6-12-2006

An Automated XPATH to SQL Transformation Methodology for XML Data

Sandeep Jandhyala

AN AUTOMATED XPATH TO SQL TRANSFORMATION METHODOLOGY FOR XML DATA

by

SANDEEP JANDHYALA

Under the Direction of Rajshekhar Sunderraman

ABSTRACT

In this thesis we present an automated system that allows users to execute XPATH queries against an XML data source. The system exploits the shared-inlining mapping from XML to Relational data. At the core of the system is an XPATH to SQL transformation algorithm that produces corresponding SQL queries for a subset of XPATH. This approach allows one to utilize standard relational databases to store XML data. Given a DTD, the system creates appropriate relational tables based on the shared-inlining method. The system is capable of transforming an XML data source that conforms to the DTD into relational data. The main component of the system is the XPATH interpreter that parses an XPATH expression for the XML data source and transforms it into an equivalent SQL query. The SQL query is then executed against the relational database and results are packaged into XML and returned as the answer to the XPATH query. The use of the relational database to store and query the XML data is transparent to the user as they interact only with the XPATH interpreter. This methodology provides a novel technique to provide an XML database system implementation.

Index Words: XML SQL transformation, XPATH to SQL queries, XSU, Data mapping.

AN AUTOMATED XPATH TO SQL TRANSFORMATION METHODOLOGY FOR XML DATA

by

SANDEEP JANDHYALA

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

In the College of Arts and Sciences

Georgia State University

2006

AN AUTOMATED XPATH TO SQL TRANSFORMATION METHODOLOGY FOR XML DATA

by

SANDEEP JANDHYALA

Major Professor : Dr.Raj Sunderraman
Committee: Alex Zelikovsky
Sushil K.Prasad

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
May 2006

Dedicated to everyone who was part of this
For all the support

Acknowledgements

I would like to thank my advisor, Dr. Raj Sunderraman, for the guidance and support provided by him during this study. He was open to new ideas and flexible with the ways of implementing the thesis, keeping the goals in mind. I would also like to thank Mr. Victor Fu, because of whom I was able to work on similar work in job field. Dr. Sushil K. Prasad & Dr. Alex Zelikovsky were kind enough to review the manuscript and provide me with fine pointers to meet the standards.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Stands for
XML	Extensible Markup Language
HTML	Hyper Text Markup Language
SGML	Standard Generalized Markup language
RDBMS	Relational Database Management System
SQL	Structured Query Language
PLSQL	Procedural Structured Query Language
JDBC	JAVA database Connectivity
DOM	Document Object Model
DTD	Document Type Definition
XSU	XML SQL Utility
API	Application Program Interface
XSLT	Extensible Style sheet Language Transformations
JVM	Java Virtual Machine

1. INTRODUCTION

For any organization to run successfully, the data of the firm needs to stored and maintained effectively. There are always issues of performance as per time efficiency, cost incurred, safety of the data, and convenience as per maintaining distributed databases. Mismanagement and inconsistency of data leads to the loss of huge amounts of money. Hence, lots of efforts and money are being invested in this area; even so more are the revenues being generated. Databases are the main source for storing large volumes of data. XML provides a platform independent way of storing data. Hence the two of them form the main hosts of data storage and retrieval.

In many organizations there is a requirement to map the data that is in one form to another form for reasons depending on the specifications of the projects. For example, the users may prefer to store all the data which is in XML files in the relational database or the relational data in the XML format. There may also a need to perform the operations which can be done on XML in relational database and vice versa such as run queries against the relational database and generate results in XML. This XML document may need to conform to a DTD given in the specifications. Two organizations using different format of data storage, may need to exchange data. XML being platform independent is one main reason why people prefer data transfer in the XML file format.

The thesis is to meant to serve these business needs, and at the same time to explore the feasibility and complexity of mapping data in relational format to XML and vice versa. The shared inlining method is used to map the DTD of an XML file to the corresponding relational schema. The system then loads the data from the XML file confirming to the given DTD into the database. At the stage the user can run all his SQL queries and generate results in XML. At the core of the thesis is an automated system for converting XPATH queries into SQL queries. The storage of data in the relational format and the data retrieved in relational format by posing SQL queries to the relational database is kept transparent to the user. The results are retrieved back in the XML format. This methodology provides a novel technique to provide an XML database system implementation.

2. BACKGROUND INFORMATION

A thesis is an intellectual proposition. The following chapter contains information on the technologies and other facts that have been used to successfully complete the research and achieve my findings. This can be used as a guide and reference for the implementation methodology.

2.1 Extensible Markup Language (XML)

Both HTML and XML are subsets of SGML. While HTML tags deal with the display styles and format of a data item, XML was designed to describe the content of the data. XML is used to store information and data is stored in the plain text format. Hence it has become a means for data communication among the systems in the web. It provides a software- and hardware-independent way of sharing data. The clients and applications can access the data stored in XML like they are accessing data from databases. XML has now become the leading standard for data communication in the World Wide Web and between databases. Few Database vendors have started releasing products which provide support for data exchange between their database and XML.

2.2 Relational Database Management System (RDBMS)

The success factor for any company is the way they store and retrieve data. This data is stored in tables in the databases, which represent a relation among certain entities called columns. In addition to the content of the data, the constraint the data has to satisfy is very crucial to the database. Three such important constraints are primary key, foreign

key and not null. ORACLE is leading database vendor and it supports SQL to create, query the database and perform other transactions. It also provides support for PL/SQL so as to run procedures on the data.

2.3 JAVA and DATA Structures

JAVA is an Object Oriented Programming Language. JAVA programs are converted into byte code, which is run by the JVM. Because of this reason JAVA supports portability on several platforms and JVM addresses security issues. It is the predominant programming language of the web. Vector is a data structure which is included the Utility package of JAVA. It implements a dynamic array. Hash table is yet another data structure which is similar in nature to a dictionary. It is integrated into the Collections framework in JAVA.

Hash table stores key/value pairs. One can specify the Key object in order to get back its corresponding value or detect the presence of a value in the hash table.

2.4 JAVA Database Connectivity (JDBC)

ORACLE supports JDBC API for applications using JAVA to connect to the database. With the use of JDBC, applications can connect to many databases simultaneously such as SYBASE, DB2 by loading their appropriate drivers. The steps involved in connecting to a database from an application involve

- Loading the database driver
- Creating a connection object, this is like establishing a connection with the database by giving it the username and password of the database.

- Creating a statement object, this is needed to specify a query to be run on the database.
- Executing the statement and process the results as per the application.

2.5 XML parsing in JAVA

JAVA is an object oriented programming language and is platform independent. Since XML documents are used by many applications, support for extracting data from those documents is necessary. ORACLE provides an XML parser package in JAVA that provides implementations for the extracting data from XML documents. There are two such parsers namely SAX and DOM. SAX parser doesn't store the entire XML document in memory as it uses an event call back mechanism. Hence it is suitable for applications which require just one pass of the XML document to extract the data. DOM parser creates a node tree of the XML document and loads it into memory. Hence it is mostly used for those applications which require multiple passes of the XML document. It is more efficient than SAX. The DOM tree consists of nodes which correspond to the elements, attributes of the XML document. It has several functions which such as get siblings, get parent, get children, get node value which provide a lot of useful information about the XML document to the applications.

2.6 Document Type Definition (DTD)

The structure of an XML document is defined in a DTD. They are similar to schema and are easier to specify. The DTD specifies the data types of the elements, order of elements, whether the elements have attributes, whether specifying an attribute for an

element is mandatory or optional, the child elements of an element, the number of occurrences of elements and a lot more. The DTD validates the XML document and raises an error if any elements or attributes are violating the given format. This way the applications which involve data transfer using XML can do the error handling.

2.7 XPATH

XPATH is a language used to query a XML document for data. The query is an expression. Given an XPATH expression, XPATH navigates through the XML document to fetch the results matching the predicates and conditions mentioned in the expression. The following are the components of XPATH expressions.

Expression	**Description**
Node name	Selects all child nodes of the node
/	Select from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects the attributes

Table 1. Operators in XPATH

Predicates are embedded in square brackets. In addition to the above operators * is also used to match any element node. If @* is mentioned, XPATH matches it with any attribute nodes of the XML document. XPATH uses a library of standard functions such as numeric functions, string functions, and functions on nodes, sequences, contexts, which aid the users to specify simpler queries. XPATH cannot address join queries, but it is very crucial. XPATH is used in many tools like XSLT, XQUERY, X pointer, X link and DOM parsing.

2.8 XML SQL Utility (XSU)

Mapping data between XML and relational databases has become necessary for many organizations, because these days XML is used as a data store in many applications. Oracle provides support for this through XSU. Some of the points here are referred from [8].

Through XSU one can

- Transform data retrieved from object-relational database tables or views into XML when a query is given.
- Extract data from an XML document, and using a canonical mapping, insert the data into appropriate columns or attributes of a table or a view given a schema for the tables or views.
- Extract data from an XML document and apply this data to updating or deleting values of the appropriate columns or attributes.

- Generated DTD's dynamically
- Generate XML Schema given an SQL query.
- SQL identifier to XML identifier escaping. Sometimes column names are not valid XML tag names. To avoid this you can either alias all the column names or turn on tag escaping
- Generate XML documents in their string or DOM representations.
- Insert XML into database tables or views.

The following figures are taken from [8].

XSU can be installed at the database end in which case the overall picture looks like

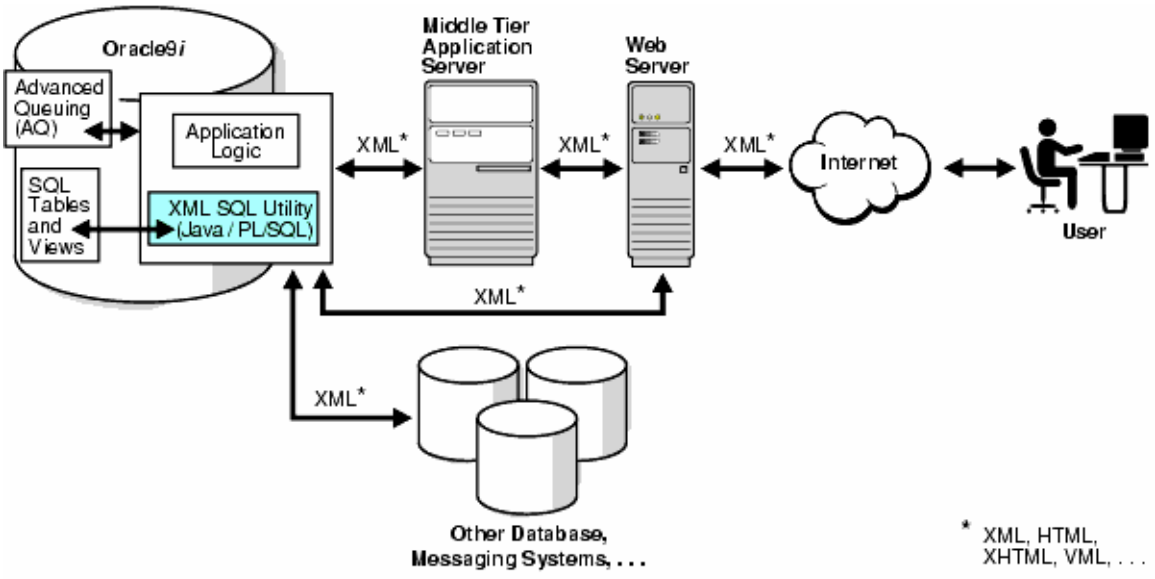

Figure 1 XSU tool in the database server

If the architecture needs an application server in the middle tier, and there is a need for XML support there, then XSU can be installed in the middle tier, the overall picture looks like

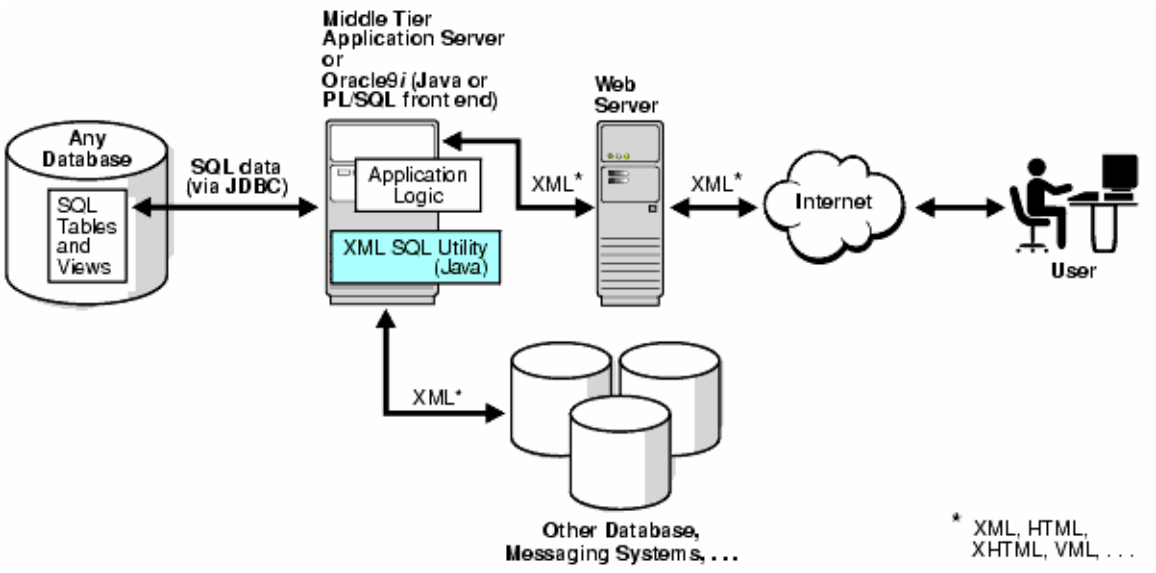

Figure 2. XSU in the Application tier

XSU can also be installed in the web server, for this the web server needs to support servlets.

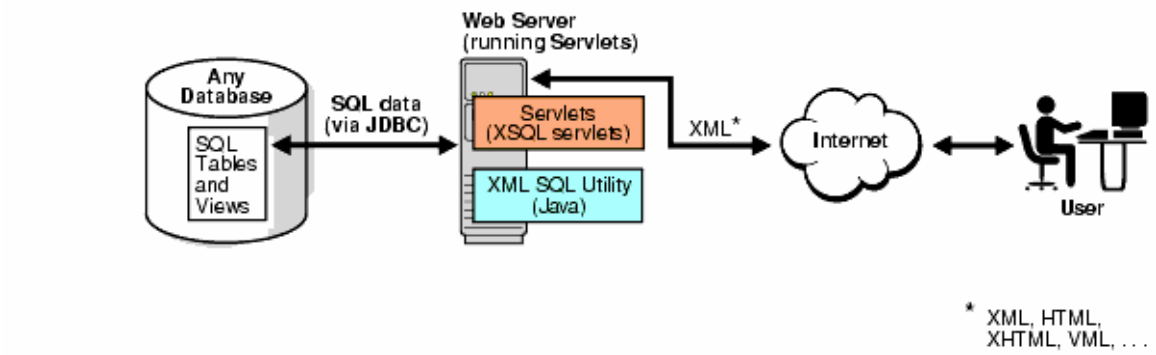

Figure 3. XSU in the Web Server end

XSU can also be installed in the client tier.

XSU can be invoked from a

- Java application or through the command prompt using java commands.
- Write PL/SQL applications that access XSU through its PL/SQL API
- Access XSU functionality directly through SQL

The following is an example of XML document generated for an SQL query on the employee table whose schema is

```
CREATE TABLE EMP
(
   EMPNO NUMBER,
   ENAME VARCHAR2 (20),
   JOB VARCHAR2 (20),
   MGR NUMBER,
   HIREDATE DATE,
   SAL NUMBER,
   DEPTNO NUMBER
);
```

Assume we have certain records in the emp table. Now, given the query,

“`select * from emp`”; XSU can generate the following XML document

```
 <?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
```

```
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <! -- additional rows ... -->
</ROWSET>
```

The <ROWSET> element contains one or more <ROW> elements .Each tag within the <ROW> corresponds to a column in the table and the attribute num=”1” signifies that this is the first record in the emp table. XSU also provides facilities where in a user can set the tag names to his desired names in the output XML document.

2.9 JAVA support for XSU

The following two classes make up the XSU Java API:

- API for XML generation: `oracle.xml.sql.query.OracleXMLQuery`
- API for XML save, insert, update, and delete: `oracle.xml.sql.dml.OracleXMLSave`

One can use either the JAVA command line or a JAVA program to obtain the desired results. Shown below is the manner in which we get the results in XML using command line invocation through JAVA. For example, to generate an XML document by querying the EMP table, the getXML command is used as follows:

```
java OracleXML getXML -user "scott/tiger" "select * from EMP"
```

Here the user name and the query are passed as arguments to the getXML command.

To insert an XML document into the `EMP` table, the putXML command is used with the following syntax:

```
java OracleXML
putXML -user "scott/tiger" -fileName "/tmp/temp.xml" "emp"
```

We pass the username, the name of the XML file to be inserted and the table into which the file is to be inserted.

There are several other options associated with these commands to display XML in the desired format such as giving the output tag names as well as set the column names.

We can also use the XSU in an application as shown

```
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;

// class to test the String generation!
class testXMLSQL {

   public static void main(String[] argv)
   {

     try{
      // create the connection
      Connection conn  = getConnection("scott","tiger");

      // Create the query class.
      OracleXMLQuery qry
               = new OracleXMLQuery(conn, "select * from emp");

      // Get the XML string
```

```
      String str = qry.getXMLString();

      // Print the XML output
      System.out.println(" The XML output is:\n"+str);
      // Always close the query to get rid of any resources.
      qry.close();
     }catch(SQLException e){
      System.out.println(e.toString());
     }
   }

   // Get the connection given the user name and password..!
   private static Connection
          getConnection(String username, String   password)
     throws SQLException
   {
      // register the JDBC driver..
      DriverManager.registerDriver
                  (new oracle.jdbc.driver.OracleDriver());


      // Create the connection using the OCI8 driver
       Connection conn =
        DriverManager.getConnection
                       ("jdbc:oracle:oci8:@",username,password);

      return conn;
   }
}
```

Using a JAVA program one can obtain the XML output as a DOM Object output or XML String output.

For obtaining the output as a DOM object we use the following command

```
org.w3c.DOM.Document domDoc = qry.getXMLDOM();
```

For obtaining the output as an XML String we use the following command.

```
String xmlString = qry.getXMLString();
```

2.10 Inlining Technique

This technique is used to create a relational schema given an XML DTD. Some of the content below is taken from [4].

2.10.1 Mapping a DTD to relational schema

The goal is to store any document conforming to the DTD in the relational schema and any XML semi structured query over a document conforming to the DTD can be evaluated over the relational database resulting in same data. While mapping this way, one needs to also consider the position of an element relative to its siblings and parent-child relationship between elements in the XML document. To accomplish this goal a complex DTD is simplified using certain transformations.

Flattening transformations: These convert the nested definition into a flat representation

Eg: `(e1,e2)* ------→ e1*,e2*`
`(e1,e2)? -----→ e1?, e2?`

Simplification transformations: These convert many unary operators into a single unary operator.

Eg: `e1** -----------→ e1*`
`e1*? -----------→ e1*`
`e1?* -----------→ e1*`
`e1?? -------------→ e1?`

Although some ordering may be lost here, we can when loading the data preserve that ordering by adding certain fields.

2.10.2 The Algorithm

Based on a node graph created for a DTD, the nodes with an in-degree of one are inlined. The nodes which have in-degree of zero are made into separate relations as they cannot be reached from any other node. Relational data model does not support set valued attributes. If an XML element has * occurrences of a child, we cannot create a

column for the child, with parent element name as a table, as it results in many values for one column. The best way to handle such situations is to create a table for the parent element with its attributes, child elements, which have only one occurrence, attributes of child elements if any as columns of the table. For each child which has a * occurrence in the DTD, create a separate table with its attributes, its child elements and their attribute names as column names. Then we are to link the parent table with the table created for a * occurrence child by adding certain fields. Each record in a table will be uniquely identified by a field called ‘id’ which tells about the index of occurrence of the element in the XML document. Tables which have dependencies, i.e tables which correspond to XML elements which have * occurrences, have an additional ‘parent id’ field which references the id field of its parent table. This way one can preserve the ordering and solve the problem of set valued attributes. This is illustrated in the following example.

The following is the DTD of a book database XML document.

```
<!ELEMENT books (book*, library)>
<!ELEMENT book (booktitle, year)>
<!ELEMENT booktitle (bookname,header*,color)>
<!ELEMENT year (monthpub, datepub)>
<!ELEMENT monthpub (#PCDATA)>
<!ELEMENT datepub (#PCDATA)>
<!ELEMENT bookname (#PCDATA)>
<!ELEMENT header (hdrsize)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT hdrsize (#PCDATA)>
<!ELEMENT library (#PCDATA)>
<!ATTLIST book author CDATA #REQUIRED>
```

The root of this DTD is element books. The schema created in the database using the above technique is as follows.

```
Create table books(
booksid int,
library varchar (25),
primary key (booksid))

Create table book(
bookid int,
author varchar(25),
bookname varchar(25),
color varchar(25),
monthpub varchar(25),
datepub varchar(25),
parentid int,
primary key (bookid),
foreign key (parentid) references books)

Create table header(
headerid int,
hdrsize varchar(25),
parentid int,
primary key (headerid),
foreign key (parentid) references book)
```

The root element ‘books’ is converted to a table and the single occurrence child library to a column name. The element ‘book ‘is a multiple occurrence child, so a separate table has been created for it, with its children as column names. The element book has an attribute ‘author’, hence we have a column name author in the book table. Since the element ‘booktitle’ has an in-degree of one, it is inlined into the ‘book’ table,

but its children namely ‘bookname’ and ‘color’ are text nodes, so we need to store that information. Hence there are column names for those elements in the book table. Similar is the case with the ‘year’ element. The element ‘header’ had it been a single occurrence type, its children like ‘hdrsize’ would be included in the book table. But, since it is a multiple occurrence child we are separating it into another table. Here the default data type is taken as varchar (25) for simplicity. Each table has a column name ‘id’ which uniquely identifies each record. The multiple occurrence child tables have a field ‘parent id’ which references its parent table.

3. SYSTEM ARCITECTURE

This chapter gives an overview of the system developed and its chief components. It also speaks about the flow of the application as per the components and the limitations used in the thesis.

3.1 Components and flow

The System comprises of the components shown in the figure. The DTD checker accepts a DTD document, validates the file against certain scope check conditions and accepts the DTD if the check passes. It then creates the relational schema corresponding to the DTD and the tables are created. Then the Data Loader component accepts the valid DTD and an input XML file. It validates the XML file against the DTD and if the XML file is valid, it loads the data in the XML format into the database in relational format. At this stage we have the entire XML document structure and data preserved in the database.

The Data Loader then invokes the Xpath to SQL converter. This prompts the user to enter an Xpath expression. The input expression is validated against certain scope check conditions, and if it passes those, the Xpath expression is converted into an equivalent SQL query and presented to the user. At this stage one can access the database and run the query against it, to get back the results in the relational format. Along with the data, the results also display the ‘id’ and ‘parent id’ fields. Through these fields one can analyze the position of the XML element in the document and relative to its parent.

The Xpath to SQL converter then invokes the XSU XML generator. This takes as input the generated SQL query and runs it against the database to get back the results in the

DOM format or the XML string format. These results in XML are displayed back to the user. This way mapping data from XML to database and vice versa are done.

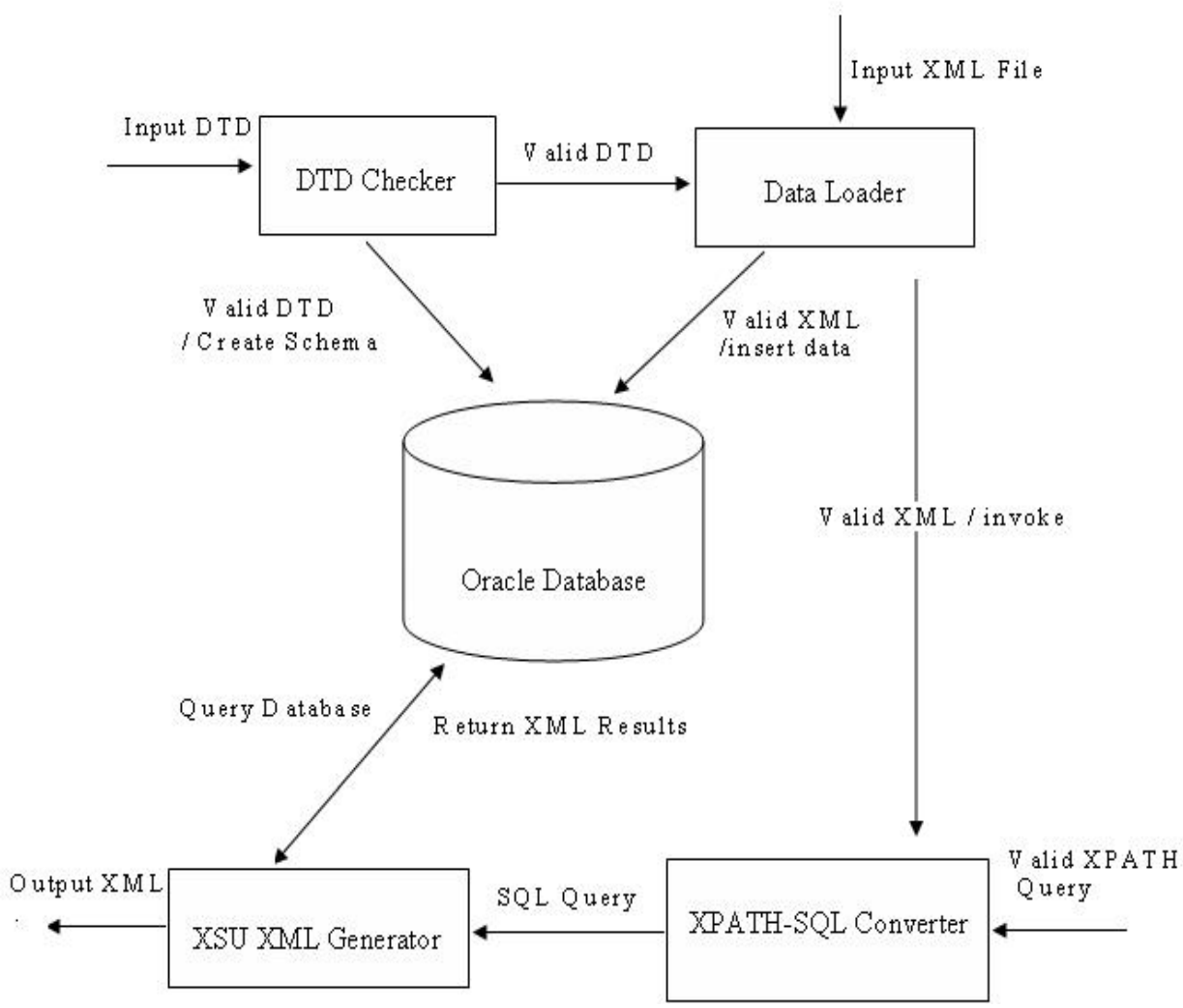

Figure 4. Application flow among system components

3.2 Limitations on the DTD and XPATH imposed in the thesis

Because the aim of the thesis is to show the feasibility of a two way mapping between XML and relational database using the inling technique, the feasibility of

implementing an XPATH to SQL converter, and also due to time limitations, the following constraints are imposed on the DTD's accepted and also the XPATH queries.

A DTD in which the following operators or keywords occur is not accepted to be valid

- ENTITY
- ANY
- | (or) operator
- ID, IDREF

Besides these the DTD needs to be single rooted and elements can't have a list of attributes.

In Xpath, the // operator and functions are not considered. The expressions begin with a / operator.

3.3 Application flow in brief

The application implements the inlining technique to create a relational schema for the XML document based on its DTD. It makes use of the ORACLE supported JAVA API for XML parsing (DOM methodology) by making a depth first traversal through the entire document to load the XML data into the relational database. The application then prompts the user for an Xpath expression and converts it to an equivalent SQL query. One can use this query to obtain results in relational format by connecting to the database from the application or logging into the database. This SQL query is then utilized by XSU to generate the results in XML. At every stage error checking is done and the accepted DTD and Xpath are valuated against the constraints.

4. SOFTWARE DESIGN

This chapter gives a description of the design of the system. The algorithms used, the inputs to the system, the outputs from each stage, the processing and the flow of the application are explained here.

4.1 Loading DTD into application

The application needs some data store of the XML structure for further processing. For this purpose hash tables and vectors in JAVA are used. An input DTD file is accepted as a command line argument. Reading line by line from the file, the application stores elements and attributes details in their respective hash tables. Each has table has a format of Key value pair. This way the parent elements form the keys and child elements form the values. When reading through the file, if ENTITY and ANY keywords are found, the DTD is rejected; otherwise the application proceeds to the next step.

4.2 Creating Relational Schema for XML

For the purpose of identifying the table names, the column names within a table and checking the presence of recursion in the DTD, a module is designed. The root of the document is first passed to the module. The child elements of the root are fetched from the hash table filled out in the earlier section. For each of those child elements the same process is repeated, by storing the column names in vectors until there are no more elements to process. If an element has an attribute, it is recognized as a column. If it is a

text node, it is recognized as a column name. If the application encounters multiple occurrence elements they are treated as special elements and stored in a separate vector.
It is in this module that the presence of | (or) operator, ID, IDREF fields are checked.
If any of those are found the DTD is not accepted. When the control returns to the called function from the module for the root element pass, the application has details of all column names for the root element and all tables that are to be created for the multiple occurrence elements. If there are no columns detected for the root element, then no table is created for it. Here we are creating columns having data type as varchar for the sake of simplicity as it supports both numbers and characters. The application then invokes the module for each of the special elements found, to detect its column names and possibly the presence of multiple occurrence elements with in it. If such elements are found the same process is repeated yet again even for those elements. After all these, the application has the list of all table names and their corresponding column names to be created. As the application loops through the child elements, it also stores the hierarchy in vectors so that, this can be used in future to detect the parent elements. Thus the presence of recursion within the DTD is also checked. Once the control returns of this module for all cases, the application checks if the DTD is single rooted and if so, the application connects to the database using JDBC and creates the tables. If not, the DTD is rejected.

4.3 Loading Data into Tables

Oracle supports JAVA API for XML parsing. The reason for choosing DOM API here is that, there will be multiple passes through the XML document in order to load the data, as the algorithm used is a depth first traversal. Since DOM creates a document node

tree in the memory, it best suits the needs. The application accepts an XML file as the input.

Given the DTD, the DOM parser validates the XML file against the DTD and reports an error if it violates the DTD. The function designed for this purpose executes recursively

starting with passing the root of the XML document to it, and runs over the roots first child and continues that way with that child becoming the root for the next iteration in order to determine its children. Here is a small snippet of the code which is used to traverse the entire XML document.

```
Void load (root)
{
for(child=(XMLNode)root.getFirstChild();child!=null;
                        child=(XMLNode)child.getNextSibling())
// at the end of this loop, we invoke the function with load
(child)
}
```

The following code specifies handlers for different types of child nodes; it can be a text node or an element node.

```
if (child.getNodeType()==child.TEXT_NODE)

 if(child.getNodeType()==child.ELEMENT_NODE)
```

If there was a table created for the root, during the schema creating process, then we store that table name in a global variable for this function. In every iteration, we check if the root which is a parameter to this function, has any attributes, if so, then we have a column value to be inserted, so we create an entry for it taking a hash table where in we have the

key as the table name and value as the attribute value. If the child is identified to be a text node, then the value of the node is a column value, the application modifies the hash table for insert statements with that table name, by adding this new value. How do we identify which column name belongs to which table? It is done as follows.

If the child is an element node, then the application has to detect the presence of a table name while traversing the document so that it collects values of the column names for respective table names. For this purpose, as we have other table names only for multiple occurrence elements, while traversing the document we need to search for multiple occurrence elements cases, by looking at an elements next sibling and previous sibling. If the element name is same as either of these, then the application changes the value of table name variable, and proceeds through the same logic as described above. In some cases even though the element is declared to a multiple occurrence one in the DTD, it occurs only once in the XML document. In such cases, we need to check the hash table created for the table names during the schema creating process.

In this process the application needs to identify two fields namely ‘id’ and ‘parent id’. Once we finish a row of values for a particular table name, if the application identifies one more row, then we increment the id field for that table name and store it in a hash table. We fill in the parent id field for each table name, once the application detects a new table name. Using the hierarchy vectors for table names and another hash table which stores the occurrence of the table names, we can determine these; this will be illustrated in the code.

The output of this function is stored in a hash table which contains all the insert statements in the SQL format. When the control returns to the calling function, we loop through this hash table, referring to the order in which table names ought to occur as per the DTD, fetching the insert statements for each table one by one. The application then connects to the database and performs the insert operations. So at the end of this we have created the schema and loaded all the data in the tables.

4.4 XPATH to SQL conversion

The application prompts the user to enter an Xpath expression. Once the user enters the expression, the application has to compute three clauses namely the select clause, the from clause and the where clause. The application identifies the select clause by looking at the last fragment of expression in the Xpath expression. However, the application adds the fields of id, and parent id so that we can determine exactly where the element occurs in the XML document. The application splits the Xpath expression based on the operator '/'. A check is done of the hierarchy of elements mentioned in the Xpath expression. This is checked against the hierarchy vectors and if does exist, then it is a valid expression. Each of the fragments which are split based on '/' operator may or may not contain a predicate. If it does contain there will be presence of [] operator. So if there are such operators, then we get the where clause from those predicates. Inside the [] operator there can another Xpath expression where there may @ operator or a specifying some expression of the form LHS =RHS. Then we are to determine which table name is that LHS a column of. For that we make use of hierarchy vectors, table names hash tables. The last part of LHS is the desired column name. Once we get the table name we

add the condition table name. column name= RHS to the ‘where’ clause. The ‘from’ clause and where clause are interdependent. If in the where clause there exists a condition of LHS=RHS, Then that table name will become a part of the from clause of it doesn’t already exist. This way all the fragments in the Xpath expression are looked into to determine from and where clause. The application thus converts the Xpath expression into an equivalent SQL query and presents it to the user. The user can then login to the database and query the database using this query to get back the results. The application can also directly connect to the database to achieve the same functionality.

4.5 XSU’s XML generation

Once the application has the SQL query, since the results are also to be displayed in the XML format, the Oracle supported XSU tool is used for this purpose. This accepts the SQL query as the input, connects to the database, runs the query, gets the results in relational format and converts the results to XML and displays the XML format output to the user. For code details refer to the background section in thesis. Here we can configure several options that are supported by the XSU to display the output in the desired format to the user. Thus the application achieves the backward mapping of relational to XML format.

5. IMPLEMENTATION AND TESTING

This chapter presents the test cases and displays the GUI and the outputs generated by the program.

Example 1: Given below is a DTD of a books database XML document. We give this as one of the inputs to the program along with a XML file.

DTD

```
<!ELEMENT books (book*,library)>
<!ELEMENT book (booktitle,year)>
<!ELEMENT booktitle (bookname,header*,color)>
<!ELEMENT year (monthpub,datepub)>
<!ELEMENT monthpub (#PCDATA)>
<!ELEMENT datepub (#PCDATA)>
<!ELEMENT bookname (#PCDATA)>
<!ELEMENT header (hdrsize)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT hdrsize (#PCDATA)>
<!ELEMENT library (#PCDATA)>
<!ATTLIST book author CDATA #REQUIRED>
```

The following is the XML file given as another argument to the program at the command line. This XML file is to confirm with the DTD given and this check is made in the application.

XML Document

```
<?xml version="1.0"?>
<!DOCTYPE books SYSTEM "book.dtd">
<books>
<book author="james">
 <booktitle>
```

```
 <bookname>Marine Biology</bookname>
  <header>
    <hdrsize>20</hdrsize>
  </header>
  <header>
   <hdrsize>10</hdrsize>
  </header>
  <color>blue</color>
 </booktitle>
 <year>
  <monthpub>may</monthpub>
  <datepub>18th</datepub>
 </year>
</book>
<book author="Foster">
 <booktitle>
 <bookname>Mass Communications</bookname>
  <header>
    <hdrsize>30</hdrsize>
  </header>
  <header>
   <hdrsize>20</hdrsize>
  </header>
  <color>Orange</color>
 </booktitle>
 <year>
  <monthpub>may</monthpub>
  <datepub>25th</datepub>
 </year>
</book>
<book author="Kimberley">
 <booktitle>
 <bookname>Organic Chemistry</bookname>
```

```
  <header>
    <hdrsize>30</hdrsize>
  </header>
  <header>
   <hdrsize>15</hdrsize>
  </header>
  <color>Red</color>
 </booktitle>
 <year>
  <monthpub>jun</monthpub>
  <datepub>18th</datepub>
 </year>
</book>
<book author="Jacob">
 <booktitle>
 <bookname>Philosophy</bookname>
  <header>
    <hdrsize>20</hdrsize>
  </header>
  <color>Grey</color>
 </booktitle>
 <year>
  <monthpub>feb</monthpub>
  <datepub>19th</datepub>
 </year>
</book>
<library>Central Library</library>
</books>
```

Given these two arguments to the program, the following output is generated. The output schema generated are displayed first and then the insert statements are displayed.

The relational schema equivalent to the DTD is

```
create table books(
booksid int,
library varchar(50),
 primary key (booksid))


create table book (
bookid int,
author varchar(50),
bookname varchar(50),
color varchar(50),
monthpub varchar(50),
datepub varchar(50),
parentid int,
primary key (bookid),
foreign key (parentid) references books)

create table header (
headerid int,
hdrsize varchar(50),
parentid int,
primary key (headerid),
foreign key (parentid) references book)
```

The program now goes into the data loading stage with the following messages being displayed.

finished parsing the XML document, loading data into relational database.....

```
insert into books values(1,'Central Library')
insert into book values(1,'james','Marine
Biology','blue','may','18th',1)
insert into book values(2,'Foster','Mass
Communications','Orange','may','25th',1)
```

```
insert into book values(3,'Kimberley','Organic
Chemistry','Red','jun','18th',1)
insert into book
values(4,'Jacob','Philosophy','Grey','feb','19th',1)

insert into header values(1,'20',1)
insert into header values(2,'10',1)
insert into header values(3,'30',2)
insert into header values(4,'20',2)
insert into header values(5,'30',3)
insert into header values(6,'15',3)
insert into header values(7,'20',4)
```

The data loading stage is now completed. At this stage there are tables generated in the database.

BOOKSID	LIBRARY
1	Central Library

Table 2. Books

BOOKID	AUTHOR	BOOKNAME	COLOR	MONTHPUB	DATEPUB	PARENTID
1	James	Marine Biology	Blue	May	18th	1
2	Foster	Mass Communications	Orange	May	25th	1
3	Kimberly	Organic Chemistry	Red	Jun	18th	1
4	Grey	Philosophy	Grey	Feb	19th	1

Table 3. Book

HEADERID	HDRSIZE	PARENTID
1	20	1
2	10	1
3	30	2
4	20	2
5	30	3
6	15	3
7	20	4

Table 4. Header table

The program is tested against some sample queries shown below.

SAMPLE QUERIES

1. enter an xpath expression, You will get back an SQL query

```
/books/book/booktitle/bookname
```

the equivalent SQL query is.

```
select bookid,bookname,book.parentid
from book
```

You can login to database and run this query to get results in relational format. The following is the output generated because of the XSU.

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <BOOKID>1</BOOKID>
      <BOOKNAME>Marine Biology</BOOKNAME>
      <PARENTID>1</PARENTID>
   </ROW>
   <ROW num="2">
      <BOOKID>2</BOOKID>
      <BOOKNAME>Mass Communications</BOOKNAME>
      <PARENTID>1</PARENTID>
   </ROW>
   <ROW num="3">
      <BOOKID>3</BOOKID>
      <BOOKNAME>Organic Chemistry</BOOKNAME>
      <PARENTID>1</PARENTID>
   </ROW>
   <ROW num="4">
      <BOOKID>4</BOOKID>
```

```
      <BOOKNAME>Philosophy</BOOKNAME>
      <PARENTID>1</PARENTID>
   </ROW>
</ROWSET>
```

Explanation of the query: The query is to generate all the names of the books. The field bookname is under booktitle which is under book which is a table. If we look at the output XML and compare it with the XML document we notice the tag name <BOOKNAME> and its corresponding value. The field BookID indicates which occurrence of bookname is it starting from the first bookname which follows the given hierarchy. For example, the Bookname “Philosophy “ has Bookid 4, which means to say, it is under the fourth book under consideration and the PARENTID 1 corresponds to this book having a parent whose ID is 1.

2. Queries with attribute operator

enter an xpath expression, You will get back an SQL query

```
/books/book[@author='Foster']
```

the equivalent SQL query is

```
select author,bookid
from book
where book.author='Foster'
```

You can login to database and run this query to get results in relational format

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
```

```
   <ROW num="1">
      <AUTHOR>Foster</AUTHOR>
      <BOOKID>2</BOOKID>
   </ROW>
</ROWSET>
```

Explanation of the query: The query is to identify the book tag with author name "Foster". If we look at the output XML and compare it with the XML document we notice the tag name <AUTHOR> and BOOKID with their corresponding values. The field BookID indicates which book under the tag BOOKS has an author with name given in the AUTHOR tag. Here it is the second book under BOOKS tag as per the given XML document.

3. Queries using predicate [] operator with expressions in it

enter an xpath expression, You will get back an SQL query

```
/books/book[booktitle|header|hdrsize='15']/year/monthpub
```

the equivalent SQL query is

```
select bookid,monthpub,book.parentid
from book,header
where header.hdrsize='15' and header.parentid=book.bookid;
```

You can login to database and run this query to get results in relational format

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
```

```
    <ROW num="1">
       <BOOKID>3</BOOKID>
       <MONTHPUB>jun</MONTHPUB>
       <PARENTID>1</PARENTID>
    </ROW>
</ROWSET>
```

Explanation of the query: The query is to find the month of publication of a book whose header size is 15. If we look at the output XML and compare it with the XML document we notice the tag name <MONTHPUB>, BOOKID and PARENTID with their corresponding values. The field BOOKID indicates which book under the tag BOOKS satisfies the property that the header size of it is 15. Here it is the third book under BOOKS tag as per the given XML document. PARENTID indicates that this book's parent has ID 1 which is the Tag BOOKS.

In the next example we consider employees, customers, orders database represented in an XML document having the following DTD. We supply the DTD and the XML file as arguments to the program and an XML file is generated as an output when some XPATH expression is entered.

Example 2 DTD

```
<!ELEMENT modb (employees,customers,orders)>
<!ELEMENT employees (employee*)>
<!ELEMENT employee (eno,ename,city,zip)>
<!ELEMENT eno (#PCDATA)>
<!ELEMENT ename (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

```
<!ELEMENT zip (#PCDATA)>
<!ELEMENT customers (customer*)>
<!ELEMENT customer (cno,cname,street,ccity,czip)>
<!ELEMENT cno (#PCDATA)>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT ccity (#PCDATA)>
<!ELEMENT czip (#PCDATA)>
<!ELEMENT orders (orderr*)>
<!ATTLIST orderr ono CDATA #REQUIRED>
<!ELEMENT orderr (takenBy,cno,receivedDate,shippedDate,items)>
<!ELEMENT takenBy (#PCDATA)>
<!ELEMENT receivedDate (#PCDATA)>
<!ELEMENT shippedDate (#PCDATA)>
<!ELEMENT items (item*)>
<!ELEMENT item (partNumber,quantity)>
<!ELEMENT partNumber (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
```

Given below is an XML file which confirms with the above DTD. This check is performed in the application as part of XML parsing in JAVA.

XML FILE

```
<?xml version="1.0"?>
<!DOCTYPE modb SYSTEM "mailorder.dtd">

<modb>
  <employees>
    <employee>
      <eno>1000</eno>
      <ename>Jones</ename>
```

```
    <city>Wichita</city>
    <zip>67226</zip>
  </employee>
  <employee>
    <eno>1001</eno>
    <ename>Smith</ename>
    <city>Fort Dodge</city>
    <zip>60606</zip>
  </employee>
  <employee>
    <eno>1002</eno>
    <ename>Brown</ename>
    <city>Kansas City</city>
    <zip>50302</zip>
  </employee>
</employees>
<customers>
  <customer>
    <cno>1111</cno>
    <cname>Charles</cname>
    <street>123 Main St</street>
    <ccity>Wichita</ccity>
    <czip>67226</czip>
  </customer>
  <customer>
    <cno>2222</cno>
    <cname>Bertram</cname>
    <street>237 Ash Avenue</street>
    <ccity>Wichita</ccity>
    <czip>67226</czip>
  </customer>
  <customer>
    <cno>3333</cno>
```

```
    <cname>Barbara</cname>
    <street>111 Inwood St</street>
    <ccity>Fort Dodge</ccity>
    <czip>60606</czip>
  </customer>
  <customer>
    <cno>4444</cno>
    <cname>Jonathan</cname>
    <street>111 Elm St</street>
    <ccity>Fort Dodge</ccity>
    <czip>60606</czip>
  </customer>
</customers>

<orders>
  <orderr ono="1020">
    <takenBy>1000</takenBy>
    <cno>1111</cno>
    <receivedDate>10-DEC-94</receivedDate>
    <shippedDate>12-DEC-94</shippedDate>
    <items>
      <item>
        <partNumber>10506</partNumber>
        <quantity>1</quantity>
      </item>
      <item>
        <partNumber>10507</partNumber>
        <quantity>1</quantity>
      </item>
      <item>
        <partNumber>10508</partNumber>
        <quantity>2</quantity>
      </item>
```

```
    <item>
      <partNumber>10509</partNumber>
      <quantity>3</quantity>
    </item>
  </items>
</orderr>

<orderr ono="1021">
  <takenBy>1000</takenBy>
  <cno>1111</cno>
  <receivedDate>12-JAN-95</receivedDate>
  <shippedDate>15-JAN-95</shippedDate>
  <items>
    <item>
      <partNumber>10601</partNumber>
      <quantity>4</quantity>
    </item>
  </items>
</orderr>

<orderr ono="1022">
  <takenBy>1001</takenBy>
  <cno>2222</cno>
  <receivedDate>13-FEB-95</receivedDate>
  <shippedDate>20-FEB-95</shippedDate>
  <items>
    <item>
      <partNumber>10601</partNumber>
      <quantity>1</quantity>
    </item>
    <item>
      <partNumber>10701</partNumber>
      <quantity>1</quantity>
```

```
        </item>
      </items>
    </orderr>

    <orderr ono="1023">
      <takenBy>1000</takenBy>
      <cno>3333</cno>
      <receivedDate>20-JUN-97</receivedDate>
      <shippedDate>20-FEB-96</shippedDate>
      <items>
        <item>
          <partNumber>10800</partNumber>
          <quantity>1</quantity>
        </item>
        <item>
          <partNumber>10900</partNumber>
          <quantity>1</quantity>
        </item>
      </items>
    </orderr>
  </orders>
</modb>
```

Once the user supplies the DTD and an XML file which corresponds to the DTD, the schema for the DTD is created and data of the XML file is inserted into the database. The following are the results produced by the program.

EMPLOYEEID	ENO	ENAME	CITY	ZIP
1	1000	Jones	Wichita	67226
2	1001	Smith	Fort Dodge	60606
3	1002	Brown	Kansas City	50302

Table 5. Employee Table

CUSTOMERID	CNO	CNAME	STREET	CCITY	CZIP
1	1111	Charles	123 Main st	Wichita	67226
2	2222	Bertram	237 Ash Avenue	Wichita	67226
3	3333	Barbara	111 Inwood St	Fort Dodge	60606
4	4444	Jonathan	111 Elm Street	Fort Dodge	60606

Table 6. Customer Table

ORDERRID	ONO	TAKENBY	CNO	RECEIVEDDATE	SHIPPEDDATE
1	1020	1000	1111	10-DEC-94	12-DEC-94
2	1021	1000	1111	12-JAN-95	15-JAN-95
3	1022	1001	2222	13-FEB-95	20-FEB-95
4	1023	1000	3333	20-JUN-96	20-JUN-96

Table 7. Orderr Table

ITEMID	PARTNUMBER	QUANTITY	PARENTID
1	10506	1	1
2	10507	1	1
3	10508	2	1
4	10509	3	1
5	10601	4	2
6	10601	1	3
7	10701	1	3
8	10800	1	4
9	10900	1	4

Table 8. Item Table

The following queries are run against the XML file and the results are back in XML. The storage of data in the database is transparent to the user.

SAMPLE QUERIES

enter an xpath expression, You will get back an SQL query

```
/modb/orders/orderr[ono='1021']/takenBy
```

the equivalent SQL query is

```
select orderrid,takenBy
from orderr
where orderr.ono='1021'
```

You can login to database and run this query to get results in relational format

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <ORDERRID>2</ORDERRID>
      <TAKENBY>1000</TAKENBY>
   </ROW>
</ROWSET>
```

Explanation of the Query: The query here is to get the employee number who has taken an order with order number 1021. We get this information looking at the TAKENBY tag in the ORDERR tag of the XML file. The output XML generated indicates that the order corresponding to order number 1021 has ORDERID 2 i.e. the second occurrence of ORDERR in the XML document. We have an extra R in the ORDERR tag because of the conflict with keyword ‘order’ in the relational database. We can see that that particular order has been taken by employee 1000.

Query 2

enter an xpath expression, You will get back an SQL query

```
/modb/customers/customer/cname
```

the equivalent SQL query is

```
select customerid,cname
from customer
```

You can login to database and run this query to get results in relational format

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <CUSTOMERID>1</CUSTOMERID>
      <CNAME>Charles</CNAME>
   </ROW>
   <ROW num="2">
      <CUSTOMERID>2</CUSTOMERID>
      <CNAME>Bertram</CNAME>
   </ROW>
   <ROW num="3">
      <CUSTOMERID>3</CUSTOMERID>
      <CNAME>Barbara</CNAME>
   </ROW>
   <ROW num="4">
      <CUSTOMERID>4</CUSTOMERID>
      <CNAME>Jonathan</CNAME>
   </ROW>
</ROWSET>
```

Explanation of the query: The query here is to get the names of all customers. The output XML generated contains the tag CNAME along with the CUSTOMERID. The CUSTOMERID tag gives the position of the customer tag in the XML document. The customer whose name is “Jonathan”, is the 4th customer. Here we have 4 customers in total.

Query 3

Enter a query

```
/modb/orders/orderr[items|item|partNumber='10601']/cno
```

the equivalent SQL query is

```
select orderrid,cno
from orderr,item
where item.partNumber='10601' and
item.parentid = order.orderrid;
```

You can login to database and run this query to get results in relational format

The XML output is:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <ORDERRID>2</ORDERRID>
      <CNO>1111</CNO>
   </ROW>
   <ROW num="2">
      <ORDERRID>3</ORDERRID>
      <CNO>2222</CNO>
   </ROW>
</ROWSET>
```

Explanation of the query: The query here is to get the customer number who has placed an order for a part number 10601. The PARTNUMBER is a tag under ITEM of the ITEMS under ORDERR. The output XML document contains two fields namely the ORDERID and the CNO. All the orders In which the part number 10601 is ordered are given along with the customer numbers who have placed those orders.

6. CONCLUSION AND FUTURE WORK

The system developed addresses the issue of mapping XML data into relational database by creating the schema corresponding to the DTD of a given XML file. ORACLE provides such software but they have not yet delivered one for query mappers. The system developed acts as an automatic converter of XPATH queries to SQL queries and generates the results in relational data format as well as XML format. Using some technologies which are developed earlier, the system presents the novel approach to query mapping. Companies can use such systems to ensure data transfer among them. Users who are good at XML can pose XPATH queries and get back the results in XML, where in the data storage in relational format remains transparent to them. Many organizations which require data transfer from XML to database or vice versa will be benefited by more work in this field.

Future work in this field will be to develop a similar kind of system which solves a broad range of problems. The limitations imposed in the thesis as per the DTD operators and XPATH operators can be removed and a full fledged system can be developed. The use of XML Schema to represent the structure of the XML document and the XQUERY language which is more close to posing queries in the SQL format will address more problems as with the use of XQUERY one can also issue join queries.

BIBLIOGRAPHY

[1] Herbert Schildt, The Complete Reference Java 2, Fifth Edition

[2] Subramanyam Allamaraju, Karl Avedal, Richard Browett, Jason Diamond, John Griffin, JAVA Server Programming J2EE Edition, Volume I

[3] Rajashekhar Sunderraman, ORACLE 9i PROGRAMMING, A PRIMER

[4] Jayavel Shanmugasundaram, Kristin Tufte, Gange He, Chun Zhang, David De Witt, Jeffrey Naughton, *Relational Databases for Querying XML Documents: Limitations And opportunities*

[5] www.w3schools.com

[6] www.xvon.org

[7] www.oracle.com

[8] http://www.cs.umb.edu/cs634/ora9idocs/appdev.920/a96621/adx08xsu.htm

[9] Wenfei Fan, Jeffrey Xu Yu, Hongjun Lu, Jianhua Lu, Rajeev Rastogi, *Query Translation from XPATH to SQL in the Presence of Recursive DTDs*

[10] Daniela Forescu, Donald Kossmann, *Storing and Querying XML Data using an RDBMS.*

[11] Mary Fernandez, Atsuyinki Morishima, Dan Suciu, Wang-Chiew Tan, *Publishing Relational Data in XML: the SilkRoute Approach*

[12] Ronald Bourret, Christof Bornhovd, Alejandro P. Buchmann, *A Generic Load/Extract Utility for Data Transfer between XML documents and Relational Databases*