used to enhance the capability of remote applications to operate and interpret upon data fetched over the web. As the application of XML becomes dominant for the purpose of data transfer, it has become almost necessary to provide an interface between XML and existing established databases. However this raises many exciting possibilities, various approaches have been adopted for design of such interfaces but an ideal solution is still awaited.

One of the most widely accepted approach is to treat XML documents as semi structured data sources. The approach uses semi structured query engines to fire a query over a set XML documents and get the result, at first glance this seems to be quite a rational approach. But the fact is that most of our data is being stored in relational databases and a lot of research has been put into developing this model which has well established itself. Another suggested approach is to use XML documents for the purpose of data transfer and use an interface which establishes a link between XML and existing database. This approach has the advantage of not undoing the labor that has been put into development and design of our current databases besides this we can also use query engines, which already exist.

An RDBMS based XML system is possible because of the fact that XML documents can be made to conform to a particular DTD. A DTD is in effect a schema for a set of XML documents and this is what serves to link XML with RDBMS. The approach we are going to use here is the following. Given an XML document and a DTD, the DTD can be processed to generate a relational schema. Now this schema can be used to store any XML conforming to the DTD that has been processed. The other part consists of constructing an XML document from a query fired over relational database that conforms to DTD supplied by the source firing the query or it generates a DTD itself for the XML document if one is not specified.

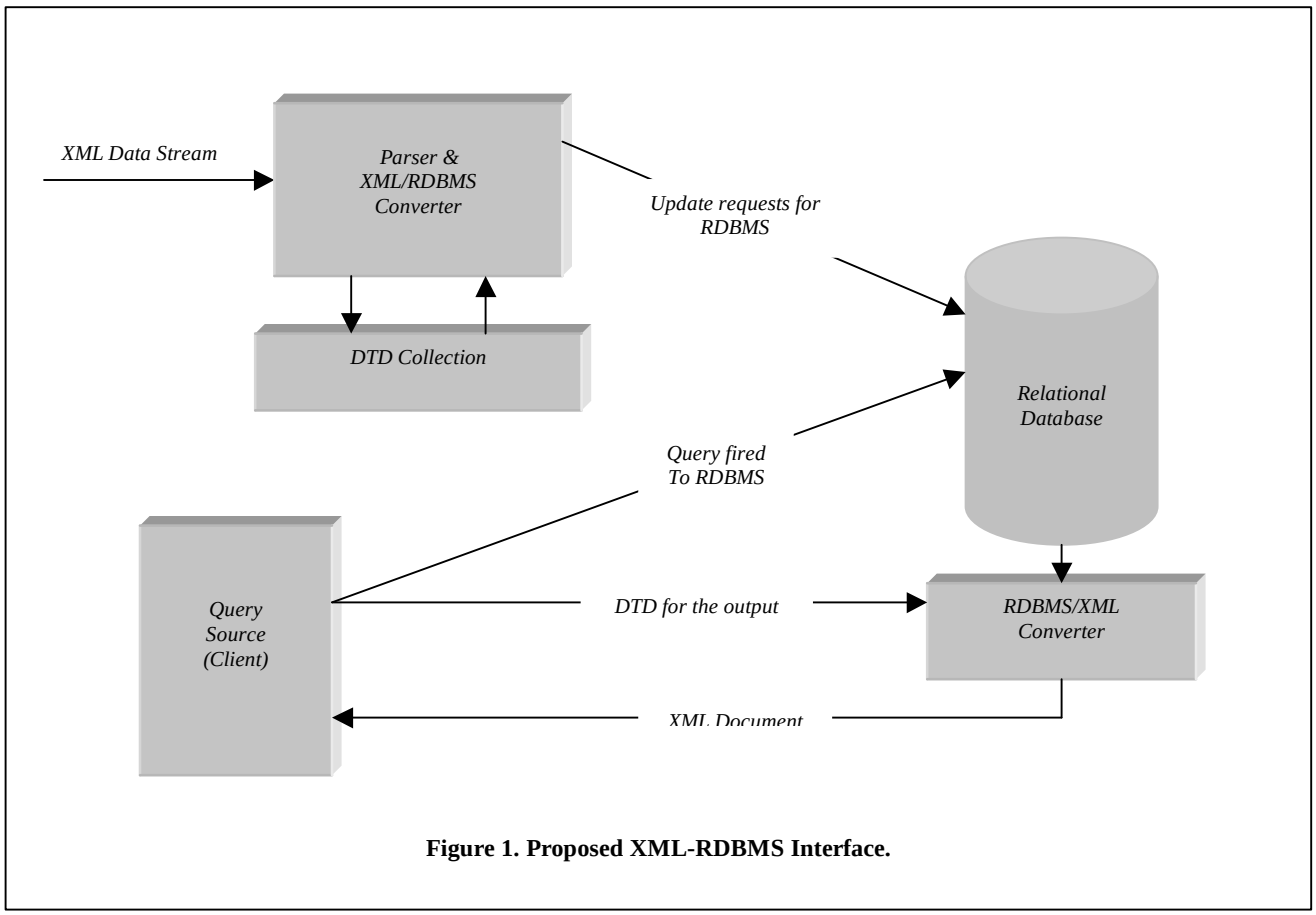

Figure 1. Proposed XML-RDBMS Interface.

The approach we have adopted works but it does not necessarily imply that it is an ideal solution. This approach has limitations that it increases the overhead of conversion to and from XML. This overhead increases as the size of document increases. A lot needs to be done to reduce this overhead of conversions.

The advantage of using RDBMS based XML system is motivated by considering the scenario of a 'Placement Service' on web that accepts resumes from various candidates applying for jobs. The employers are provided with the resumes from various candidates satisfying their eligibility criteria. In an ad hoc situation, the applicants can apply in varied formats. This makes it difficult to categorize and store the relevant information in a manner such that it can be easily retrieved, depending on desired criteria.

However, if all candidates are provided with a fixed DTD and they are required to submit their resumes as XML documents conforming to it, the task simplifies. The conformance of all resume to fixed DTD makes it easy to store the XML document into a pre-existing relational DBMS or any database system using XML-RDBMS interface. Now whenever some employer requests for candidates having certain qualifications, the database can be searched for and the result published as an XML document in a format which suits the employer's existing database. Here the company can specify the format of XML document by supplying the 'Placement Service' with a DTD that matches their database.

1.1 Related Work

A lot of work has been going on in the field of XML, much of the research in this area has been focussed on the development of special purpose query engines for semi-structured data. However, our aim here is to exploit the similarities between XML and RDBMS based systems. Our work is quite similar in its approach to the work [1] but here we are concentrating not on development of interface that converts semi-structured queries to relational queries. Our approach here is to fire a relational query on a database along with a DTD and to get the result as an XML document, which conforms, to the given DTD. However if no DTD is provided the result is an XML document along with a DTD for the same.

Most of the proposed techniques for DTD to Relational Schema conversion are such that they tend to lose any similarity to the original DTD. The algorithm that we have proposed tries to overcome this drawback.

1.2 Roadmap

The rest of the paper has been organized as follows. An overview of XML documents, schemas and DTD is given in Section 2. A

discussion of the proposed DTD to Schema conversion algorithm is contained in Section 3 along with the implementation issues. Section 4 describes the process of storing of XML documents to relational tables. Section 5 deals with the firing of queries and retrieval of XML documents from relational results. Section 6 finally concludes by discussing the brighter aspects and the drawbacks of the proposed interface, besides giving a list of areas where there is a scope of further improvement.

2.An XML Primer

We give a very brief overview of XML documents, schemas and DTDs in this section. More information can be obtained from the references.

2.1 XML Documents

XML (eXtensible Markup Language) may be defined as a simplified subset of SGML that is meant to be used on the Web. XML incorporates the features of both SGML and HTML. It includes those pieces of SGML that were heavily used but leaves out all the optional features that held SGML back. Because of this, it retains the power and flexibility of SGML without the complexity. XML is more powerful than HTML because it is extensible. Users can define new tags and attributes and are not limited to the finite set that never seems to satisfy anyone.
An XML document consists of nested element structures, starting with a root element. Table 1 shows an XML document that represents a typical Resume of a candidate. In this example there is a 'Resume' element which contains a number of sub-elements. Further information can be found in References [3,4].

```
<?xml version="1.0"?>
<!DOCTYPE Resume SYSTEM "resume.dtd" [
]>
<Resume Height='6 feets' Marital_Status ='no'>
    <Name>
        Ankur
    </Name>
    <Age>
        20
    </Age>
    <Sex>
        M
    </Sex>
    <Address>
        <Addr_Current>
            <Street>
                Rajpur Road
            </Street>
            <City>
                Dehradun
            </City>
            <PIN>
                226024
            </PIN>
        </Addr_Current>
        <Addr_Permanent>
            <Street>
                Rajpur
            </Street>
            <City>
                Lucknow
            </City>
            <PIN>
                226024
            </PIN>
        </Addr_Permanent>
    </Address>
    <Qualifications>
        <Qualification>
            <Degree>
                X
            </Degree>
            <Year>
                1996
            </Year>
            <Division>
                I
            </Division>
        </Qualification>
        <Qualification>
            <Degree>
                XII
            </Degree>
            <Year>
                1998
            </Year>
            <Division>
                I
            </Division>
        </Qualification>
    </Qualifications>
    <Experience>
        <Exp>
            none
        </Exp>
        <Exp>
          1yr
        </Exp>
    </Experience>
    <Projects_Undertaken>
        <Proj>
            abc
        </Proj>
    </Projects_Undertaken>
    <Hobbies>
        <Hobby>
            Chess
        </Hobby>
    </Hobbies>
</Resume>
```

XML document representing Resume "Resume.xml"
Table 1.

2.2 DTDs & Schema

A Document Type Definition describes the legal elements and attributes that can be used to markup a document. This is essentially a contract between the application and the user of the markup language - if the user marks up a document in a certain way, then the application can be relied upon to respond accordingly. The additional advantage of a DTD is that they are defined on a rigorous syntax, which means that it becomes possible to 'validate' (i.e. check) a document against its DTD to see whether it conforms to the letter of the contract.

XML schemas can be considered as extensions of DTDs. Schemas allow typing of values and they also allow us to set size specifications. XML schemas are yet to become standard and if they become, it will allow us to create tables with integer attributes rather than just using strings.

3 Generating Relational Schema out of DTD and constructing Tables

The DTD describes a model of the structure of the content of an XML document. This model says what elements must be present which ones are optional, what their attributes are, and how they can be structured in relation to each other. In this section, we discuss the generation of relational Schemas from XML DTDs. This section has been divided into three parts, each of which corresponds to the three major issues that are going to be dealt with (a) Simplification of DTDs and extraction of relevant information (b) Carving out tables from the information gathered in preceding section and (c) Resolving conflicts, dealing with attributes and special cases.

3.1 DTD Simplification

A DTD, whether external or internal , is generally the most complex part of an XML family. To our advantage methods exist that can simplify complex DTDs to simpler ones and still the resulting Schema developed from simplified DTD is capable of storing any XML document conforming to the given DTD. The guiding factors while generating a relational Schema are that the tables created out of it must be able to store any conforming XML document and that any XML semi-structured query must be transformable to an equivalent relational query so as to produce similar results.

In a DTD the sub-elements that constitute XML elements are specified using the various operators provided for the purpose *(set with zero or more elements), +(set with one or more elements), ?(set optional), and |(or). These specifications add to the complexity of the DTD. Algorithms exist for simplification of DTDs and these are good enough so as not to reduce the effectiveness of queries over documents conforming to DTD. One of the algorithms as was proposed in [1] is described by the transformations given in table 2.

These transformations can be applied to any DTD in a step by step manner for the purpose of simplification. Let us for example consider an element defined as X((a?,(b*,c**,d*?)?)*,(a|b)*,(c|d*)?). Now, by using the above transformations this element can be evaluated and expressed as a simplified element. The given element simplifies to X(a*,b*,c*,d*).Before proceeding with these transformations '+' operators are transformed to '*'. It may be worth noticing that simplification may at times change the relative ordering of the elements but this can be retrieved from XML documents conforming to DTD.

Table 2. Transformation Table

(a)	
*(X1,X2)**	*X1*,X2**
(X1,X2)?	*X1?,X2?*
(X1|X2)	*X1?,X2?*
(b)	
*a***	*a**
a?*	*a**
*a**	*a**
a??	*a?*
(c)	
…a,…,a*,…*	*a*,…*
…a,…,a?,…*	*a*,…*
…a?,…,a,…*	*a*,…*
…a?,…,a?,…	*a*,…*
…a,…,a,…	*a*,…*

Once the DTD has been simplified we can proceed with the pre-processing which is required for generating the Schema. In this pre-processing, we extract the information from DTD. All the elements which are directly used to store information (i.e. elements defined as (#PCDATA) or ANY) are stored in a 'variable list'. The next step requires us to identify those elements followed by *,?,+ and these are stored in a 'exception list'

3.2 Construction of Relational Tables

The construction of relational tables from a given DTD is motivated by the fact that in a relational model one table can be related to the other by having an unique identifier which links the two table. The proposed algorithm has the disadvantage of constructing an appreciable number of tables but the fact that it preserves the basic structure of the DTD negates this disadvantage. The algorithm has been designed keeping in mind that it must produce tables that are capable of storing any XML document conforming to the DTD. Further it must be able to accept any query (when converted to required format) that is meaningful when fired over the relevant XML document.

A set of general rules may be defined to facilitate the conversion of DTD to Schema. The term 'variable' in our terminology stands for any element that has no sub-elements (i.e. one defined as (#PCDATA) or ANY), rest all elements will be referred to as 'non-variables'. The various combination of elements in a parenthesis can be mapped to corresponding create table statements using the Table 3.

Table 3. Conversion Table

T(var1,var2,var3,…,varN) Þ Table(ID,var1,var2,var3,…,varN)
T(vara1,…..,varaM*,var1,…,varN)Þ Table(ID,r_ID,var1,…,varN) + vara1_Table(ID,vara1) +… … +varaM_Table(ID,varaM)* *(here Table.r_ID=vara1_Table.ID=…..=varaM_Table.ID)*
T(non-var1,….,non-varM,var1,…,varN) Þ Table(ID,r_ID,var1,…,varN)+Tables for non-var's *(here Table.r_ID=ID column value in table for non-var's)*
T(non-var1,…,non-varM*,var1,…,varN) Þ Table(ID,r_ID,var2,…,varN)+Tables for non-var's* *(here Table.r_ID=ID column value in table for non-var's)*
T(non-var) Þ Table for non-var*
T(…….)+attribute list(at1,at2,…,atN)Þ T(ID,at1,at2,…,atN,…….)
T(var) Þ Table(ID,var)*

- *It may be noted here that these conversions are to be applied after simplification of the DTD and that here * can be replaced by ?*
- *These conversions hold good irrespective of the position of the variables and non-variables in the parenthesis.*

Table 4. A typical DTD representing Resume of a candidate. "Resume.dtd"

```
<!ELEMENT Resume ( Name, Age, Sex,  Address, Qualifications, Experience, Projects, Hobbies )>
<!ATTLIST Resume   Height          CDATA
                   Marital_Status CDATA
>
<!ELEMENT Address ( Addr_Current, Addr_Permanent )>
<!ELEMENT Addr_Current ( Street, City, PIN)>
<!ELEMENT Addr_Permanent ( Street, City, PIN)>
<!ELEMENT Qualifications ( Qualification*)>
<!ELEMENT Qualification ( Degree, Year, Division)>
<!ELEMENT Experience ( Exp*)>
<!ELEMENT Projects ( Proj*)>
<!ELEMENT Hobbies ( Hobby*)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Age (#PCDATA)>
<!ELEMENT Sex (#PCDATA)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT PIN (#PCDATA)>
<!ELEMENT Degree (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Division (#PCDATA)>
<!ELEMENT Exp (#PCDATA)>
<!ELEMENT Proj (#PCDATA)>
<!ELEMENT Hobby (#PCDATA)>
```

Using the transformation given in Table 2 we can convert a DTD to a set of statements which when executed will result in tables for the specified DTD. Let us for example consider the Document Type Definition (DTD) given in Table 4. The DTD maps to the set of tables shown in
figure 2 .

3.3 Resolving Conflicts & dealing with special cases

While implementing the above algorithms, the following factors need to be kept in mind. The members of the 'exception list' always have a table corresponding to them. The members of 'variable list' appear as column names in a table

4. Storing XML documents in Relational Tables

In this section we will deal with the issue of storing XML documents into relational tables. This section has been divided into two parts in the first part we discuss the algorithm for storing XML documents into relational tables. In the next part of the section the implementation of the same has been discussed.

4.1 Extracting data out of an XML document conforming to relational tables

On a close examination of any XML document, the following observations are worth noticing

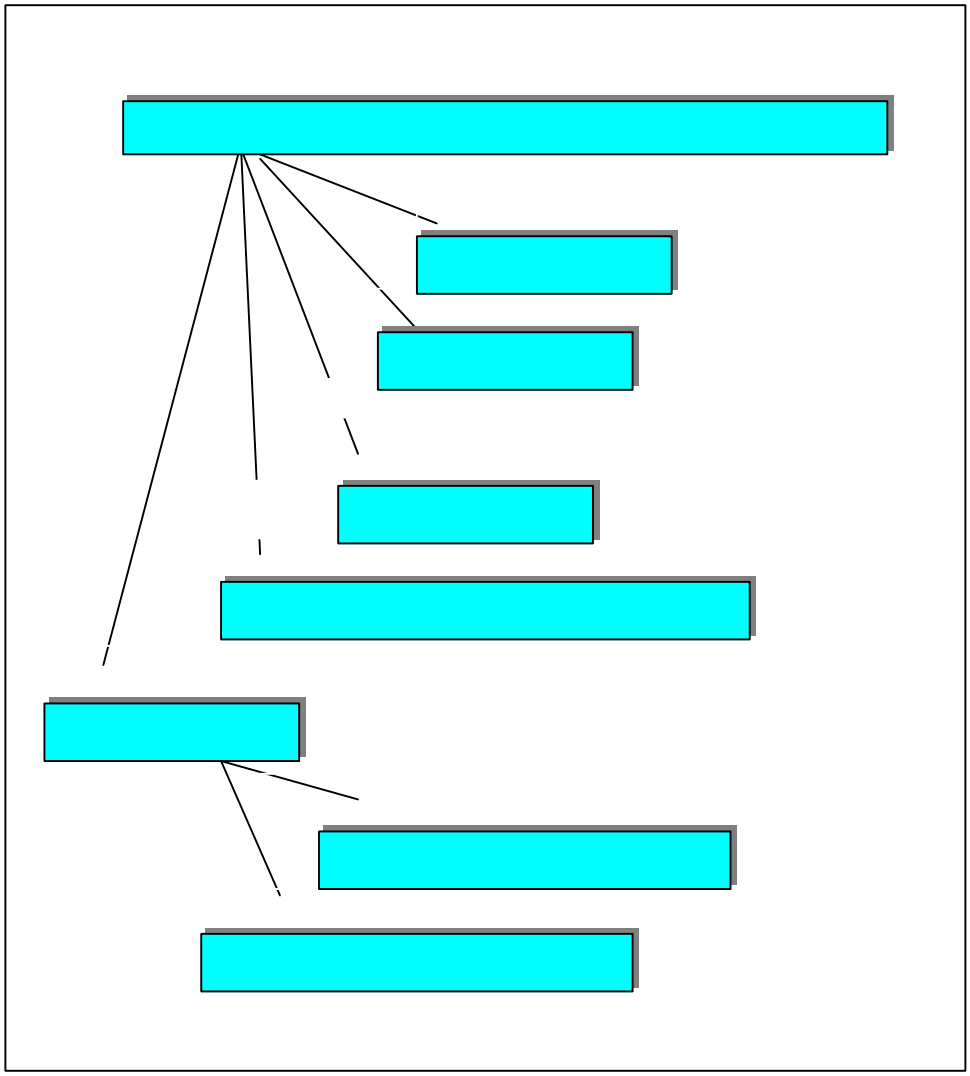

Figure 2. A Schematic representation of Tables formed for Resume DTD

a. If an XML document is read in sequential manner from start to end we encounter as many opening tags as the closing tags.
b. At any given position inside an XML document more than one tag may not have been closed.
c. Only one tag is active tag at any instant and that the active tag is the one that has been most recently opened.

The above observations make it very clear that there exists a stack like structure somewhere in an XML document. The PUSH operation can be defined as opening of a tag and closing of a tag may be referred to as a POP operation. The tag on the top of the stack may be referred to as the active tag.

Also define a variable 'tag_level' which represents the number of tags that are open when the tag is pushed.

Let us do this study in relation to a simpler XML document and the corresponding DTD.

```
<?xml version="1.0"?>
<!DOCTYPE musicians SYSTEM "music.dtd" [
    ]>
<musicians>
    <musician>
        <name>
                ankur
        </name>
            <instrument>
                drum
            </instrument>
    </musician>
</musicians>
```

"music.xml"

```
<!ELEMENT musicians ( musician*)>
<!ELEMENT musician ( name,instrument)>
<!ELEMENT name (#PCDATA)>
    <!ELEMENT instrument (#PCDATA)>
```

"music.dtd"

On applying the algorithm, discussed in the previous section, the DTD is transformed to the following set of relational tables using the conversions.

T(non-var) Þ Table for non-var*
T(var,var) Þ Table(ID,var,var)

So, we get here as a result the table

musician

ID	*name*	*instrument*

The following set of rules may be defined for the extraction of data from an XML document.

1. Whenever a start tag is encountered push the tag onto the stack as a string TAG + (+ID +, attributes…, .
2. Any data element encountered is added to string of currently active tag as STRING + element +, .
3. Any attribute elements for a tag are added after ID elements.
4. If a non-element tag is encountered then an element r_ID is added after ID if it does not exists to currently active tag and then the new tag is pushed onto the tag with ID value = r_ID value.
5. Whenever an end tag is encountered the currently active tag string is terminated by replacing the last ',' by ')' and the string is written to file with the following exceptions.

(Exception :- If the string contains only two elements then the element other than ID is written to the stack string of the tag which becomes active on poping of the current tag. This is done only when the tag is not a member of exception list defined earlier.)

Using these rules and applying them on the "music.xml" file we get the following strings :

musicians(ID(auto-generated),r_ID(auto-generated))

musician(ID,ankur,drum)

It may be noticed here that here a string for musicians table is also created but since the table does not exist it would not be executed.

4.2 Populating the relational tables

The strings obtained by following the procedure given in previous discussion can be used to populate the tables. The string for inserting data into table should be used only when a table corresponding to them exists. However for complex DTDs which have undergone simplification an additional check for relative ordering of elements has to be introduced.

5. Firing Queries and Retrieval of XML documents from Relational Tables.

Till now, we have discussed the part of interface which deals with the storage of XML documents into Relational Tables. Once an XML document has been stored into a relational table some method is needed to retrieve data from such tables in the form of XML documents. In this section we deal with the part of interface which corresponds to the retrieval of data in form of XML document when a query is fired on a relational database. Here we consider two cases that arise due to the fact that at times the query maker may want the results to conform to a particular DTD.

Case 1: A DTD is not given
On firing a query on a set of Relational Tables the resultset we get is itself a table which contains the columns that were requested for and records matching the conditions set in the query, are written to this result set.

When a DTD is not given the results can be framed into an XML document having more or less a linear DTD. Complications arise when two columns in different tables have similar names to avoid these complications the name of duplicate columns is always post-fixed with an auto-generated number and is then used as a tag. The results are published as a simple XML document having a general format.

```
<Result>
    <record>
        <tag1>
            data
        </tag1>
        .
        .
        .
        <tagN>
            data
        </tagN>
    </record>
    .
    .
    .
</Result>
```

where N is the number of columns that were requested in the query.

Case 2: A DTD is given for Results

Before proceeding, we must specify the assumptions that are being made in the design of algorithm for this section.

Assmp. 1: The query that is fired along with the DTD must contain all the columns (tags) that are being referred in the DTD.

Assmp. 2: The tag names must match with those in the relational tables.

Assmp. 3: To avoid conflicts due to co-existence of columns with similar names when a column has to be referred it is prefixed with the name of the table it belongs to.

Assmp. 4. The name of the root element is always Result and it is defined as (record*)

The procedure to publish results in conformance to given DTD includes the following steps:

Step 1: Fire the query on the Relational database and obtain the resultset
Step 2 : From the resultset obtained obtain the names of the columns that are contained in each record and store them in an array in the order in which they appear in the resultset.

Step 3 : Get the order of elements in which they appear in the DTD for a record and rearrange the resultset in the same order using a redirecting array.

Step 4 : Sort out the variables and the non-variables as given in section 3.

Step 5 : Starting from root element which is a collection of records read the specification for record onto a stack and start reading sub-elements sequentially.

Step 6 : If a variable is encountered then fill in the data from resultset, and attach a start and end tag.

Step 7 : If a non-variable is encountered attach a start tag and push the sequence of sub-elements corresponding to non-variable onto a stack.

Step 8 : fetch sub-element from top of the stack if they are available. If no more sub-elements are available perform a pop operation and attach a corresponding end tag. If the stack is empty jump to step 9 else fetch sub-element from the top and Repeat from step 6

Step 9 : Increment the resultset cursor. If more records are available then goto step 5 else stop.

To understand in detail the working of the above algorithm let us consider an example. The database that user has to access has the following two tables (fig.3)

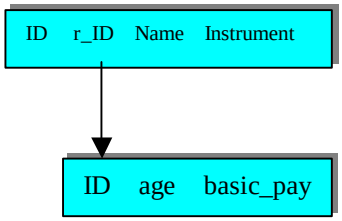

Figure 3. Table Schema

musician

ID	r_ID	Name	Instrument
1	musician1	Ankur	Drum

information

ID	Age	basic_pay
musician1	20	0

Now, the client fires the query

*SELECT **
FROM musician, information
WHERE ID = 1 and information.r_ID = musician.ID

with a DTD of form:
<!ELEMENT Result (record)>*
<!ELEMENT record (musician.Name, Info)>
<!ELEMENT Info (musician.Instrument, information.age)>

Processing the query step by step in accordance to the algorithm, we arrive at the following XML document

```
<Result>
    <record>
        <musician.Name>
                Ankur
        </musician.Name>
        <Info>
                <musician.instrument>
                        drum
                </musician.instrument>
                <information.age>
                        20
                </information.age>
        </Info>
    </record>
</Result>
```

This algorithm works well for most of the DTDs but fails when an attribute is encountered. This is however due to the fact that specifying value of attribute is not possible in the DTD itself. However, with some assumptions we are trying to implement algorithm that can process attributes as well. The algorithm we have proposed may require some processing to be done on the client side as well to make the XML fully conform to client database in terms of column names.

6. Concluding Remarks

In this paper we have proposed the design of an interface which can serve for the conversion of XML document to relational tables further the same interface can serve for the conversion of results obtained from query on Relational Tables to XML document. The proposed interface has been found to work well for valid XMLs and DTDs. The overhead of conversion to and from XML makes it this approach costly in terms of system utilization for large XML documents and large queries.
Possibilities of future work include studying the impact of parallelism on the application of proposed algorithms. Furthermore the interface we have proposed needs to be optimized to reduce the overheads.

Acknowledgements

This work has been done under the guidance of Prof. N.L.Sarda at the Department of Computer Science & Engineering, Indian Institute of Technology, Bombay, as part of summer project. The author is grateful to the department for extending the facilities to carry out the work.

References

[1] J. Shanmugasasundaram, Gang H., Tufte K., Zhang C., Dewitt D.J. & Naughton J.. *Relational databases for querying XML documents: limtations and opportunities, Proceedings of the 25th International Conference on VLDB.* (1999).

[2] McHugh J. & Wisdom J.. *Query Optimization for XML, Proceedings of the 25th International Conference on VLDB.* (1999).

[3] J. Bosak, T. Bray, D. Connolly, E. Malor, G. Nicol, C.M. Sperberg-McQueen, *W3C XML Specification DTD,* http://www.w3.org/XML/1998/06/xmlspec-report.htm.

[4] R. Cover, *The SGML/XML Web Page,* http://www.oasis-open.org/cover/xml.html.