AUTOMATIC MAPPING OF XML DOCUMENTS INTO RELATIONAL DATABASE

BY

IBRAHIM MOHAMMAD IBRAHIM DWEIB

A thesis submitted to the School of Computing and Engineering
of the University of Huddersfield
for the degree of Doctor of Philosophy

School of Computing and Engineering
The University of Huddersfield
(September 2010)

LIST OF PUBLICATIONS AND RESEARCH ACTIVITIES RELATED TO THIS WORK

Dweib, I., Awadi, A. and Lu, J. (2009) *MAXDOR: Mapping XML Document into Relational Database.* The Open Information Systems Journal, 3 (2). pp. 108-122. ISSN 1874-1339

Dweib, I., Awadi, A., Fath_Elrhman, S. and Lu, J. (2008) *Schemaless approach of mapping XML document into Relational Database.* 8th IEEE International Conference on Computer and Information Technology, 2008. CIT 2008. pp. 167-172.

Dweib, I., Awadi, A, Lu, J. Yip, J. and Allen, G. (2007) *Flexible Approach for Querying XML Document-Centric Documents Stored in Relational Database*, ECIG2007 Conference, Soussi, Tunisia, Oct 2007

Dweib, I., Lu, J. Yip, J. and Allen, G. (2007) *Schema based approach of mapping XML documents to relational database.* In: Proceedings of the 2007 International Conference on Internet Computing. CSREA Press, pp. 332-338. ISBN 1601320442

Dweib I., *Automatic Mapping of XML Document to Relational Database Management System*, poster, 2008, research festival, University of Huddersfield

ABSTRACT

Extensible Markup Language (XML) nowadays is one of the most important standard media used for exchanging and representing data through the Internet. Storing, updating and retrieving the huge amount of web services data such as XML is an attractive area of research for researchers and database vendors. In this thesis, we propose and develop a new mapping model, called MAXDOR, for storing, rebuilding, updating and querying XML documents using a relational database without making use of any XML schemas in the mapping process. The model addressed the problem of solving the structural hole between ordered hierarchical XML and unordered tabular relational database to enable us to use relational database systems for storing, updating and querying XML data. A multiple link list is used to maintain XML document structure, manage the process of updating document contents and retrieve document contents efficiently.

Experiments are done to evaluate MAXDOR model. MAXDOR will be compared with other well-known models available in the literature (Tatarinov et al., 2002) and (Torsten et al., 2004) using total expected value of rebuilding XML document execution time and insertion of token execution time.

DEDICATION

To my parents, brothers, sisters, wife, sons, and daughter with love

ACKNOWLEDGEMENTS

At the outset I wish to thank Allah, the Almighty for His blessings bestowed upon me with the strength to complete this thesis.

I would like to convey my sincere thanks to my grateful supervisor, Dr. Joan Lu for her precious guidance and supervision that enabled me to complete this thesis. Without her guidance and dedication this work would not have seen the light. Also I would like to thank my co-supervisors Dr Gary Allin and Prof. Jim Yip for their kind coordination and guidance while carrying out my research.

My thanks go also to all the staff at the Research Studies, University of Huddersfield for their cooperation and helpfulness in academic matters.

Also I am especially thankful to my family for their endless prayers and moral support. I dedicate this thesis to them. I am also grateful to my loyal friends who had helped me all along to accomplish this task, Dr. Walid Abu Dayyeh, Ayman Alawadi, Yasir Allaham, and Hussien Almegbali.

Last, but not the least, great thanks to all who had been involved directly and indirectly throughout the making of this thesis. I really appreciate it. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

ABBREVIATIONS & TERMINOLOGIES

Abbreviation	Details
DOM	Document Object Model
DTD	Document Type Definition
ER	Entity-Relationship diagram
FDLs	Four Dimensional Links
GUI	Graphical User Interface
MAXDOR	MApping XML Document intO Relational database
RDB	Relational Database
RDBMS	Relation Database Management System
SAX	Simple Application Interface for XML
SQL	Sequel Query Language
Token	Represent element or element's attribute
Tuple	Row or record
XML	eXtended Markup Language
XSLT	Extensible Stylesheet Language Transformations
XPATH	XML Path Language
XQUERY	XML Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium

CHAPTER 1 INTRODUCTION

The World Wide Web (WWW) nowadays is an important medium used by many people for many activities in their daily life (i.e.; e-management, e-learning, e-mail, e-library and e-business). Many enterprises are working together using XML technologies for exchanging their web services data. Exchanging, sorting, updating and retrieving these huge data has become a source of concern for researchers and database vendors.

At present, storing and retrieving of XML documents can be done using mainly three approaches, i.e., native XML database (Jagadish et al., 2003;M. Grinev et al., 2004), Object Oriented Database (Chung and Jesurajaiah, 2005) and Relational Database (Zhang and Tompa, 2004a;Shanmugasundaram et al., 1999); (Fujimoto et al., 2005;O'Neil et al., 2004) (Tan et al., 2005) (Leonardi and Bhowmick, 2005;Atay, 2006;Atay et al., 2007a;Min et al., 2008,Yun and Chung, 2008;Ahlgren and Colliander, 2009) .

The most important factor in choosing the target database is the type of XML documents to be stored, data-centric (e.g., bank transaction, airlines transactions) or document-centric (e.g., emails, books, manual).

Using a hybrid approach of relational database to store and retrieve data and XML to exchange and represent it. This will solve most of the data issues of integrity, multi-user access, retrieving, exchanging, concurrency control, crash recovery, indexing, security, storing semi-structure data, and reliability. The previous studies of this approach can also be studied. These are: Loss of information, difficulties in updating its contents and difficulties in rebuilding of original document. The mapping techniques of this approach can generally be classified into two tracks: Schemaless-centric technique and schema–centric (Dweib et al., 2008). Schemaless-centric technique is used to make use of XML document structure to manage mapping process (Zhang & Tompa, 2004; Yoshikawa et al., 2001; Jiang et al., 2002; Tatarinov et al., 2002; Soltan and Rahgozar, 2006). In schema–centric, XML schema information is used to develop a relational storage for XML documents (Shanmugasundaram et al., 1999; Atay et al., 2005; Yahia et al, 2004; Lee et al, 2006; Knudsen et al., 2005; Fujimoto et al., 2005, Xing et al., 2007). Unfortunately, relational storages constructed from schema-centric approach need database reconstruction as any change in the XML schema is very expensive. Each approach introduced some solutions for the mapping process but failed to solve others.

In this thesis we will concentrate on a new approach for mapping XML documents into relational database which is called MAXDOR (i.e. Mapping XML Document into Relational database). The model does not make use of any XML schemas to manage mapping process. In this model, the document structure and document contents are stored in relational database tables. It uses multi-links to reserve document structure and elements relations within the document as parent-child, ancestor-descendant, left- sibling and right-sibling. The use of multi-links will make the insertion process cost for new elements and attributes any where in the document close to constant value, since there is no need to relabel the elements and the attributes following the inserted element or attribute. Other models (Tatarinov et al., 2002) (Torsten et al., 2004) which consider the element or attribute label as an identifier to reserve document structure, the cost of insertion in this case will vary depending on the position of insertion, since relabeling is needed after each insertion to maintain the document order.

The proposed model uses a process of four steps: (1) Mapping XML document into relational database. To achieve this objective, a fixed relational schema is presented and used to maintain document contents relations and manage the contents. (2) Building XML document from relational database without a need to the original document. To achieve this objective, the document contents are retrieved from the relational database and a new XML document file is created for it, and its name is represented by the document identification. (3) Updating XML document contents within the relational database without going back to the original document. To achieve this objective, an editor is created to browse the document as tree structure with a tool bar identifying the position of insertion for the new token in reference to the candidate token. (4) Querying and retrieving document contents through the use of XPath language. To achieve this objective, an editor is created to write the XPath expression, execute and display the results as tree view and grid view.

1.1. Problem Definition

The transformation method of XML documents to RDB should fulfill many requirements while each requirement is to fulfil certain application needs. In some applications it is extremely important to maintain nodes' order such as properties of an XML tag. However, in others order is not so significant. Some of these requirements are the following:

1. Maintain document structure without losing information during shredding.
2. Ease of process, transforming a fresh document should be an easy task, and updating an already transformed document should also be straight forward.
3. To reconstruct the XML document or part of it from relational database.
4. To perform semantic search.
5. To preserve the ordering nature of XML data and its structure.

From previous sections, it can be seen that some studies work on optimizing query time (Torsten, 2002;Soltan and Rahgozar, 2006), but they fail to update XML document stored in relational database. That is because each insertion requires a lot of nodes to be relabelled after insertion of new node or subtree. Others (Chung and Jesurajaiah, 2005;Li and Moon, 2001;O'Neil et al., 2004) solve partially the updating problem by creating a gap within the label, but there is still a need for relabeling after consuming the reserved space. Other studies (Fujimoto et al., 2005;Shanmugasundaram et al., 1999;Tan et al., 2005;Chen et al., 2003;Amer-Yahia et al., 2004;Xing et al., 2007a;Atay et al., 2007b) work on storage optimization and create a relational schema depending on XML schema. Redundant data are removed by creating new relation for each recursive child (or inlining some child in parent relation to reduce the number of created relation). Sometimes a large number of relations are needed to be created for some complex document. Consequently, large numbers of joins are needed to retrieve document information from a relational database. Also sometimes XML schema is not available for some documents which require reconstructing XML schema first from document structure, and creating relational schema based on it. XML reconstruction is considered as a time overhead in this case. In some studies like (Zhang and Tompa, 2004b), they do a map for some parts of the XML document. They used the query to optimize the mapping time from XML document to relational database. They did not store the entire content of a document in a relational database. This method requires a mapping for each query, and can not make use of other data stored in relational database.

It can be concluded that there is still a problem while updating an XML document content stored in relational database. A lot of data in a relational database is needed to be overwritten after inserting each new element or attribute in XML document. That is done to maintain XML document structure and reserve elements and elements' attributes order within the document.

1.2. Research Aim

The aim of this research is to minimize the updating execution time cost of XML document without affecting its structure. It seeks to achieve this aim throughout fulfilling the following two goals:

- Building XML document contents relations in an efficient way to maintain document structure and minimize updating execution cost.
- Forwarding queries to a subset of nodes that is most likely to have relevant information.

The above goals are achieved in the current research by the following objectives:

1. Relational engine will not be modified that may result in consistency problem.
2. The model will be efficient and will perform well for large XML documents.
3. The model is schema-independent. The model design does not depend on the schema information for the mapping process, since relational storages based on schema-centric approach need database reconstruction as any change in the XML schema.
4. Identify fixed relational schema to reserve XML document contents and structure depending on the previous objective.
5. Build XML document from relational database after updating its contents without significant difference in the execution time of building the original one.

6. Make the scheme of objective 2 applicable to queries, in such away that a query is forwarded to a set of nodes that cache information about desired XPath expression.

1.3. Contributions

The following are the main contributions presented throughout this thesis:

XML document mapping into relational database: a novel method is introduced to partition XML document into tokens (elements and attributes). It relies on assigning a tuple in relational table for each token information and relations with its neighbours. The method works efficiently and performs well for large XML documents.

Building XML document from relational database: a novel method is introduced to rebuild original XML document or update one from relational database. It relies on retrieving document contents depending on token links and token level which formulate XML document as a group of subtrees.

Updating XML document contents: a novel method is used to update (i.e. insert new token or modify its name or value) XML document contents stored in relational database. It is based on creating links for each token with its neighbours to maintain document structure without a need to relabel or re-index document contents.

Querying and retrieving XML document: a novel method is introduced to access most of XPath axes preceding-sibling, following-sibling and descendant without storing all possible XPath information for document contents. It relies on creating a dummy table “XPathQuery table” for the desired XPath expression storing all interested tokens.

1.4. Thesis Outline

We present a brief outline of the thesis:

In Chapter 2, the research background is discussed. This includes XML model, XML query languages, XML schema languages, XML Application Program Interface, XML documents types, XML data storage approaches, relational database model, and the similarities and differences between XML model and relational database model.

In Chapter 3, the approaches for storing XML documents in relational databases and for querying and retrieving XML Data from relational databases will be discussed according to their classification into schema-based mapping and schema-less mapping. Commercial Database Management System such as, DB2, Oracle, and SQL Server solutions to support XML will be discussed and reviewed. Rebuilding XML from RDBMS, their issues and approaches will be reviewed. Comparison of mapping approaches, their advantages and disadvantages will be discussed in the last sections.

In Chapter 4, a full description of a novel model is given and introduced in the thesis for Mapping XML Document into Relational database. This is called MAXDOR. This includes the main mathematical concepts that are used in this model. A description of the labelling method used to label the XML document and identifying its contents, the design framework for maintaining document structure, (i.e. parent-child, ancestor-descendant and siblings relations) between document contents is given. Mapping XML to relational database algorithm, building XML document from relational database algorithms using SAX parser, and updating of XML document contents which is stored in relational database algorithm are presented.

Translating XPath query to SQL statements algorithm is included along with the query results in XML format.

In Chapter 5, a presentation of the system architecture, and the tools used for implementing the system of MAXDOR model is given. Theory implementation on a case study is also presented. The main classes for mapping XML document into relational database, building XML document from relational database, updating XML document contents stored in relational database and XpathToSql query translation and building the result in XML format methods, are also presented. XML data sets from selected XML bench marks and XML data repository will be identified to be used for testing and evaluating the model.

In Chapter 6, a description of the experiment setup is given through experiment environment and performance measurement. In fact, a set of experiments are performed on mapping XML document into relational database, building XML document from relational database, updating XML document stored in relational database and retrieving document contents from relational database using XPath expressions. These experiments are performed to check the scalability and effectiveness of our model. Then, the model will be compared with the Global Encoding model (Tatarinov *et al.*, 2002) and the Accelerating XPath model (Torsten *et al.*, 2004). The comparison is performed in four stages of mapping, building, updating and retrieving, since the other studies just took one or two stages and did not address the others. Some took retrieving, while others took updating or updating and retrieving, but most of them did not consider mapping and rebuilding.

Finally, **in Chapter 7**, a summary of the thesis and discussion of further research directions are presented.

CHAPTER 2 RESEARCH BACKGROUND

In this chapter the research background will be discussed. This includes XML model, XML query languages, XML schema languages, XML Application Program Interface, XML documents types, XML data storage approaches, relational database model, and the similarities and differences between XML model and relational database model. Finally the chapter summary is given.

2.1. XML Model

"*EXtensible Markup Language (XML), is a W3C Recommendation in 1998 for marking up data*" (Bray et al., 2007). It is designed for publishing and exchanging a large scale of digital data over the Internet. It is a Markup language that is used to define the structure of information and its elements' contents, where HTML is used to define the way in which the elements are displayed on a web page. It can also be considered as an ideal format for server-to-server transfer of structured data (Bansal and Alam, 2001).

The importance of XML documents transformation is largely increased. Moreover different XML models have common requirements and limitations as tools for data management. For rich data to be shared among different groups, all concepts need to be placed into a common frame of reference. XML schemas must be globally standardized among groups, or mapping must be created between all pairs of related data. Parsing and text conversion slows down the access of the data.

A well-formed XML document is one that corresponds to the XML 1.0 (Bray *et al.*, 2007) grammar specified by W3C. It has exactly one root element, which is called document element. Each starting element tag should have a corresponding closing tag. The elements should be nested within one another. The tags and nesting rules allow XML to represent information in a hierarchical manner. ***Figure 2.1*** shows an example for valid XML document.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Books>
        <Book Price="39.99" id="101">
                <Name>Visual Basic programming</Name>
                <Authors>
                        <Author id="A100">Tom, Criss</Author>
                        <Author id="A150">Jim, Divad</Author>
                </Authors>
                <ISBN>1254315121</ISBN>
        </Book>
        <Book Price="59.99" id="102">
                <Name>Visual C# with SQL</Name>
                <Authors>
                        <Author id="A150" >Mike, Roudy</Author>
                </Authors>
                <ISBN>487524545</ISBN>
        </Book>
</Books>
```

Figure 2.1: An example of XML document

In recent years, significant development in the XML domain has been achieved. Many languages based upon XML Markup have been designed; XML Schema and XML XQuery have been developed. These standardized technologies augment the data processing abilities of XML. The following sections give a brief description of a variety of XML based languages and technologies.

2.2. XML Query Languages

XML query languages are used to enable the user to retrieve data from a single XML document using XPath language, or from multi-documents using XQuery language.

2.2.1 XPath Language:

XPath stands for the XML Path Language(Berglund et al., 2007). It is used for retrieving parts of a single XML document by using a path notation, like those used in URLs. Every XPath expression evaluates to one of four basic types:

- Node-set (An unordered list of nodes)
- Boolean
- Number (floating-point number)
- String (a sequence of UCS characters)

An XPath location can be either a relative or an absolute location in an XML document. It can deal with seven node types:

- Root node
- Element nodes
- Attribute nodes
- Namespace nodes
- Processing instruction nodes
- Text nodes
- Comment nodes

The amount of nodes matched by an XPath location can be restricted further by specifying additional requirements for a match like comparison operators, functions or predefined variables. XPath supports equality operators and helper functions operating on the four basic types (i.e. node-set, Boolean, number and string), for instance substring extraction, summation of the values in a node-set or the number of nodes in a node-set to name a few. ***Table 2.1*** shows an example of some XPath expressions to retrieve data from the XML document in ***Figure 2.1***.

Table 2.1: Example of some XPath expressions

./author	All <author> elements within the current context. Note that this is equivalent to the expression in the next row.
author	All <author> elements within the current context.
/books	The document element (<books>) of this document.
//author	All <author> elements in the document.
book/ISBN	All <ISBN> elements that are children of a <book> element.
books//name	All <name> elements one or more levels deep in the <books> element (arbitrary descendants). Note that this is different from the expression in the next row.
books/*/name	All <name> elements that are grandchildren of <books> elements.
author[1]	The first <author> element in the current context node.
book/*	All elements that are the children of <book> elements.
book[@price &lt "60.0"]	All <book> elements where price attribute is less than "60.0".
ancestor::name[parent::book][1]	The nearest <name> ancestor in the current context and this <name> element is a child of a <book> element.

2.2.2 XML XQuery 1.0 Language:

XQuery (Boag et al., 2007) is an XML Query Language according to W3C Candidate Recommendation on 23rd January 2007. The mission of the XML Query project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web. Users can retrieve data from multiple XML documents using complex nested query expressions by XQuery. Therefore, it is providing eventually the needed interaction between the Web World and the database world.

XQuery is an extension of XPath version 2.0; it does not operate on the syntax of an XML document, but on its abstract, logical structure known as the XQuery 1.0 and XPath 2.0 data model. The XQuery language does not utilize XML Markup but has a syntactic grammar of its own.

The special feature of XQuery is that it has FLWOR expressions. FLWOR is a shortcut for FOR-LET-WHERE-ORDER BY-RETURN and it works similarly to SELECT-FROM-WHERE-ORDER BY statements in SQL. FLWOR expressions are used to combine and restructure XML data; it binds variables to values in “for” and “let”, clauses. Such binding of a variable to some value is called a tuple. The “for” clauses produce a stream of tuples. This tuple stream can be stored by a let clause into a variable. This variable can be used later by “where”, “order by” and “return” statements.

Table 2.2 shows some XQuery expressions that can be used to retrieve data from the XML document in ***Figure 2.1***. The first three expressions look like XPath expressions and the last one looks like an SQL statement. The last two expressions give the same results, but they are different in form. So, users can use any one of the two forms to retrieve their data.

Table 2.2: Example for some XQuery expressions

doc("books.xml")/books/book/ name	Select all the name elements in the "books.xml" file
doc("books.xml")/books/book[@price<30]	Select all the book elements under the books element that have a price attribute with a value that is less than 30
doc("books.xml")/books/book[@price>30]/name	Select all the name elements under the book elements that are under the books element that have a price attribute with a value that is higher than 30.
for $x in doc("books.xml")/books/book where $x/@price>30 order by $x/name return $x/name	Select exactly the same as the path expression above. Except names are sorted using order by clause.

2.3. Schema Languages for XML

XML Schema languages (i.e. DTDs, XML Schema (Fallside and Walmsley, 2004;Thompson et al., 2004), RELAX NG (Murata et al., 2001), DSD (Møller, 2005), Schematron (Jelliffe, 2006)) are used to validate XML documents. Validating a document is the process of verifying whether XML documents conform to a set of structural and content rules expressed in one of many schema languages; it works as firewall against invalid documents and allows skipping document validation in data processing applications because the parser will have already validated the document. Validation occurs on at least four levels: (Ray, 2003)

1. Structure: the use and placement of Markup elements and attributes.
2. Data typing: patterns of character data (e.g. numbers, dates, text).
3. Integrity: the status of links between nodes and resources.
4. Business rules: miscellaneous tests such as spelling checks, checksum results, and so on.

2.3.1 Document Type Definition (DTD) has been used for validating SGML structures (OASIS, 2002), and then it has become in use to provide validation for XML documents. It provides a regular expression language for imposing constraints on the content model (i.e. elements and subelements), but it is very limited in the control of attributes and data elements as it is not designed originally for XML data. ***Figure 2.2*** shows a DTD example, which can be used to validate the XML document in ***Figure 2.1***.

```
<!ELEMENT books (book*)
<!ELEMENT book (name, authors, ISBN)
<!ATTLIST book price CDATA #REQUIRED>
<!ATTLIST book id ID #REQUIRED>
<!ELEMENT name (#PCDATA) >
<!ELEMENT authors(author*)>
<!ELEMENT author(#PCDATA)>
<!ATTLIST author id ID #REQUIRED>
<!ATTLIST author address CDATA>
<!ELEMENT ISBN (#PCDATA)>
```

Figure 2.2: DTD example

2.3.2 XML Schema is a W3C recommendation aimed for replacing DTDs as the official schema language for XML documents (Fallside and Walmsley, 2004;Thompson et al., 2004). It provides a large number of improvements over DTDs. The first and most evident improvement is the switch to an XML-based syntax, which improves it in terms of flexibility and automatic process ability. Moreover XML Schema is completely namespace-aware. Another major contribution of XML Schema is the Post Schema Validation Infoset (PSVI), i.e., the additional information that the validation adds to the nodes of the XML document so that downstream applications can make use of it for their own purposes. The most important advantage of PSVI is certainly the type, or the set of legal values that a node can have. Types in XML Schema are either simple (strings with various constraints) or complex (Markup substructures of the XML document including elements, attributes and text nodes). A large number of built-in simple types are provided, ranging from integers to dates, times, and URIs. ***Figure 2.3*** shows an example for XML Schema, which can be used to validate the XML document in ***Figure 2.1***.

2.3.3 RELAX NG is a schema language for XML developed by an international working group, ISO/IEC JTC1/SC34/WG1 (Murata *et al.*, 2001). It is based on two preceding languages: Tree Regular Expressions for XML (TREX) (Clark, 2001), designed by James Clark, and Regular Language description for XML (RELAX) (Makoto, 2002), designed by Murata Makoto. Patterns are the central concept of RELAX NG. They widen the scope of the concept of content model, while in DTDs a content model is an expression over elements that are limited to text. In RELAX NG a pattern is an expression of elements, text nodes and attributes. External definitions of data types can be used for constraining the set of values of text nodes and attributes. ***Figure 2.4*** shows an example for RELAX NG Schema, which can be used to validate the XML document in ***Figure 2.1***.

2.3.4 Document Structure Description (DSD) is a schema language developed jointly by AT&T Labs and BRICS (Møller, 2005;Klarlund et al., 2000). Constraints are the central concept in DSD. A constraint is used to specify the content of an element, its attributes and its context (i.e. the sequence of nodes from the root to the element). An element definition is specified as a pair consisting of an element name and a constraint. The element content is constrained by a content expression, that is, a regular expression over element definitions. Context patterns are used to enforce constraints on the context of an element.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- definition of simple elements -->
<xs:element name="name" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
<xs:element name="ISBN" type="xs:string"/>
<!-- definition of attributes -->
<xs: attribute name="price" type="xs:decimal"/>
<xs: attribute name="id" type="xs: positiveInteger "/>
<xs:attribute name="address" type="xs:string"/>
<!-- definition of complex elements -->
<xs:element name="books">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="book"/>
</xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="book">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="name"/>
   <xs:element ref="authors"/>
   <xs:element ref="ISBN/>
  </xs:sequence>
<xs:attribute ref="price" use="required"/>
<xs:attribute ref="id" use="required"/>
 </xs:complexType>
</xs:element>
<xs:element name="authors">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="author" minOccurs="1"/>
  </xs:sequence>
<xs:attribute ref="id" use="required"/>
<xs:attribute ref="address" minOccurs="0"/>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 2.3: Shows an example for XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="books"
xmlns="http://relaxng.org/ns/structure/1.0">
   <element name="book"
   <attribute name="price"/>
   <attribute name="id"/>
   <element name="name"><text/></element>
   <element name="authors"
      <element name="author"><text/>
      <attribute name="id"/>
      <optional>
        <attribute name="address"/>
      </optional>
      </element>
   </element>
   <element name="ISBN"><text/></element>
   </element>
</element>
```

Figure 2.4: Shows an example for RELAX NG schema

2.3.5 Schematron is a rule-based schema language created by Rick Jelliffe at the Academia Sinica Computing Centre (ASCC) (Jelliffe, 2006). It is mainly used to check co-constraints in XML instance documents. A Schematron document defines a sequence of <rule>s, logically grouped in <pattern> elements. Each rule has a context attribute where XPath pattern determines the elements in the instance document to which the rule applies. Within a rule, a sequence of <report> and <assert> elements is specified having a test attribute which is an XPath expression evaluated to a Boolean value for each node in the context. The content of both <report> and <assert> is an assertion which is a declarative sentence in natural language. When the test of a <report> succeeds, its content becomes output.

2.4. XML API

The XML Application Program Interfaces (XML APIs) has been designed to allow a programmer in most programming languages, such as Java, C++, and Perl, to access their XML documents information without writing a parser in their Programming Language.

2.4.1 DOM Parser

DOM (Document Object Model) parser is used as a hierarchical object model to access the XML document information. It reads the entire document information and forms its corresponding DOM object tree of nodes in the main memory. This approach makes XML parser suitable for small XML document that can fit in the memory. DOM parser can be used for the documents in which the sequence of elements is very important (i.e. document centric documents) since it preserves the sequence of elements that it reads from the XML documents. It contains functions for traversing XML trees, inserting, deleting, and accessing nodes. ***Table 2.3*** shows some properties and methods used by DOM parser. (Hégaret et al., 2005;W3C, 2005)

2.4.2 SAX Parser

SAX (Simple Application Interface for XML) parser gives access to XML document information as a sequence of events, which makes it faster than DOM parser. It fires an event for every open tag, every closing tag, #PCDATA and CDATA section. The document handler will have to interpret these events and the sequence in which these events are fired. SAX can be used for large XML documents, since the documents do not need to be parsed in the main memory first. It can also be suitable for structured XML documents since elements order is not necessary. Another point of difference between SAX and DOM is worth mentioning here. SAX has a limitation in that no insertion of new contents can be done on the document, i.e., read only. DOM has the ability to do that through some methods and function for accessing, inserting and deleting nodes, i.e., read and write over XML document. ***Table 2.4*** shows main methods used by most XML SAX parsers. (www.Altova.com/XMLSpy, 2008)

Table 2.3: Some properties and methods used by DOM parser

Some XML DOM properties:	
• x.nodeName	- the name of x
• x.nodeValue	- the value of x
• x.parentNode	- the parent node of x
• x.childNodes	- the child nodes of x
• x.nextSibling	- the right sibling of node x
• x.attributes	- the attributes nodes of x
• x.previousSibling	- the left sibling of node x
Some XML DOM Methods:	
• x.getElementsByTagName(*name*)	- get all elements with a specified tag name
• x.appendChild(*node*)	- insert a child node to x
• x.removeChild(*node*)	- remove a child node from x
Where x is referring to a node object.	

Table 2.4: Some methods used by SAX parser

startDocument ()	Invoked when the Parser encounter document start
endDocument ()	Invoked when the Parser encounter document end
startElement (String name, AttributeList attrs)	Invoked when the Parser encounter element starting tag> The attributeList parameter has the list of all attributes declared for the current element in the XML File
endElement (String name)	Invoked when the Parser encounter element closing tag.
characters (char buf [], int offset, int len)	Invoked when the Parser encounter extra characters like space or enter character are encountered.
processingInstruction (String target, String data)	Invoked when the parser encounters a processing Instruction which is declared like

2.5. XML Documents Types

Using of XML technology in most web services such as e-business, e-commerce, e-banking, e-mail, e-library, e-government generates different types of XML data. These data can be classified according to their structure into: 1) Document centric documents, 2) Data centric documents, and 3) Mixed documents. (Bourret, 2005)

A comparison between XML document types are shown in ***Table 2.5***. Characterizing XML documents as data-centric or document centric will help in deciding the kind of database to use. As a general rule, data can be stored in a traditional database, such as a relational, object-oriented, or hierarchical database. This can be done by third-party middleware or by capacity built into the database itself. In the latter case, the database is said to be XML-enabled. Documents can be stored in a native XML database, (i.e. a database designed especially for storing XML), or a content management system (i.e. an application designed to manage documents and built on top of a native XML database).

These rules are not absolute. Data, especially semi-structured data, can be stored in native XML databases and documents can be stored in traditional databases where few XML-specific features are needed. Furthermore, the boundaries between traditional databases and native XML databases are beginning to fade away, as traditional databases add native XML capabilities and in turn native XML databases support the storage of document fragments in external databases, which are usually relational databases.

Table 2.5: Overview of XML documents types

	Document type	**Used for**	**Document characteristics**	**Order of sibling element**	**Document originality**	**Examples**
1.	Data-Centric	data transportation, machine consumption	fairly regular structure, fine-grained data	generally not significant, except when validating the document	database	Sales orders, flight schedules, scientific data
2.	Document-centric	data publishing, human consumption	less regular or irregular structure, larger grained data	significant	RTF, PDF, or SGML, Documents then converted to XML	Books, emails, advertisements, user's manual, and almost any hand-written XHTML documents.
3.	Mixed Document	A + B types, or B + A types	A + B types, or B + A types	insignificant part + significant part	database & other document types (A + B)	- Invoice, might contain large-grained, irregularly structured data, such as a part description. - Books, might contain fine-grained, regularly structured data, such as an author's name and a publication date

2.6. XML Data Storage Approaches

Since XML inception in 1998, a lot of research studies have looked for efficient storage and query medium for storing XML documents. Athena Vakali discussed existing options of XML storage which depends on the underlying framework's particular level showing their storage format, main advantages and main disadvantages (Vakali et al., 2005). ***Table 2.6*** summarises these options. The discussion shows that Relational Database Management System (RDBMS), Object Oriented Database Management System (OODBMS) and native XML database are the most accepted approaches.

2.6.1 RDBMS

Relational Database Management System (RDBMS) which has been proposed by Codd in 1970s is reliable, widespread and a well established medium for storing and retrieving data in the business area. Some approaches have been proposed to store XML documents into relational database and retrieve its content again from relational database (Fujimoto et al., 2005; Shanmugasundaram et al., 1999; Tan et al., 2005; Zhang and Tompa, 2004b; Xing et al., 2007b). Relational database has power capabilities in indexes, triggers, data integrity, security, multi-user access, query optimization by SQL query language, and crash recovery. The youth XML technology is looking for achieving some of these capabilities.

2.6.2 OODBMS

Object Oriented Database Management System (OODBMS) can deal with complex applications such as multimedia data and geographic information systems. However, there are some limitations: 1) OODBMS is language dependent often, (i.e. a specific API of specific language is used only to access the data), 2) it is schema dependent, (i.e. any modification to the schema or any class should be done to the classes interacting with instances of this class) This will involve the system in a wide recompile and will extend the time for updating the entire instance object within the database according to its size. But there are some works for storing XML documents in OODBS since both of XML and OODBS are hierarchical in their nature structure (Chung and Jesurajaiah, 2005).

2.6.3 XML database

A new set of languages are dedicated for XML documents which are the native XML database (Jagadish et al., 2003;Grinev et al., 2004). These languages which include XLink, XPath, XQuery and XSLT are designed for the particular purpose of storing and querying XML documents (Bray *et al.*, 2007). Also XML languages do not reach the power capabilities of existing relational database system; they do not allow users to query data in XML documents and other data in RDBMS simultaneous.

The above discussion has shown that RDBMS is the most suitable storage for XML data until now; in addition, it has a widespread implementation as a storage and retrieval medium in the business area. But there is a difference in the structure between the hierarchical ordered XML and tabular unordered RDB. This difference expresses the need for mapping techniques from XML documents to RDB in order to utilize their advantages and make the XML technology more acceptable by the RDB users.

Table 2.6: Overview of popular XML storage approaches (Vakali et al., 2005)

Framework	XML Storage Format	Main Advantages	Main Disadvantages
File-system-oriented	- ASCII files stored in the file system or database - management system (DBMS) as binary large objects (Blobs) or character large objects (Clobs)	- Easy implementation - Suitable for small XML sets	- Accessing and updating are difficult
Relational DBMS	- Tables	- Scalability and reliability - Easy implementation	- Requires many joins due to XML document factorization
Object-relational DBMS	- Tables and objects	- Easy implementation - Abstract data type support	- XML document factorization
Native XML	- Ad hoc data models or typical database models	- Flexibility - Improved access performance	- Less mature than conventional DBMSs (such as RDBMSs)
Directory servers	- Tree structure	- Optimized for queries - Effective data retrieval	- Low update performance

2.7. RDBMS Model

A database can be defined as a collection of related files. The relation between these files depends on the model used to describe these data, relational, hierarchical network or object-oriented model. Currently, RDBMS is the one used most often (Codd, 1970; Codd, 1971; Delobel, 1978; Codd, 1983). The relational model can be determined by some rules and facts such as:

1- Database is a collection of related tables (relations).

2- Each table consists of a set of records (tuples).

3- Each record consists of a fixed number of fields (attributes) which give descriptions for an object or a person.

4- Each field gives a specific characterization of data for the object, (i.e. single data type: name, age, or date). Relational model supports many data types including number, string, varchar, memo, date and Boolean.

5- One of those fields should uniquely identify the object; for example, student number in student table. This field is called the primary key.

6- The primary key in a table can be used as an additional field in other tables to create relations among them. This field is called a secondary key. So, the relations inside the database can be preserved using those primary and secondary keys.

7- The relation type between tables can be one-to-one relation or one-to-many relation which depends on the number of occurrence of the secondary key in one of them.

8- The relational model provides a set of relational operators' including selection, production, join, and cartesian product to process data in the database.

9- Database normalization helps to reduce data duplication and to increase data integrity.

10- Structured Query Language (SQL) offers a set of commands for accessing database through inserting, deleting and updating data.

2.8. The Similarities and Differences between XML Model and RDB Model

XML was originally proposed to represent, publish and exchange data between business applications on the Internet (Bray *et al.*, 2007) in 1998. RDB was proposed by Codd in the 1970s for storing and retrieving data (Codd, 1971;Codd, 1970). XML and its related technologies provide something found in database as XML documents for storing, DTDs and XML Schema for validating, XPath and XQuery for querying, and DOM and SAX for parsing XML documents. But, XML languages lack many things that are found in traditional databases such as indexes, triggers, data integrity, security, crash recovery, and multi-user access (Zhou et al., 2006).

XML can organize data in a hierarchical, object-oriented, and multidimensional way in the form of a tree with an arbitrary depth and width (Chen et al., 2006;Wang and Meng, 2005) as shown in ***Figure 2.5***. Meanwhile, a traditional relational database table can be thought of as a tree of depth two with unbounded fan-out at the first level, and fixed fan-out at the second level, with the first level representing tuples (rows) and the second level representing fields (columns). ***Figure 2.6*** and ***Figure 2.7*** show a sample of relational database representation (i.e., as tree and table respectively). An XML tree is clearly a more expressive way of representing data as no constraints are placed on either depth or width.

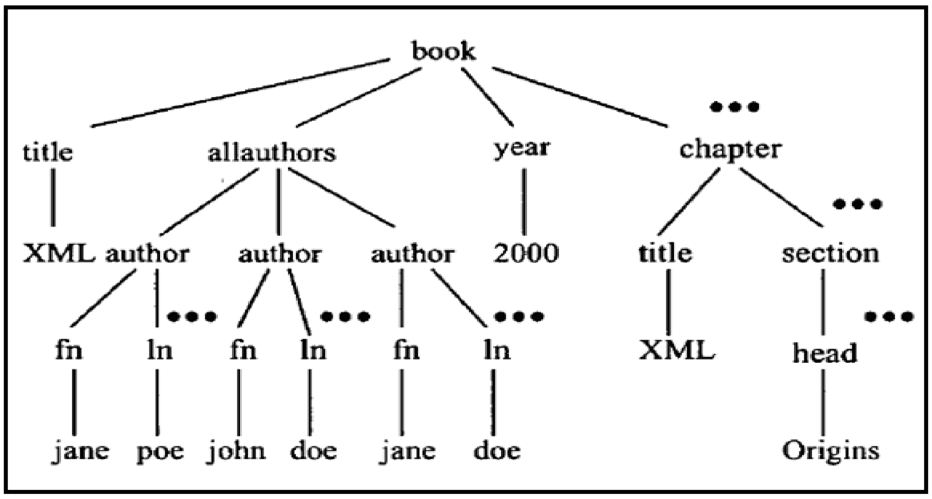


Figure 2.5: A sample of XML tree representation (Chen et al., 2006)

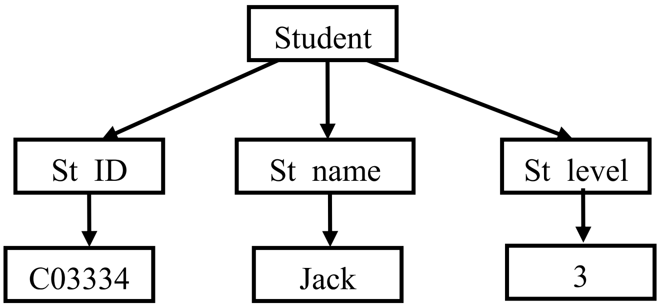


Figure 2.6: A sample of relational database tree representation

Students table

St_ID	St_name	St_level
C03334	Jack	3

Figure 2.7: A sample of relational database table representation.

A comparison between XML technology and RDB technology was given in (Bansal and Alam, 2001) as shown in ***Table 2.7***. The comparison in ***Table 2.7*** shows that there is a structural hole between hierarchical ordered XML and tabular unordered RDB. As a result, mapping between the XML and RDB is the best solution to exploit their advantages, and makes the XML technology more acceptable by the RDB users. For this reason, mapping

XML documents to RDB has been studied by many researchers, and relational database vendors (e.g., Oracle, DB2, and SQL Server).

Table 2.7: A comparison between XML and RDBMS (Bansal and Alam, 2001)

XML	RDBMS
Data in single hierarchical structure	Data in multiple tables
Nodes have element and/or attribute values	Cells have a single value
Elements can be nested	Atomic cell values
Elements are ordered	Row/column order not defined
Elements can be recursive	Little support for recursive elements
Schema optional	Schema required
Direct storage/retrieval of XML documents	Joins often necessary to retrieve data
Query with XML standards (XQuery, XPath)	Query with SQL
Human and machine readable	Machine readable

2.9. Summary

In this chapter, a review of the XML language and other supporting languages, XPath, XQuery, XSLT, and XML schema were given. This review shows that XML technology has received a lot of attention from researchers and database vendors to improve and to make this technology available to the market and user in a highly standard form. Also, it shows that this technology needs a lot of work to solve data processing problems such as multi-user access, security, crash recovery, concurrency control, data querying and retrieving, and data integrity, which have been already solved by database management and object oriented databases. These issues show the need to think of other storage options for storing and retrieving XML data. Reviews of these options were presented in this chapter and a comparison between them was made. Relational database is the mostly expected candidate for this choice since it solves most problems of data access issues. Some rules and facts about the relational database model were raised, and a comparison with XML model was introduced. The comparison shows the need for mapping techniques to map XML data to relational database to take advantages of their attributes since there is a gap between the two models. In chapter three, different mapping techniques for storing, rebuilding, and retrieving XML data from relational databases are discussed.

CHAPTER 3 STATE OF THE ART TECHNOLOGY

This chapter presents the state of the art approaches for storing and retrieving the XML documents from relational databases. Approaches are classified into schema-based mapping and schemaless-based mapping. It also discusses the solutions which are included in Database Management Systems such as SQL Server, Oracle and DB2. The discussion will address the issues of: rebuilding XML from RDBMS approaches, and comparison of mapping approaches: their advantages and disadvantages. The chapter concludes of the issues addressed.

3.1. Approaches for storing and querying XML

A number of different techniques for storing XML documents in a RDB have been established. These techniques can be divided into two groups: the schemaless-centric technique and the schema–centric technique (Dweib *et al.*, 2008). The first one makes use of XML document structure to manage the mapping process (Tatarinov et al., 2002;Dweib et al., 2008;Soltan and Rahgozar, 2006;Zhang and Tompa, 2004b;Jiang et al., 2002;Yoshikawa et al., 2001). The second one depends on schema information to develop a relational schema for XML documents (Fujimoto et al., 2005; Shanmugasundaram et al., 1999; Amer-Yahia et al., 2004; Atay et al., 2007b; Xing et al., 2007b; Knudsen et al., 2005; Lee et al., 2006).

The aim of mapping XML documents into relational database is to make use of the capabilities of the relational database which are: indexes, triggers, data integrity, security, multi-user access, and query optimization by SQL query language. In the meanwhile XML technology is trying to gain the above-mentioned capabilities, developed for RDBs, and efficiently store, retrieve, and rebuild XML data from RDBs.

The studies that address the problem of mapping XML document into RDB take care of the above issues, and attempt to translate users' XML queries, either XPath expression (Berglund *et al.*, 2007) or W3C's recommendation XQuery expression (Boag *et al.*, 2007), into SQL queries (Oracle, n. a.). XQuery gives power to the translation method since XQuery comprises XPath, and it is recommended by W3C, while XPath is not. The translation method should also consider its ability to rebuild, the stored XML document without losing information, and retrieve it in an acceptable time. Many studies have tried to address translation and restore constructing labelling methods. Labelling methods aim to reserve nodes order, parent-child and ancestor-descendant relationships, and document structure(Tatarinov et al., 2002;Chung and Jesurajaiah, 2005;Soltan and Rahgozar, 2006;Li and Moon, 2001;O'Neil et al., 2004;Wu et al., 2004;Kobayashi et al., 2005).

3.1.1 Schema-Based Mapping

One of the early studies in this area was conducted by (Shanmugasundaram *et al.*, 1999) from the University of Wisconsin-Madison. They proposed three mapping techniques: Basic, Shared, and Hybrid Inlining. These are proposed to map DTDs into relational schemas. Basic Inlining proposed building a separated table for each element in the DTD while in the Shared Inlining each element is represented in one table. The Hybrid Inlining technique inlines shares an element which is not repeated or recursively related. These techniques are different from one another in the degree of redundancy; they vary from being highly redundant in Basic Inlining, to containing no redundancy in Hybrid Inlining.

The above approach offers limited structures to represent the features of XML data, such as nested relationships, ordering of XML documents, and the DBMS schema representations. Querying these structures is usually complex since the end users are not familiar with them.

Mapping algorithms for XML DTDs to relational schemas were proposed by Atay *et al.* (2007b) from Wayne State University . They attempted to enhance the shared-inlining algorithm (Shanmugasundaram *et al.*, 1999), in away to overcome its incompleteness and eliminate redundancies caused by the shared elements. They claimed that the algorithm can deal with any DTDs including arbitrary cyclic DTDs, but shared-inlining algorithm deals merely with two mutually recursive elements. Dealing with cycles which involve more than two elements in a DTDs is not clear. ***Figure 3.1*** shows the three cases they considered in their inlining procedure. In case 1, a node *a* is connected to a node *b* by a normal edge, and *b* has no other incoming edges. In this case, node *b* is inlined into its parent node *a*, and the parent-child relationships are maintained between *b* and its children. In case 2, node *a* is connected to node *b* by a normal edge where *b* has other incoming edges (i.e. *b* is a shared node). In this case node *b* is not inlined into its parent node *a* since *b* has multiple parents. In case 3, node *a* is connected to a node *b* by a star edge, such that every node of *a* can contain multiple occurrences of *b*. In this case, the node *b* is not combined into its parent node *a* in order to avoid redundancy. ***Figure 3.2*** gives an example of the idea of the inlining procedure clear. ***Figures 3.2.A*** and ***3.2.C*** show the DTD graphs, where the inlining results are shown in ***Figures 3.2.B*** and ***3.2.D*** after applying the inlining algorithm. It could be noted from figures that nodes which are connected by, -edge or *-edge and,-edge must point to a shared node.

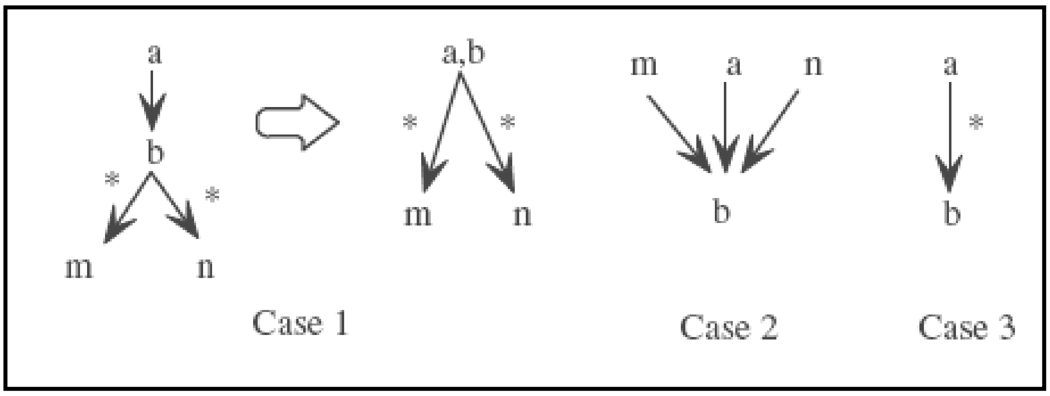


Figure 3.1: The three case of inlining (Atay et al., 2007b)

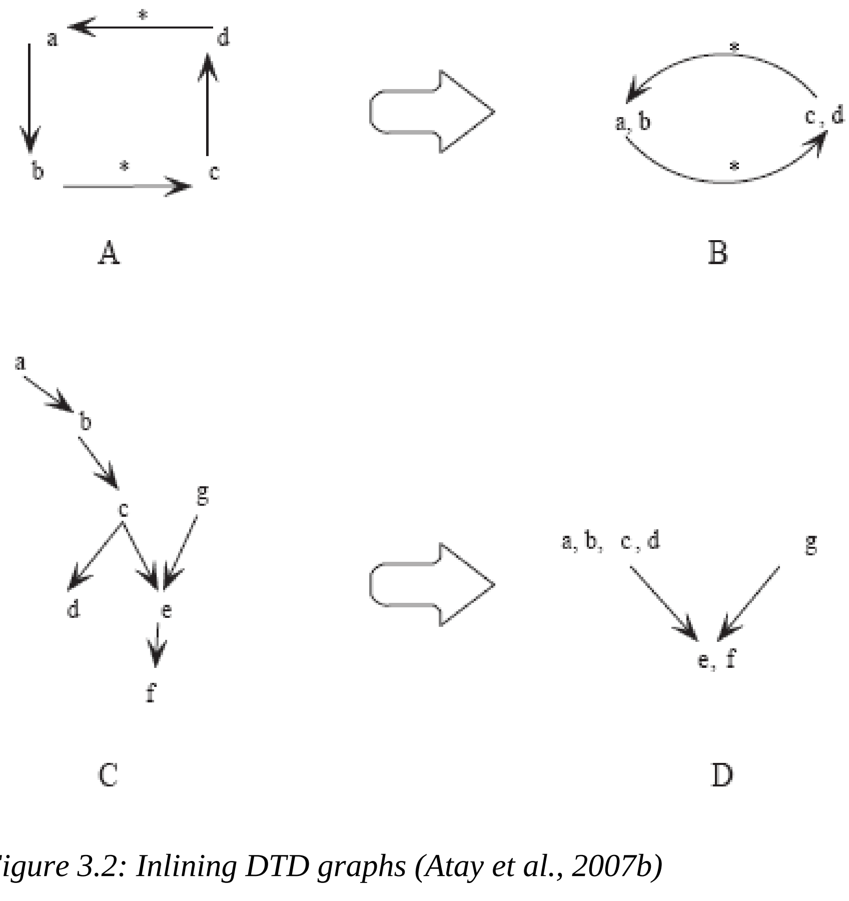
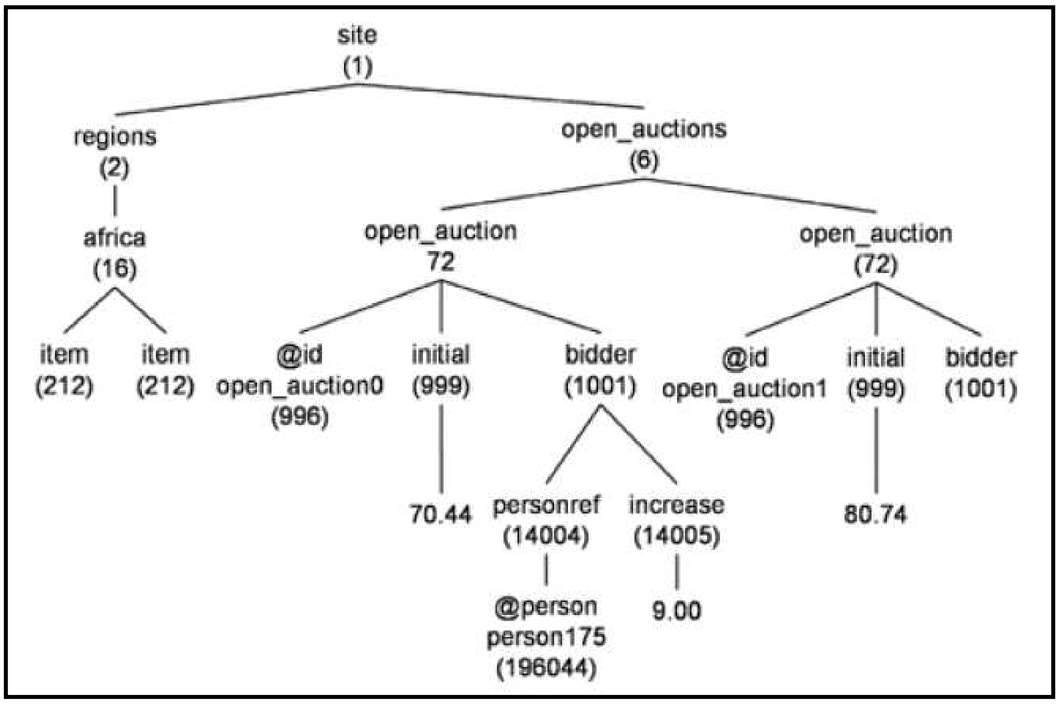


Figure 3.2: Inlining DTD graphs (Atay et al., 2007b)

Redundancy reduction XML storage in relations (RRXS) within XML Functional Dependency (XFD) was proposed by (Chen *et al.*, 2003). They defined constraints to capture the structural constraints as well as semantic information. It makes use of XML schema semantic constraints. Using the semantics of a document could reduce the redundancy since node identifiers can be removed where value based keys are still available for particular elements. Unfortunately the suggested rewrite rules are not complete. So, this algorithm cannot guarantee redundancy reduction.

SPIDER (Schema-based Path IDentifiER) is an approach for a node labelling scheme identified by Fujimoto *et al.* (2005), from Nagoya University and the Nara Institute of Science and Technology. They aimed to preserve XML tree structure. The approach used document's DTD information to give unique numbers for all paths from the root node. It assigns unique integers to each sequence of elements and attributes from the root node to any node in the XML tree. Since SPIDER could not distinguish between multiple nodes appearing in the same path, *Fujimoto et al.* introduced Sibling Dewey Order to identify such nodes. Consequently, several nodes are to be relabelled in order to insert a new node into an XML document, and to maintain nodes order. Only Sibling Dewey Order is relabelled but SPIDER is not affected. ***Figure 3.3*** and ***Figure 3.4*** show the difference between SPIDER labels and SPIDER and Sibling Dewey Order labels for XML tree. And ***Figure 3.5*** shows the relational schema used by SPIDER. Four relational tables are used each of which handles a different type of information; one for elements, one for attributes, one for texts and the last for paths of the document.

SPIDER uses string matching to handle the path that contains ancestor relation "//". This matching requires joining "element" and "path" relations, causes degradation of the approach performance. Moreover, this method cannot exactly preserve node order in some cases such as in the case of multiple components in the DTD declaration which have the same name but appear in different places. On the other hand, node indexing involves large extra space relative to the size of the original data. And indexing a document with a large number of nodes is very difficult. As a consequence, this method needs extra time overhead that is consumed to rebuild the original XML document.



.

Figure 3.3 : Node labelling using SPIDER (Fujimoto et al., 2005)

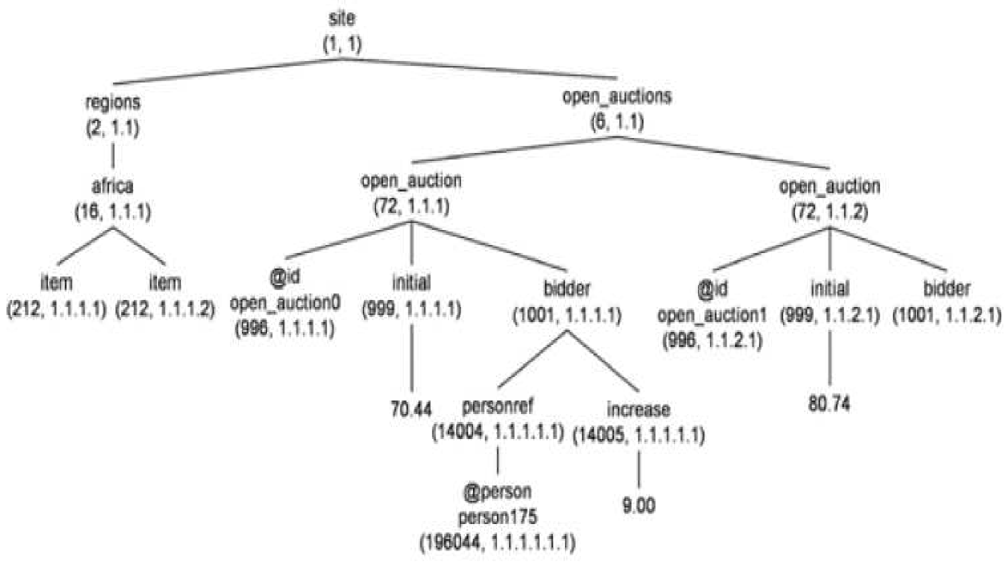


Figure 3.4 : Node labelling using SPIDER and Sibling Dewey Order (Fujimoto et al., 2005)

Element (docID, nodeID, spider, sibling, parentID)
Attribute (docID, nodeID, spider, sibling, parentID, value)
Text (docID, nodeID, spider, sibling, parentID, value)
Path (spider, path, pathexp)

Figure 3.5: SPIDER relational schema

Space reduction is needed to store XML documents which is a requirement to improve the performance of querying data. To reduce the space that is used to store the labels, and to make rebuilding of original XML documents easier, methods for indexing a group of XML nodes have been proposed by (Xing *et al.*, 2007a). These methods include: using path information to refine the storage, indexing a group of XML nodes instead of an individual node, and query evaluation based on the "nodes of interest".

Introducing nodes of interest can reduce the number of path joins required to process the query. ***Figure 3.6*** shows the way of grouping nodes in the XML tree. Each group of nodes is stored in one or at most two tables which can be linked together under the*"label"* field.

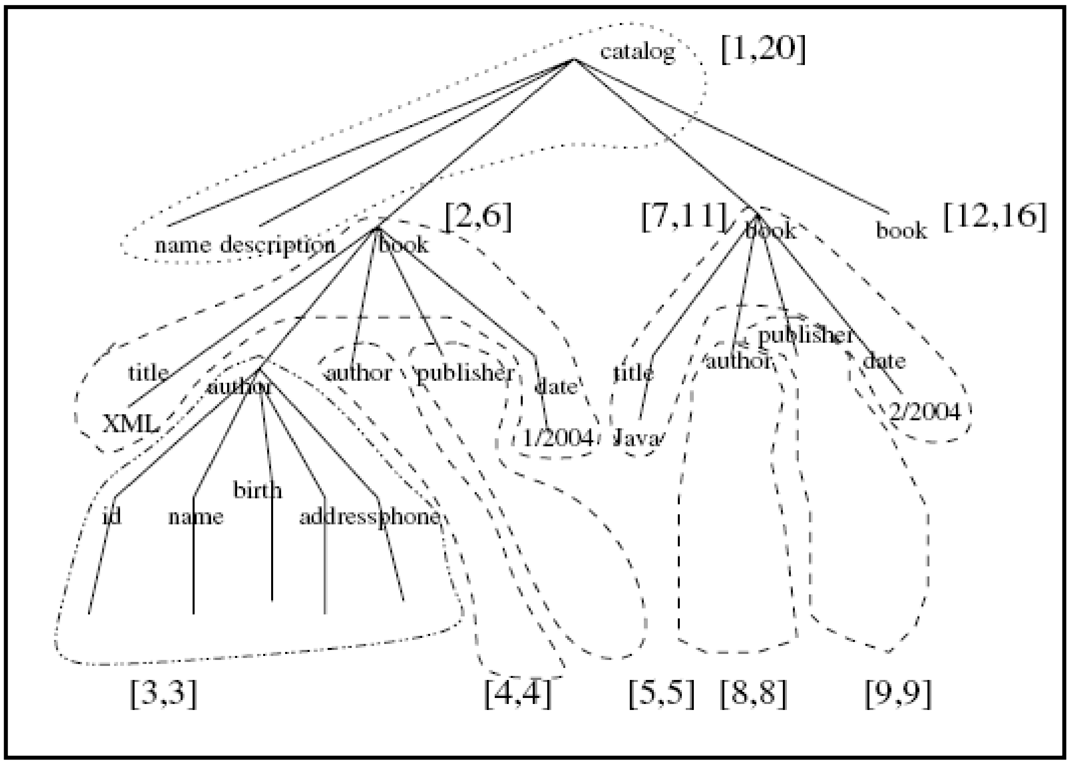


Figure 3.6: Grouped nodes

Amer-Yahia *et al.* (2004) at the AT & T Labs, proposed ShreX a mapping framework which stores the XML document in a RDBMS. XML schema was used to simplify the mapping process in ShreX by using a generic shredding process, which also translated XQuery into SQL. An extension of Shrex Mapping, called XShreX was proposed by Lee *et al.* (2006), from the National University of Singapore. Thus, XShreX mapped more constraints. They also developed semantic keys to replace the auto-generated keys of the ShreX in order to reduce redundancy and to decrease the size of the generated database.

3.1.2 Schema-Less Mapping

One of the issues of mapping XML to RDB is the loss of information due to the XML documents' shredding and inlining into RDB tables (Shanmugasundaram *et al.*, 1999). A dynamic shredding was proposed by Zhang and Tompa (2004b) in order to preserve the original XML document information and to solve the problem of the document size limitation. Documents are shredded to meaningful fragments according to users' judgement. These fragmentations are stored depending on relational schema. XML queries in XQuery are also needed to be translated by the users into SQL statements to retrieve shredded documents. The main idea of this approach is to keep the original document untouched; so, there will be no need to rebuild it. But that will make it impossible to connect with the data which already exists in the relational database since the XML document will not be saved in the relational database. In addition, there will be a need to translate each XQuery with a support of appropriate structured text operators.

XRel (Yoshikawa *et al.*, 2001) and XParent approaches (Jiang *et al.*, 2002) are used to store XML documents in RDB. Both approaches are path-based approaches and use predefined fixed relational schema for storing the XML tree information. The relational schemas that are used in both approaches are shown in ***Figure 3.7*** and ***Figure 3.8*** respectively. In XRel, elements, attributes and text are stored in different tables (i.e. element, text and attributes tables). The region (i.e. starts and end positions) of each node of element, attribute and text along with its ordinal and pathID are stored in the tables. The fourth table is used as a path table for document paths where the path is the sequence of elements from the root to the candidate element.

In XParent, element table stores each element in the document, and data table stores attributes and text values. LabelPath table stores all paths in the document and the length of each path. DataPath table stores all parent-child relations.

Path (PathID, PathExp)
Element (DocId, PathID, start, End, Index, Reindex)
Text (DocID, PathID, Start, End, Value)
Attribute (DocID, PathID, start, End, Value)

Figure 3.7: XRel relational schema

XRel and XParnet make a path expression to be easily evaluated by comparing path IDs. But allocating one code for each element in both approaches result in larger storage for large XML documents, and larger number of path joins to process a query.

LabelPath (ID, Len, Path)
DataPath(PID, CID)
Element (PathID, DID, Ordinal)
Data (PathID, DID, Ordinal, Value)

Figure 3.8: XParent relational schema

Tatarinov *et al.* (2002) proposed Global, Local and Dewey for labelling XML tree. In Global label each node is assigned a number that represents the node's absolute position in the document as in

Figure ***3.9***. In this label, dynamic update is very difficult since all the nodes placed after the inserted node need to be relabelled. And extracting the parent-child and ancestor-descendant relationship is also impossible.

In the Local Labelling, each node is assigned a number that represents its relative position among its siblings, as in ***Figure 3.10***. In this label, a combination of node's position and that of its ancestors forms a path vector that identifies the absolute position of the node within the document. Updating the Local label has led to better performance than in the Global label because only the following siblings of the new node need to be renumbered. But it is still hard to extract the parent-child and ancestor-descendant relationships.

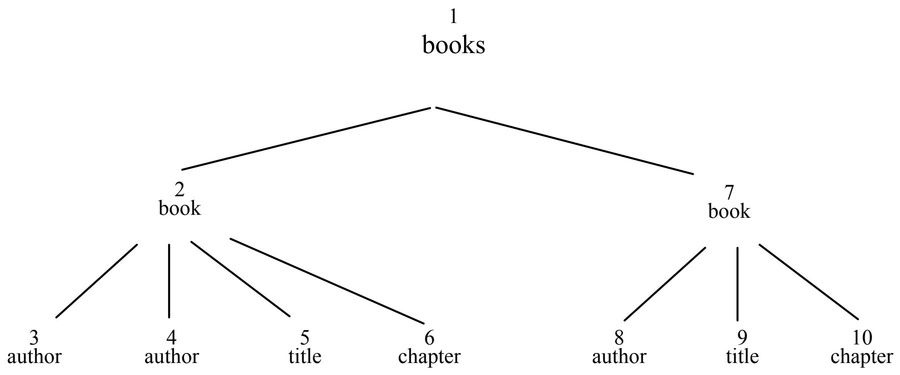


Figure 3.9: Global labels for XML Tree

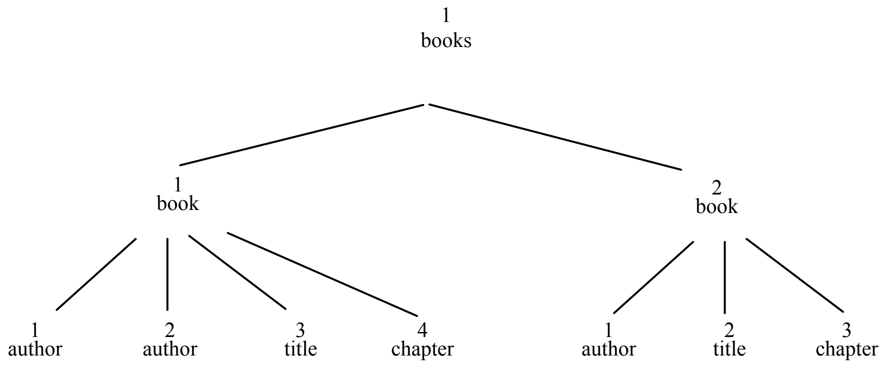


Figure 3.10: Local labels for XML tree

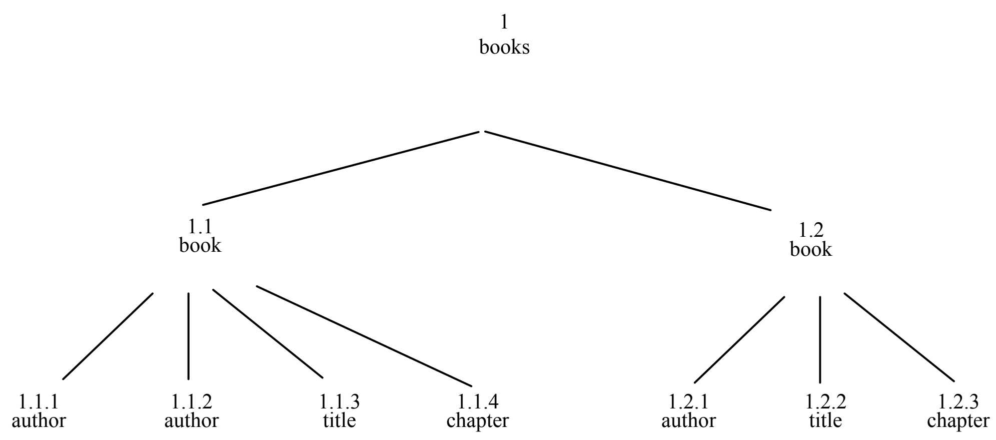


Figure 3.11: Dewey labels for XML tree

In the Dewey label, a node label is generated by combing its parent label and private integer number. ***Figure 3.11*** shows an example of labelling using Dewey labels. Extracting node label from its ancestors is very easy. But a large sized RDB could be generated in this case because a private label is given for each node, and an update of the following nodes labels is needed when new node is inserted.

ORDPATH, a hierarchical labelling schema implemented in Microsoft SQL Server 2005, was introduced by O'Neil *et al.* (2004). Nodes labelling of XML tree in this approach does not need an XML schema. It used two tables to store XML data. ***Figure 3.12*** shows ORDPATH relational schema.

Node (OrdPathCode, Tag, NodeType, Value, PathID)
Path (PathID, PathExp)

Figure 3.12: ORDPATH relational schema (O'Neil et al., 2004)

Contrast to the Dewey Labelling method, ORDPATH makes it possible to insert new nodes in uninformed locations in the XML tree without the need to update old nodes labels. This is because only positive odd integers are assigned to the nodes for the first scan, and even-number and negative integers are reserved for future insertions in the existing tree. Labels are assigned during initial loading. ***Figure 3.13*** shows ORDPATH labelling for an XML document. ORDPATH labelling update is efficient and it can maintain XML document structure. But it fails to perform semantic search or path search.

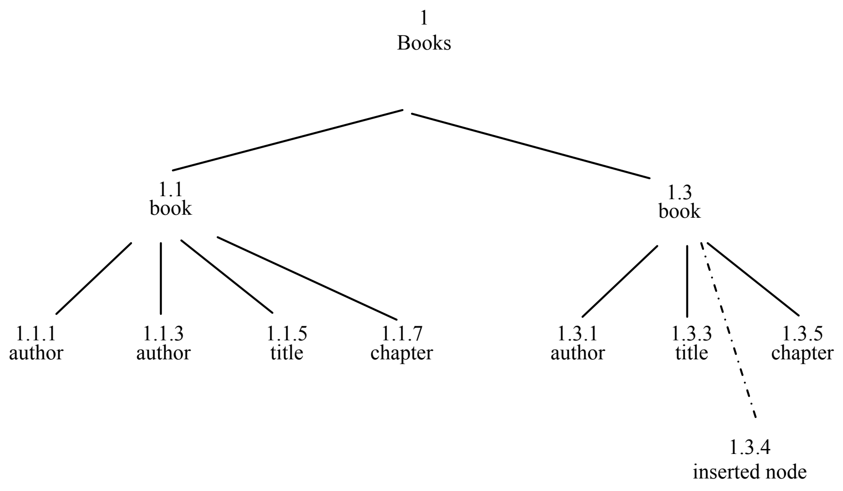


Figure 3.13: ORDPATH labels for XML tree

Pre-order and post-order traversing of tree structure is presented by Torsten, (2002). The method is designed to maintain nodes' ordering within the document, and identifies parent-child and ancestor-descendant relationships. The idea of this method can be described as follows:

1- Area A: where (post-order > Node(post-order)) and (pre-order < Node(pre-order)), consider nodes as ancestors to candidate node, which are identify by the path from the root to this node.

2- Area B: where (post-order > Node(post-order)) and (pre-order >Node(pre-order)), consider nodes as following the candidate node.

3- Area C: where (post-order < Node(post-order)) and (pre-order >Node(pre-order)), consider nodes as descendant of the candidate node, i.e. they are forming a subtree rooted by the candidate node. Subtrees are used to form the nested subtree that fragments the XML document.

4- Area D: where (post-order < Node (post-order)) and (pre-order <Node (pre-order)), consider nodes as preceding the candidate node.

Pre-order and post-order method optimizes the XML query by minimizing the area of search.

Figure 3.14 and ***Figure 3.15*** highlight how the pre-order and post-order method could minimize the area of search in the XML document. But this method encounters high cost of inserting a new node or new subtree since all nodes of pre-order label following the inserted node are to be relabelled and all nodes of post-order label for the following nodes and ancestors nodes for inserted node are to be relabelled.

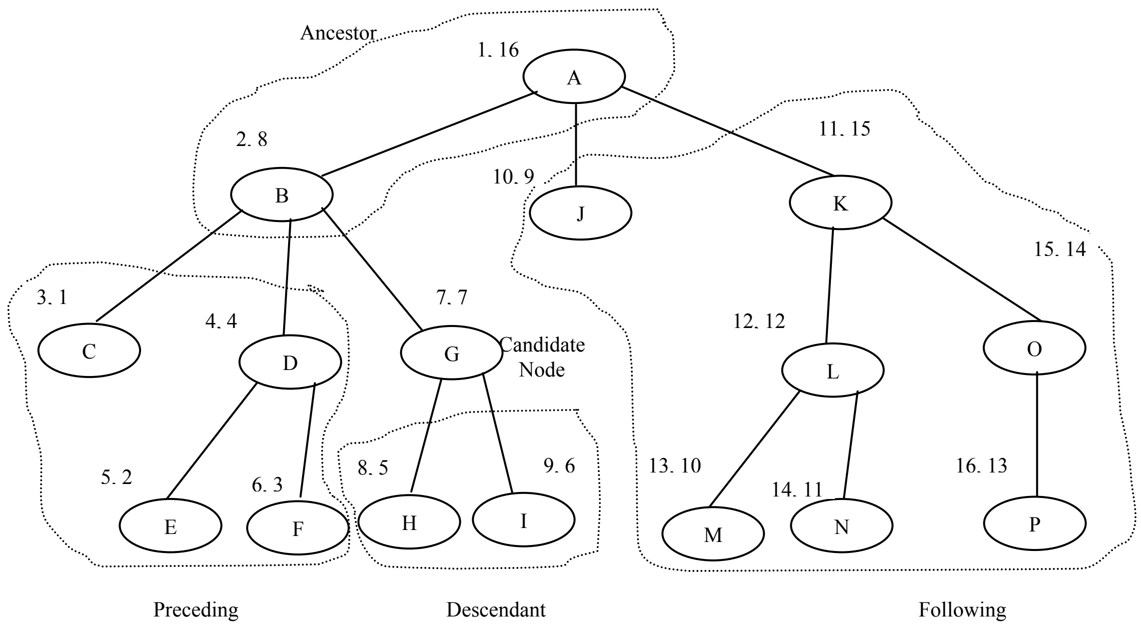


Figure 3.14: Tree representation for XML document with pre-order post-order labelling

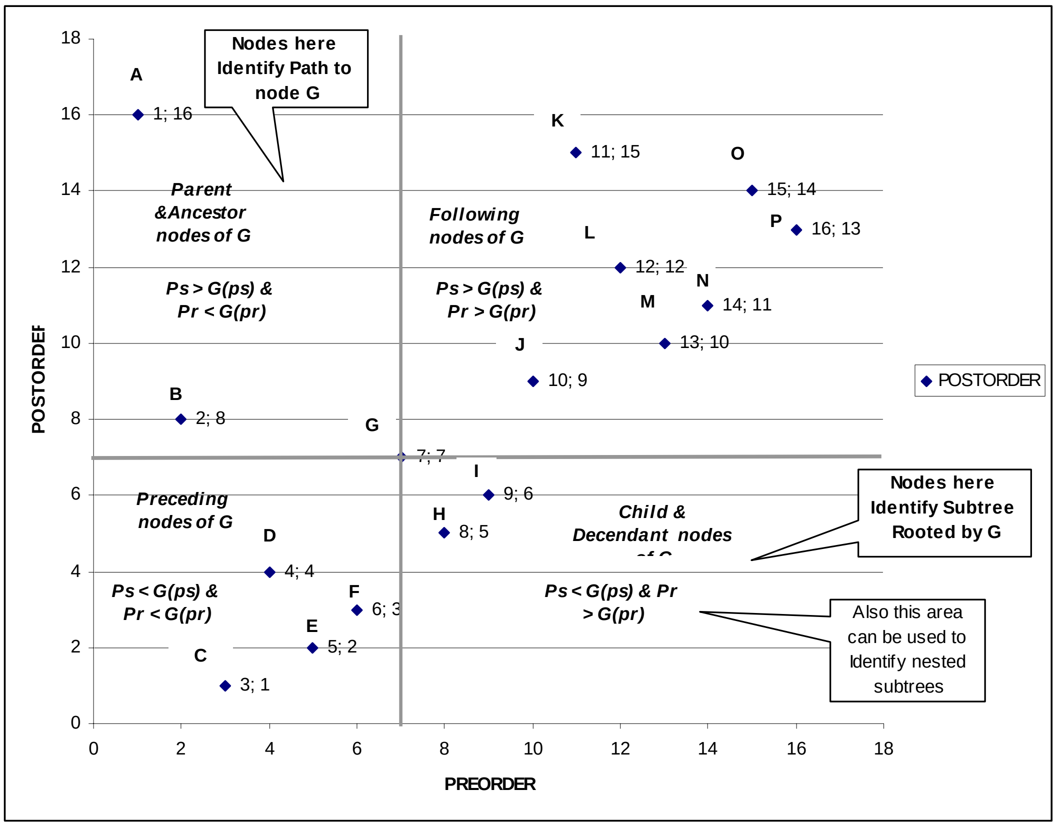


Figure 3.15: Pre-order post-order label optimization areas

A clustering-based scheme for labelling XML trees was proposed by Soltan and Rahgozar (2006). It uses a label for a group of elements not for each single element, and classifies elements into different groups in which each group is assigned for all sibling elements. And this group of elements are stored in a single relational record. ***Figure 3.16*** and ***Figure 3.17*** show clustered labelling method for an XML tree and its relational schema respectively. In this way, the database size needed for the mapping process is reduced because relational records numbers are less than those of using single record for each node. It also reduces the number of path joins needed to process the query, and makes the rebuilding of XML document from RDMB faster. But it experiences a problem of dynamic update; i.e. many nodes should be relabelled when a new node is inserted. But it fails in performing path and semantic search.

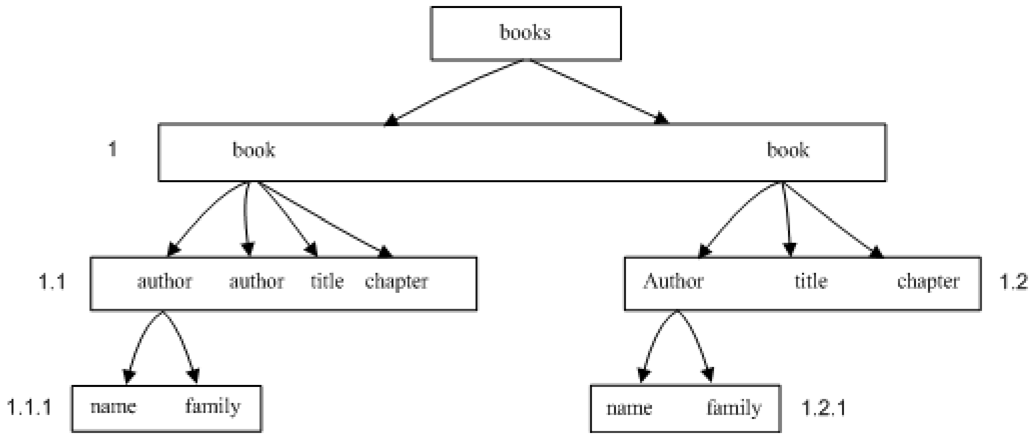


Figure 3.16: Clustered labels for XML Tree (Soltan and Rahgozar, 2006)

Node (ClusteredCode, Tag, NodeType, Value, PathID)
Path (PathID, PathExp)

Figure 3.17: Clustered relational schema (Soltan and Rahgozar, 2006)

XTRON Min *et al.* (2008) is a schemaless system to manage XML data as relational database. It merges the edge and the region approaches to manage parent-child and ancestor-descendant relationships. The edge approach is used to manage parent-child relationship, and the region approach is used to manage ancestor-descendant relationship. An extra space is used to maintain renumbering at each new node insertion. If the XML schema is not available, then document structural information is extracted. The system needs six tables to represent the merged numbering approach. The path information is transformed into intervals to speed up the query performance. But enhancing query performance increases size of the relational database. And there will be a very high cost of renumbering a larger number of relational fields.

3.2. Commercial DBMS XML Solutions

3.2.1 IBM DB2 Extender: using the XML Extender Document Access Definition (DAD) as XML Schema for mapping XML document into RDB. DADs can be used for storing XML document into RDB and for publishing RDB as XML. It provides two functions:

- dxxShredXML() function is used to decompose an XML document and store it in relational database, and
- dxxGenXML() function is used to build a shredded XML from relational database.

IBM DB2 provided some procedures for handling XML columns:

- XMLVarCharFromFile() is used for type conversion.
- Varchar(XMLVarChar) is used for retrieval.
- Update(xmlobj, path,value) is used for update.
- ExtractVarChar() is used as selection function.

In IBM DB2, XML columns can be assigned a type of:

- XMLCLOB is used for large documents;
- XMLFile is used for documents stored outside DB2.
- XMLVARCHAR is used for small documents

XML Extender also provides an XML DTD repository. Each XML database contains a DTD reference table called DTD REF which is used to store Meta information on users' mappings. The user can access this table to insert their own DTDs. These DTDs can be used to validate XML documents. Given the mapping, the system reads an arbitrary XML document and loads it into a DB2 database. IBM DB2 is using CLOBs (Character Large OBjects) and some extra tables for indexing structured data contained in the text for mixed content XML documents. These extra tables are updated automatically when new documents are added.

3.2.2 Oracle: XML was first supported in Orcale8i. This support was limited for publishing relational data in XML format. In Oracle9i Database Release 1 XDK, a number of tools are added for storing XML into relational database and generating XML from relational database. These tools include: XML Parsers, XSLT Processor, XML Schema Processor and XML SQL Utility to generate XML documents, DTDs and schemas from SQL queries. New data types for supporting XML storage were added to the kernel, which are XMLType and URI-Ref types. Several operators are linked to XMLType to facilitate processing XML data such as extract(), getNumberVal(), getStringVal() and existsNode().

Oracle XML DB was introduced in Oracle9i Database Release 2 (Oracle 9iR2). XML DB offers two options for mapping XML Schema either created automatically or by the user. Then, XML DB loads the schema file, stores mapping information internally and creates SQL types and tables' indexes.

Oracle 10g gave two solutions through Oracle XMLDB (DB, n.a). In the first solution, XML document is stored as CLOB in a single special type field (XMLType), or shredding the content of an XML document in a set of rows. The second gives an option for XML document shredding, either automated or controlled by the user, depending on the XML schema. SQL standards have been developed such as to be compatible with XML features. Database connectivity for SQL, XPath, XQuery and ODBC are provided. But XML schema is required before transmission to relational schema for shredding options. Oracle solutions are adapted only to Oracle systems which is expensive and not available for other DBMS.

3.2.3 Microsoft SQL Server:

To publish relational data as XML documents, Microsoft SQL Server uses the FOR XML clause as extension to the SQL. It uses three publishing modes: RAW, AUTO and EXPLICIT:

- RAW creates flat XML documents by converting each SQL result row into an XML element and each non-NULL column value to an attribute.
- AUTO mode uses query results to build nested documents where each table in the FROM clause is represented as an XML element. The columns listed in the SELECT clause are mapped onto attributes or sub-elements.
- EXPLICIT mode defines an SQL view to gather related rows. Special column names such as Tag and Parent are used. Nesting is explicitly specified as part of the query.

Microsoft implements three solutions for storing XML documents:

- The generic Edge technique.
- Users' annotation of an XML schema in order to determine the XML-to-relations mapping.
- OpenXML that compiles an XML documents into an internal DOM representation using *sp_xml_preparedocument* procedure.

These solutions are created using the XML Schema Definition (XSD), and are used to create the mapping schema that could be used for validating the XML document that is loaded in the relational database.

SQL Server 2005 adds a new XML data type to the relational table by using Transact SQL (T-SQL) or SQL Server Management Studio (Pal et al., December 2005). Adding a new XML data type incorporates a definition of the following options:

1. The type of the XML field: Either typed (specify a Schema collection) or un-typed (well-formed XML).

2. Document storage: Either stores the complete documents or fragment of it.

3. Schema: To store XML document depending on either a single or multiple schemas.

Microsoft provides storing XML documents as CLOBs. But, unlike IBM DB2 Extender, no extra tables are provided for indexing mixed content data.

In SQL Server, the relational database schema is constructed from XSD, which makes it difficult to query the XML data from other resources. SQL Server XML side can not be applied to other DBMSs such as DB2 or Oracle.

Consequently, each database vendor has to carry out special research for the development of XML support. Solutions are dedicated to the vendor's products and can not be used in other products. Therefore, many research efforts are needed to leverage and utilize relational database and XML technologies and their advantages.

3.3. Rebuilding XML from RDB

Storing XML documents into relational databases makes use of relational database management systems facilities, (i.e. multi-user access, data integrity, security, crash recovery) and makes use of its high potential query language SQL. Using original XML document after the mapping process will be out of use if any updating is done on the document. This makes rebuilding of XML documents from relational databases is equally important as a big deal. The rebuilding process raises a lot of issues to be considered, such as: 1) Reserving the structure of the original document, including nodes order and relationships when efficient labelling methods are used for rebuilding (i.e., parent-child, ancestor-descendant and preceding-following relationships), 2) Making sure that all document contents are stored (elements, attributes, comments … etc), 3) The rebuilding process should be efficient for the entire document or some parts of it.

These rebuilding solutions depend on the method used for mapping, and the way of labelling the contents of XML document in relational database.

3.4. Comparison of Mapping Approaches

Table 3.1 and ***Table 3.2*** show a comparison between some procedures of mapping of XML documents into relational database. The bases that are used for comparison are: schema-less or schema-based, number of tables used in relational schema, recursive consideration, and the query language (XPath or XQuery) used for retrieving the data. Mapping could be classified also according to the method used for labelling XML documents because the efficiency of the labelling method affects the performance of querying and updating documents' contents. ***Table 3.2*** reviews some methods presented in the literature for labelling XML document contents including elements and elements attributes.

Table 3.1: A summary of XML to RDB related works

Technique	**Schema/ Schemaless**	**No. of Tables**	**Cost-based**	**Preserve Order**	**preserve Constraints**	**Recursive consideration**	**XML query XPath/XQuery**
(Shanmugasundaram *et al.*, 1999)	Schema	> 2	yes	no	yes	no	XPath
XRel (Yoshikawa *et al.*, 2001)	Schemaless	4	no	Yes	No	no	XPath
Dewey (Tatarinov *et al.*, 2002)	Schemaless	4	no	Yes	No	no	XPath
XParent (Jiang *et al.*, 2002)	Schemaless	4	no	Yes	Yes	no	N/A
(Zhang and Tompa, 2004b)	Schemaless	> 2	no	yes	yes	no	XQuery
ORDPATH (O'Neil *et al.*, 2004)	Schemaless	2	no	Yes	Yes	No	XPath
ShreX (Amer-Yahia *et al.*, 2004)	Schema	> 2	No	Yes	No	no	Partial XPath
RELAXML (Knudsen *et al.*, 2005)	Schema	> 2	yes	yes	no	no	N/A
SPIDER (Fujimoto *et al.*, 2005)	Schema	4	yes	Yes	yes	no	XPath
(Atay *et al.*, 2007b)	Schema	> 2	yes	yes	yes	yes	N/A
LegoDB & FleXMap (Ramanath, 2006)	Schema	> 2	yes	No	No	yes	XPath
XShreX (Lee *et al.*, 2006)	Schema	> 2	yes	Yes	Yes	yes	XPath
(Soltan and Rahgozar, 2006)	Schemaless	2	no	Yes	Yes	No	N/A
Oracle interMedia Text, 2006	Schemaless /Schema	1	no	Yes	yes	-	XPath, XQuery
DB2 Text Extender, 2006	Schemaless /Schema	1	no	Yes	No	-	N/A
XTRON (Min *et al.*, 2008)	Schemaless	6	no	Yes	Yes	No	Partial XQuery

Table 3.2: A summary of XML labelling methods

Technique	Name	Advantages	disadvantages
(Li and Moon, 2001)	Interval encoding based on the number of words	Partially solves dynamic update problem	Relabelling of many nodes is needed in case of inserted data size exceeding reserved space
(Tatarinov *et al.*, 2002)	Global order label	It can help in answering XPath queries such as following and following-sibling.	All nodes of higher label than inserted node must be relabelled. It is difficult to answer ancestor-descendant relationship
(Tatarinov *et al.*, 2002)	Local order label	Only the following siblings of the inserted node need to be relabelled.	Just Sibling nodes following inserted node must be relabelled. Maintain parent-child relation is not easy.
(Tatarinov *et al.*, 2002)	Dewey order label	It is easy to maintain parent-child and ancestor-descendant relation	Sibling nodes right to the inserted node and their descendant must be relabelled
(Torsten, 2002)	Pre-order post-order	It minimizes the searching area within the document to accelerate XPath location step	All following nodes are needed to be relabelled after an insertion of new node. So, an insertion cost depends on the location where the new node is inserted.
(O'Neil *et al.*, 2004)	ORDPATH	It provides an ability for nodes insertion without a cost to relabel any existing node. Also it reserved parent-child relation	Many nodes need to be relabelled after the reserved space is used up. It fails in performing path and semantic search
(Wu *et al.*, 2004)	Prime number labelling	It is easy to identify ancestor-descendant relationship as it depends on whether their labels are divisible or not. Also insertion of new node and giving it prime number is easy.	- Large space size since candidate node label is self-label product from the root to node. - To reflect document order, they use simultaneous congruence value based on Chinese Reminder Theorem. And these value need to be re-calculated is considered time consuming. - Insertion between parent and child nodes is not supported.

Technique	Name	Advantages	disadvantages
(Kobayashi *et al.*, 2005)	Variable Length Endless Insertable (VLEI Code)	Parent-child and ancestor-descendant relationship are reserved. It reduces insertion cost since relabelling it not needed. Using octal number with "9" delimiter reduces the space needed for labelling.	Using octal and "9" delimiter instead of "." As character reduces the space but increases the time for relabelling since it as Dewey without space
(Soltan and Rahgozar, 2006)	Cluster based order	It is easy to maintain parent-child and ancestor-descendant relation. Also it decreases the # of records in the table	All sibling cluster right to the inserted cluster and their descendant must be relabelled
(Chung and Jesurajaiah, 2005)	Dynamic interval-based labelling	Parent-child and ancestor-descendant relationship are reserved. It solves partial insertion and updating issues.	Still some nodes need to be relabelled if no space available at the position of insertion. Also, extra space is needed for identifying each element. The querying process becomes high when the label is too long

3.4.1 Advantages and Disadvantages of Previous Approaches

Schema-less centric techniques reviewed above do not require an XML DTD or XML Schema. Present proposals depend on XML document's structure to manage the mapping process. In such approaches, XML document is entirely stored as a large solid object data type (CLOBs, BLOBs[1] for example). Another way is to map the tree or graph of the XML document generically onto predefined relations. These approaches depend on using a long-character-string data type, such as CLOB in SQL, to store XML documents or fragments as texts in columns of tables. The advantages of these approaches are: (1) They could provide textual fidelity since they preserve the original XML at the character string level, and (2) there is no need for an XML schema in the storing process. The drawbacks of these methods are: (1) They can not make use of the XML Markup structural information, (2) they don't take into account the query workload while constructing the relational schema, (3) the XML document structure is not preserved, and (4) it is difficult to deal with huge XML documents.

Schema centric techniques need XML schema to develop the relational schema. Such techniques need to create a relational schema to store the XML schema. The created schema is used during and after shredding the XML documents. The data that is captured from the XML document is stored in the created relational tables. The advantages of these techniques are: (1) They restrict XML structure to the defined schema (i.e. assign and use of Markup elements and attributes according to the defined schema), (2) they enforce referential constraints, primary and foreign key relationships, and (3) they simplify the mapping process because users are not involved in addressing a new mapping language. But, the techniques reviewed above are (1) all heuristic; (2) do not consider multiple possible relational mappings so

[1] Are data types provided by most relational database vendors (e.g., Oracle interMedia Text, DB2 Text Extender)

as to choose the optimal one; (3) moreover, fixed shredding of XML documents will lead to a loss of information from the original one, (Atay *et al.*, 2007b is an exception), (4) XML schemas are sometimes not available, so there is a need to construct the schema first and then do the mapping. 5) A reconstruction of database schema is needed as any change in the XML schema happens, which makes it very expensive in this case. 6) Sometimes, a large number of relations need to be created depending on the XML schema; consequently, a lot of joins are needed to retrieve XML document information.

3.5. Summary

In this chapter, a review and discussion of related methods and techniques for mapping XML documents into relational database have been presented. Maintaining document structure and reserving nodes' order within XML documents are too important as in document-centric documents (i.e. books, emails). Nodes labelling is another issue in mapping XML document into relational database, since relational database structure is an unordered tabular form, and XML document has a hierarchically ordered structure by nature. Some labelling methods for XML documents contents have also been discussed in this chapter.

The discussion shows that most of the labelling methods are concerned with the increase of query performance, but they ignore or fail to achieve efficient updating of XML document. The reason for that fail is a lot of elements and attributes are needed to be overwritten in case new elements or attributes are inserted into the document.

In general, transformation methods from XML document to relational database should satisfy many requirements. The significance of each requirement is application-dependant. In some applications it is extremely important to maintain order of nodes such as emails, books, journals and documents. In other applications, such as bank transactions, sales order and flight schedules documents, order is insignificant. Some of the requirements that should be met are the following:

1. Maintain document structure without loss of information while shredding.
2. Make the process of transforming a fresh document an easy task, and the updating of an already transformed document done with a constant time cost.
3. Ability to reconstruct the XML document or part of it from relational database.
4. Ability to perform semantic search.
5. Preserve the ordering nature of XML data and its structure.

In next chapters, the mapping model (MAXDOR) and the labelling technique introduced in this thesis will be represented. This model attempts to meet some of these requirements which are not available in the literature including the update problem.

CHAPTER 4 MAXDOR MODEL

This chapter gives a full description of the proposed model introduced by the author of this thesis. The new model is called MAXDOR for mapping XML document into relational database. The description includes mathematical concepts that are used in this model; the labelling method used to label XML document and identify its content, the design framework used to maintain the document structure, parent-child, ancestor-descendant, and siblings relations among document contents. It also presents a set of algorithms for mapping, reconstructing, updating and retrieving XML documents

4.1. MAXDOR Theory

Storing XML document into relational database means storing ordered, hierarchical and structured information into an unordered tables. XML manipulation is still facing some problems such as retrieving information, updating data contents, concurrency control and multi-user access. These problems can be overcome by using relational database to store, update and retrieve XML documents contents. Labelling techniques are used in order to preserve XML document structure, and the relations among its contents. MAXDOR adopts the Global Labelling method with some modifications (Tatarinov *et al.*, 2002). Global Labelling is modified to make the cost of the execution time of XML document updating constant, and to preserve parent-child and ancestor-descendant relationships. The modified method uses document structure information to guide the mapping process, Consequently DTD or XML Schema information availability is not required.

4.1.1 Theory Background

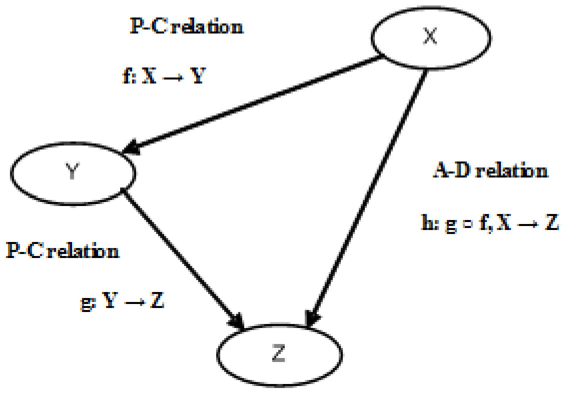
The hierarchy of XML document could be represented as a tree structure. XML tree can clearly represent the relationships between nodes of document content. Definitions 1 and 2 identify composite and associative relations between XML document elements, both as parent-child and ancestor-descendant relations. These relations help retrieve XML document contents as regular XPath expressions, and optimize query process. More details are given in section 4.2.4.

Definition 1: Composite relation

Given that f is a parent-child relation between X and Y, in away that f: $X \rightarrow Y$, and g is a parent-child relation between Y and Z, g: $Y \rightarrow Z$, then the composition h: $g \circ f$ is ancestor-descendant relation between X and Z as h: $X \rightarrow Z$, (Oosten, July 2002). ***Figure 4.1*** illustrates this composite relation.

Definition 2: Associative relation

Suppose f is a parent-child relation between X and Y as f: $X \rightarrow Y$, g is a parent-child relation between Y and Z as g: $Y \rightarrow Z$, and h: is a parent-child relation between Z and W as h: $Z \rightarrow W$, then the composition i: $g \circ f$ is ancestor-descendant relation between X and Z, j: $h \circ g$ is ancestor-descendant relation between Y and W, and K: $(h \circ g) \circ f = h \circ (g \circ f)$ is also ancestor- descendant relation between X and W, (Oosten, July 2002). ***Figure 4.2*** illustrates this associative relation.



Where: P :: Parent, C :: Child, A :: Ancestor, D :: Descendant

Figure 4.1: Composite parent-child relations

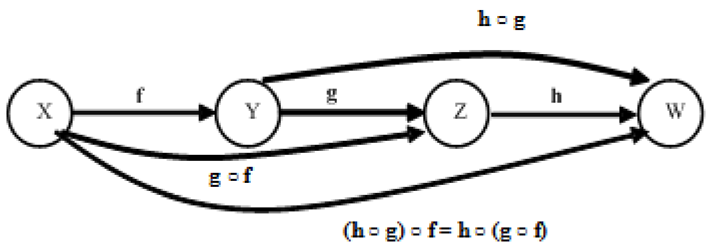


Figure 4.2: Associative ancestor-descendant relations

Definition 3: An XML tree is a collection of many nested subtrees of depth two. It can be denoted as follows:

$$T = \sum_{i=1}^{n} \sum_{j=1}^{m} S_{ij} \qquad (4.1)$$

where:

J = 1, 2, 3 … m represent the order of subtree number within i^th level;

I = 1, 2 …. n represents tree level number and 1 also represents the tree root; and

S_{ij} represents a subtree structure and is denoted as

$$S_{ij} = E_{ij}\left(\sum_{r=1}^{l_1} \mathrm{X}_{jr}, \sum_{k=1}^{l_2} \mathrm{A}_{jk}, \sum_{w=1}^{l_3} \mathrm{E}_{jw}, \sum_{z=1}^{l_2+l_3} \mathrm{G}_{jz}\right) \qquad (4.2)$$

where:

E_{ij} represents the root of the subtree S_{ij}

l_1 represents number of text (X) in subree S_{ij}

l_2 represents number of attributes (A) in subree S_{ij}

l_3 represents number of elements (E) in subree S_{ij}

$G_{i,z}$ is a finite set of edges between E_{ij} and its childs representing parent-child relationship (l_2+l_3).

An XML document is a tree of nested elements, each element can have zero or more attributes. There can only be one root element, which is called document element. Each element has a starting and ending tag, closed by angle brackets, with content in between:

<element>…content…</element>

The content can contain other elements, or can consist entirely of other elements, or might be empty. Attributes are named values which are given in the start tag, with the values surrounded by single or double quotations:

<element attribute1="value1" attribute2="value2">

One of the important characteristics of XML document is 'well-formed'. A well-formed XML document conforms to some rules, such as:

- Having only one root element.
- All start tags have matching end tags.
- Elements must be nested properly.
- Attribute values must always be quoted.
- Tags are case sensitive.

These restrictions on XML document structure makes shredding process and storing of XML document in relational database easier.

Definition 3 moves the organization of XML document, from being a tree of multi-dimensional way with arbitrary depth and width, to a tree structure of depth two. The resultant tree is unbounded fan-out at first level and fixed fan-out at the second level. The first level can be represented in relational database as tuples (i.e. rows) and the second level can be represented by fields (i.e. columns).

The processing and handling XML content is very important in optimizing data updating and retrieval. The search space is reduced into a subtree instead of working with the entire document tree. Consequently, definitions 4 and 5 given below make it possible to deal with an XML document as a dynamic-sized partition.

Definition 4: A dynamic fragment (shred) *df(i)* is defined to be the attributes and text (i.e. child leaves) of the subtree *i* of the XML tree plus its root r_{i-1}, as follows:

$$df(i) = (A_i, X_i, r_{i-1}) \qquad (4.3)$$

where:

A_i is a finite set of attributes in the level i

X_i is a finite set of text in the level i.

r_{i-1} is the root of the leaves in level *i*.

Definition 5: The root of the fragment (shred) is the node that has an out-degree more than one.

Definition 6: A multiple linked list is a data structure in which each node has its data and contains links to the preceding node, the following node, and to the parent node.

Multiple linked lists give the ability to access its content in different directions, and to insert a new node in constant number of operations. This makes it possible to update document in contact time cost, and efficiently retrieve preceding sibling element, following sibling, and parent-child. But more space is needed to create this type of linked list than single and double linked list. This issue is considered as a drawback for multiple linked list over single and double linked list.

Figure 4.3 gives an overview on the multiple linked list and the relations between its nodes.

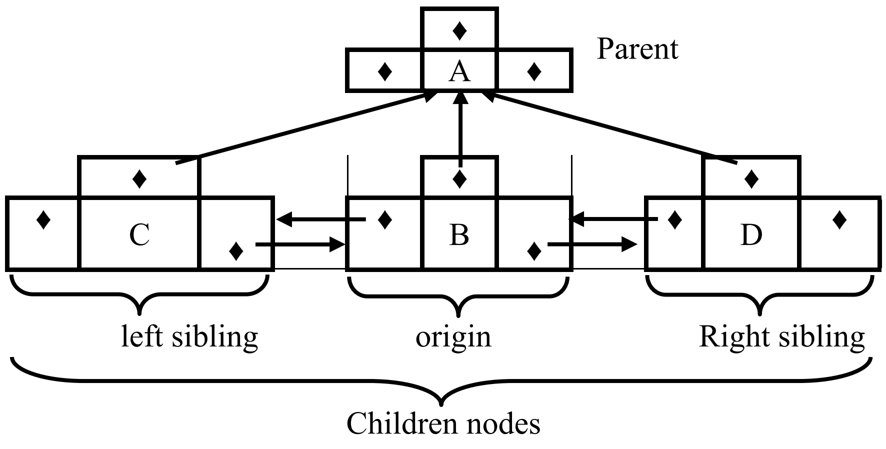


Figure 4.3: Multiple linked list over view

4.2. Mapping Framework

The mapping framework includes an algorithm to map XML documents into relational database and an algorithm to reconstruct XML documents from relational database. It also includes a method for updating stored XML document in relational database and querying and retrieving stored data from relational database. User's queries in XQuery or XPath languages are transformed into SQL statements, and SQL results are constructed into XML data format. Our approach considers well formed XML documents, which are shredded and decomposed into elements and attributes, and then these elements and attributes are inserted into the relational database tables. It does not consider the XML schema for the following reasons:

- Many applications need highly flexible XML documents whose structure is not easy to define by DTD or fixed schema. Therefore, schema-less approach is better to deal with such XML documents.
- It is not practical to design many candidate relational schemas for all potential XML data which may have different XML schema.

4.2.1 Labelling Method

Four Dimensional Links (FDLs) are used to maintain the XML document contents. FDLs' uses a global labelling approach that gives labels for XML elements and attributes. A unique label is given for each element and attribute. The sequence of label is not essential as (Tatarinov et al., 2002; Soltan and Rahgozar, 2006). Point out, an initial pre-order traversing for the XML document is performed to assign a label for each element or attribute. No re-labelling is needed for XML document elements and attributes (tokens) in case of adding new element or attribute. In contrast (Tatarinov *et al.*, 2002), (Torsten, 2002), (Soltan and Rahgozar, 2006), and (Torsten *et al.*, 2004); proved the reverse, all tokens that follow the new inserted token should be relabelled. In pre-order, post-order two labels are to be updated. In order to achieve this objective, FDLs uses the following format to identify a token:

- Token *(tokenID, leftID, parentID, rightID, prevID)*
- *tokenID* is a unique label given to identify each token.
- *leftID* (Left-sibling) is the *tokenID* of the preceding sibling token.
- *parentID* (Parent) is the *tokenID* of the current token parent.
- *rightID* (Right-sibling) is the *tokenID* of the following sibling token.

- *prevID* is the *tokenID* of the previous token of the current token in the document structure.

Figure 4.4 shows an example of FDLs labelling method for XML tree structure and identifies the relationships between its contents.

The *tokenID* and *parentID* are used to maintain the parent-child and ancestor-descendant relationships, while *leftID* and *rightID* together with *tokenID* are used to maintain elements and attributes order as siblings and brothers relationships within the documents structure.

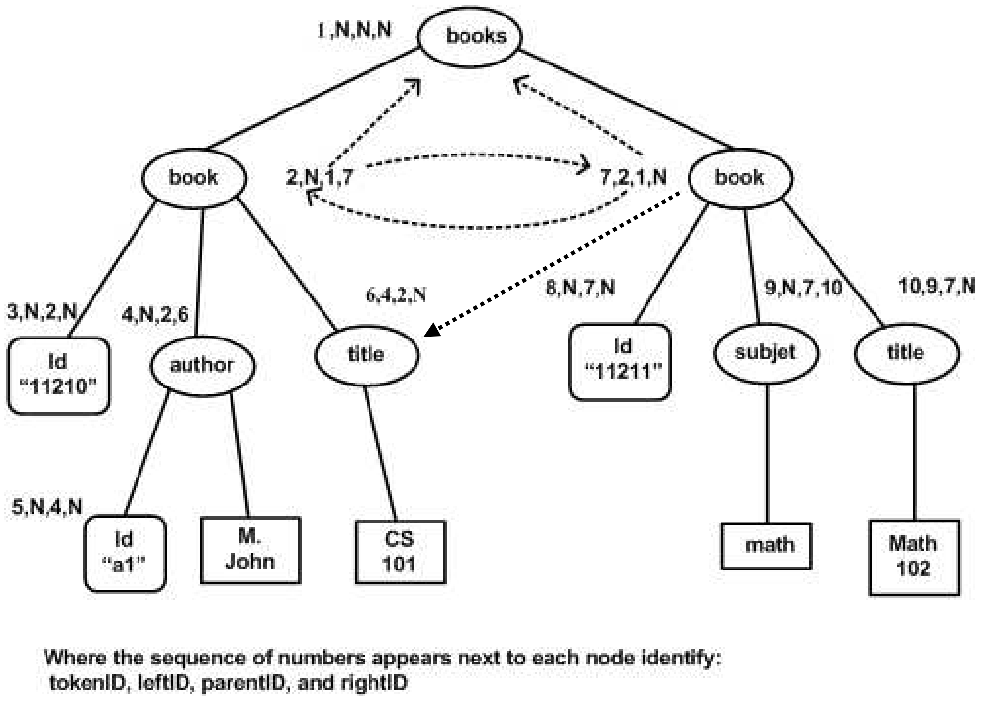


Figure 4.4: A tree representation for XML document

A fixed relational schema consisting of three tables is used to store XML documents' contents and their structure. The first table is called "documents table"; it preserves XML documents information. The second table is called "tokens table"; it preserves XML documents contents and structure. The third table is called "XpathQuery table"; it is a temporary table used to preserve token paths for a desired XPath expression from a document's root down to the desired token.

4.2.2 Relational Schema

This section gives a description of the relational schema used in FDLs, which consists of the following tables:

1. Document master table: It is called "documents table". This table keeps information about documents themselves; its minimal structure is:

   Documents(<u>documentID</u>, documentName, Header, docElement, schemaInfo, maxTokenId, XpathCount)

a. *DocumentID* is a unique ID generated for each document.

b. *DocumentName* is the external name for XML document.

c. *Header* is used to keep document header which specifies document encoding.

d. *SchemaInfo* keeps the document's schema if it exists for documentation purpose.

e. *DocElement* represents the document's root.

f. *MaxTokenId* represents the number of tokens in the document (i.e. total number of elements and attributes). It is used for future insertion, since a new inserted token is given a new ID following the last token number given in the document.

g. *XpathCount keeps the number of paths created for a specified query.*

2. "*Tokens table*" A table to store the actual content and structure for all documents. Documents will be shredded into pieces of data called tokens. Each document element, or element attribute will be considered as a token. The "tokens table" will have the following structure:

Tokens(<u>documentID, tokenID</u>, leftID, parentID, rightID, treeLevel, prevID, tokenName, tokenValue, tokenType)

a. *TokenID* field is the primary generated ID for each token.
b. *DocumentID* field is a foreign key linking the "tokens table" with the "documents table" to achieve referential integrity constraint.
c. *LeftID* (left-sibling) field keeps the ID of the left sibling token of current node. It is used to preserve tokens' order and document's structure.
d. *ParentID* field keeps the ID of parent's node. It is used to preserve parent-child and ancestor-descendant relations.
e. *RightID* (Right-sibling) field keeps the ID of the right sibling token of current node. It is to preserve the document's structure and tokens' order.
f. *PrevID* field keeps the ID of the previous token in the document structure.
g. *TreeLevel* field reserved the token level in the document or tree. It is starting from 1 for document element and increases by 1 for the nested element.
h. *TokenName* field is the tag name or the property name as found in the original XML document.
i. *TokenValue* field is the text value of the XML tag property.
j. *TokenType* field is used to differentiate between elements and attributes. (1 = element, 2 = attribute).

3. "*XpathQuery table*": A dummy table that is used to store all tokens involved in desired XPath expression. This table will have the following structure:

XpathQuery(<u>documentID</u>, XpathID,tokenID, TreeLevel, ParentID, tokenName, TokenValue, TokenType)

a. *DocumentID* field is a foreign key linking the “XpathQuery table” with the “documents table” to achieve referential integrity constraint.

b. *TokenID* field is the primary generated ID for each token.

c. *ParentID* field keeps the ID of parent’s node. It is used to preserve parent-child and ancestor-descendant relations.

d. *TreeLevel* field reserved the token level in the document or tree. It is starting from 1 for document element and increases by 1 for the nested element.

e. *TokenName* field is the tag name or the attribute name as found in the original XML document.

f. *TokenValue* field is the text value of the XML tag property.

g. *TokenType* field is used to differentiate between elements and attributes.

(1 = element, 2 = attribute).

Figure 4.5 represents the Entity-Relationship (ER) diagram for MAXDOR model showing the entities and the relation types connecting them. While ***Figure 4.6*** represents the relational schema used in MAXDOR model and shows the three tables (i.e., “Documents table”, “Tokens table”, and “XpathQuery table”), their attributes and primary keys.

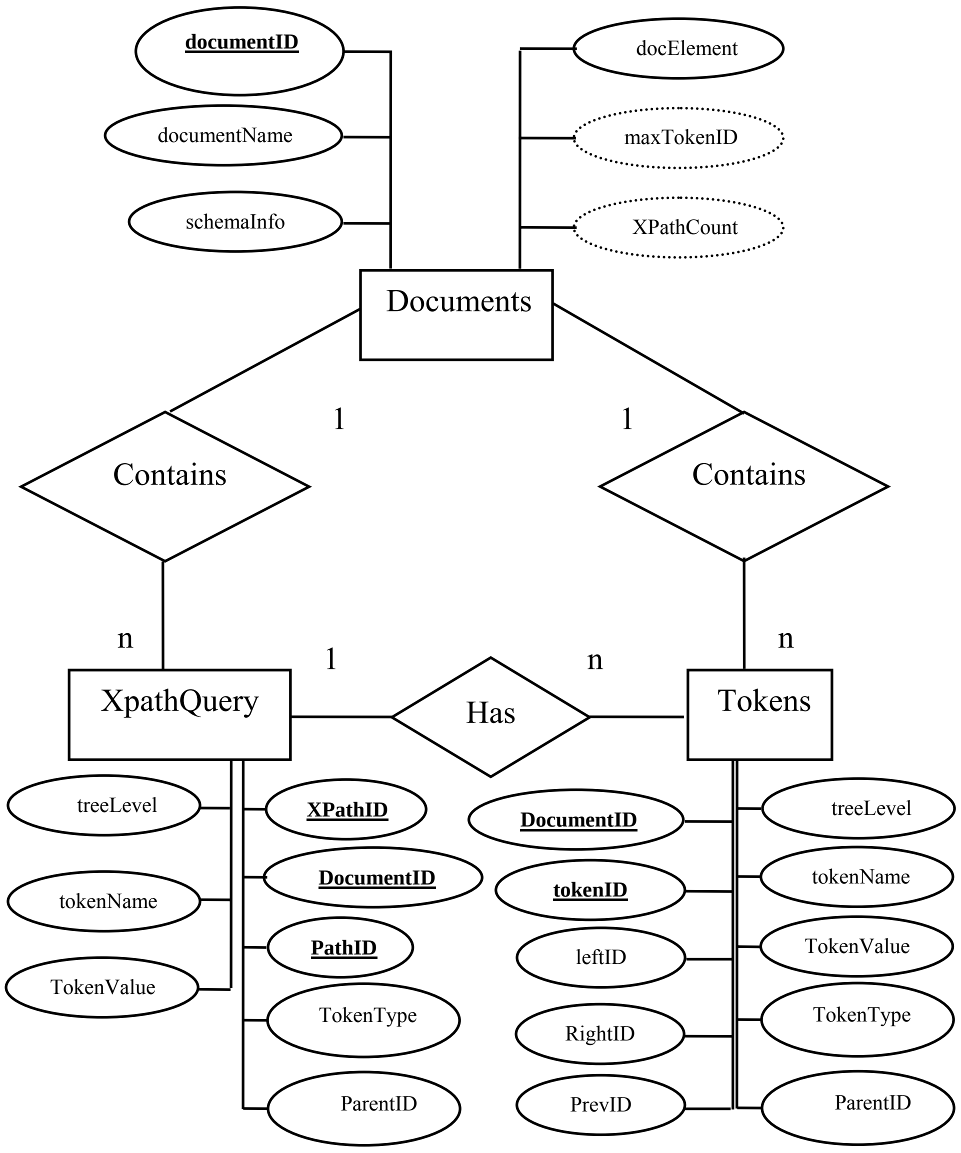


Figure 4.5: The entity-relationship diagram ER of MAXDOR model

- **Documents**(documentID, documentName, docElement, maxTokenId, maxPathId, schemaInfo)
- **Tokens**(documentID, tokenID, leftID, parentID, rightID, treeLevel, prevID, tokenName, tokenValue, tokenType)
- **XpathQuery**(documentID, XpathID, tokenId, TreeLevel, ParentId, tokenName, TokenValue, TokenType)

Figure 4.6: Relational schema

4.2.3 SAX-Based Approach

SAX parser (Megginson, 27-April 2004) is used for parsing XML document in order to store it in relational database. It is used instead of DOM (Document Object Model) to deal with large XML documents. SAX parses XML document as a sequence of events (i.e., startDocument, endDocument, startElement, endElement … etc), in the contrary of DOM that constructs the whole document tree (in memory) first and then parses it. DOM has an advantage over SAX that it offers XML update, but SAX provides XML for read only. In our approach updating XML contents is provided over the data stored in relational database and not on the XML document itself.

4.2.3.1 Mapping XML Document to Relational Database Algorithm

In this algorithm, the XML document is scanned once and is shredded into tokens. Each token represents one element or an element attribute in the document. The hierarchical structure of XML document imposes the use of a stack data structure. The stack is used to preserve element information that establishes links between sibling elements. These links (ParentID, leftID and rightID, prevID) are used to preserve document structure and the order of elements within the document.

The system automatically assigns each document a unique identification (DocumentID). During document scanning, maximum token identification is automatically generated for each new token. And any new inserted element or attribute will be assigned a new token ID following the maxTokenID value.

A document is scanned sequentially as tree structure in pre-order traversal. And the generated elements and elements' attributes are assigned token IDs in that order. As the document scanned sequentially, all descendant elements are pushed into the stack buffer formulating a full path from the document root (i.e. document element) going down through descendant element until reaching leaf nodes.

Attributes of elements are written directly to the "Token table", since they are leaf nodes listed in order at the starting tag of an element, and their relations (Parent-child, preceding-sibling, and following-sibling) are easily formulated at this stage. The left-sibling of the first attribute is assigned zero identifier. While the right-sibling and left-sibling links between element attributes are assigned incremental identifiers as a new attribute is caught. The right-sibling of the last attribute is assigned zero identifier.

The stack reserves information of elements in order to create the links between sibling elements. A right-sibling of the current element can not be assigned until the next sibling is caught, which can not be done until all the descendant elements of the current element are scanned. Once the right sibling of an element is caught, or an element whose tree level is less than the element's level which is found at the top of the stack, all elements in the stack will have tree levels. This is greater or equal to that element level which popped from the stack and the appropriate links are established for these elements. Finally, the new element is pushed to the stack.

Stack size depends on two factors:

1- The depth of the document; stack size is directly proportional in this case to the document depth.

2- The length of elements' names and values. Also in this case, stack size is directly proportional to the length of elements’ names and values.

Stack size can be managed as follows: In most document-centric XML, document depth is less than that of data-centric document, while elements' names and elements' values are larger than that of data-centric document. Experiments in Chapter 6 applied to selected data sets will give more clarification on this statement.

An implementation of this algorithm is described in Chapter 5 as XML2Base class, and experiments on different data sets are done in Chapter 6 to test the algorithm usability and performance.

4.2.3.2 Rebuilding XML Document from RDB

The rebuilding process of XML document from relational database is needed for the following reasons:

1. To make sure that the mapping method, used in the research, efficiently maintain the entire XML document without losing information.
2. To update document content after being mapped into relational database; updating takes place in the relational database. So, the original XML file become obsolete; i.e. not reflecting the current state of the content of database table.

For the preceding factors, a rebuilding algorithm is used to:

1. rebuild the entire XML document that can be exchanged or exported by the user somewhere, or
2. rebuild part or some parts of the document as a result of user queries using XPath or XQuery that are translated into SQL statements retrieved by relational database system.

Reconstruction or rebuild algorithm depends on the labelling method and the relational schema described in previous sections used for MAXDOR model. It manages the rebuilding process in two ways:

1. Fresh document or un-updated (unchanged) document: In this case, the document is built as it was read from relational database, and in the sequence it was stripped in. A stack data structure is used to reserve ending tags of ancestor elements. As a starting tag of the element and its attributes (if it exists) are written directly to the output XML document file. The algorithm uses treeLevel in order to manage nested elements. As new element is identified to be next element, its treeLevel is compared to the top element of the stack; if it is less than the top of the stack, then pop the stack, write the popped element to the output file, write elements' closing tag, until the top of the stack becomes less than or equal to the new element. Finally, the new element closing tag is pushed into the stack. The process is repeated until building the entire document is completed.
2. Updated document: to manage document fragmentation that resulted from updates (insertion and deletion) on a document, three stack data structure are being used because no relabeling is allowed for document contents after insertion to reserve element order. These stacks are:
   I. A stack for pending elements: It is used to hold elements that can not be written directly to the output file since new inserted elements are assigned labels not in the same order of their predecessor elements in the document structure. Those elements should be written to the output file before their new successors after pended elements on the stack.

II. A stack for element attributes: It is used to manage element attributes in their logical order to be written in the same order as their order is in the element starting tag.

III. A stack for nested elements closing tags: It is used to reserve ancestors' closing tags because processing goes from parent to child. If a new element is caught and its treeLevel is less than that of the element treeLevel in the stack, then all elements of treeLevel greater than that or equal to treeLevel of the new element pop and their closing tags are written to the output file as XML document.

Whenever a new element is caught, and before writing it to the output file, its attributes are popped from the attributes' stack and appended to it.

An implementation of this algorithm is described in Chapter 5 as xbsXML2Base class. In addition, some experiments on reconstructing XML documents from relational database are conducted in Chapter 6.

4.3. Updating XML Document Contents

4.3.1 Insertion of New Token

This section gives more evidence that the method used in this research makes insertion time cost of new token, anywhere in the document, constant; this, one of the main objectives of the research is achieved. The insertion process can be clarified by the following rules:

a. Insertion of a new token to the left of a subtree, left to S1:

1) The new token T gets a label tokenID following the maxTokenID in the document. *TokenID(T) = maxTokenID + 1.*
2) *RightID(T) = RightID(S1)*
3) *LeftID(T) = 0*
4) *LeftID(S1) = tokenID(T)*

5) *ParentID(T) = ParentID(S1)*
6) *prevID(T) = prevID(S1)*
7) *PrevID(S1) = tokenID(T)*

b. Insertion of a new token to the right of a subtree, right to S1:

1) The new token T gets a label tokenID following the maxTokenID in the document. *TokenID(T) = maxTokenID + 1.*
2) *LeftID(T) = TokenID(S1)*
3) *RightID(T) = 0*
4) *RightID(S1) = tokenID(T)*
5) *ParentID(T) = ParentID(S1)*
6) *prevID(T) = prevID(followS1)*
7) *PrevID(followS1) = tokenID(T)*

c. Insertion of a new token T as a leaf and child of S1:

1) The new token T gets a label tokenID following the maxTokenID in the document. *TokenID(T) = maxTokenID + 1.*
2) *LeftID(T) =0*
3) *RightID(T) = 0*
4) *ParentID(T) = TokenID(S1)*
5) *prevID(T) = tokenID(S1)*
6) *PrevID(followS1) = tokenID(T)*

d. Insertion of a new token T as a parent of S1:

1) The new token T gets a label tokenID following the maxTokenID in the document. *TokenID(T) = maxTokenID + 1.*
2) *LeftID(T) =LeffID(S1)*
3) *RightID(T) = RightId(S1)*
4) *ParentID(T) = ParentID(S1)*

5) *TreeLevel(T) = TreeLevel(S1)*

6) *prevID(T) = prevID(S1)*

7) *PrevID(S1) = tokenID(T)*

8) *LeftID(S1) = 0*

9) *RightID(S1) = 0*

10) *Look for all descendant and update treeLevel by 1*

Figure 4.7 gives an overview of inserting new token (i.e. element or attribute) in the XML document. In the figure, the new element "subject" is inserted between "author" (labelled [4, N, 2, 6]) and "title" of label [6, 4, 2, N]. The new element is given tokenID equals to maxTokenID + 1, which is 11. And the token links are updated as follows:

1) *rightID(subject) = rightID(author)*

2) *leftID(subject) =leftID(title)*

3) *rightID(author) = tokenID(subject)*

4) *leftID(title) = tokenID(subject)*

5) *prevID(subject) = prevID(title)*

6) *prevID(title)= tokenID(subject)*

7) *ParentID(subject) = ParentID(title)*

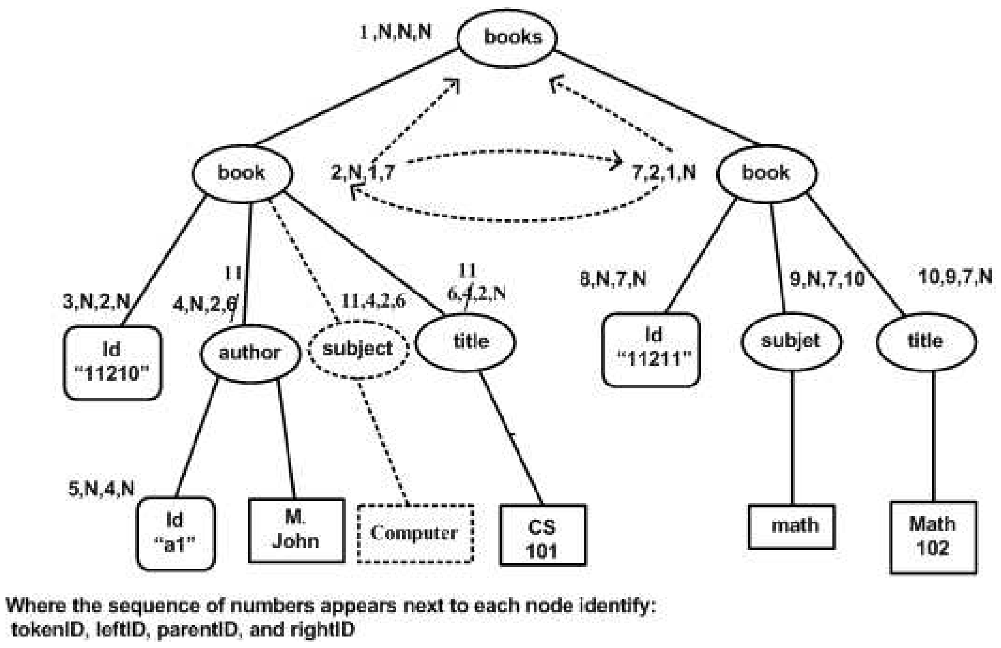


Figure 4.7: Inserting new token in XML tree

As seen from the previous example, there is no need for relabeling the tokens that follow the inserted token "subject". All tokens' labels in the document remain as they were before the insertion process. While in (Tatarinov *et al.*, 2002) (Torsten, 2002) (Soltan and Rahgozar, 2006), all the following nodes of new inserted element “subject” must be relabelled, and the cost of relabeling depends on the location of the new inserted element. The highest cost is gained when the insertion happens at the beginning of the document, and the lowest cost is gained when the insertion takes place at the end of the document.

An implementation of this algorithm is described in Chapter 5 as dbxTokens class. And an evidence of the previous claim that insertion of new tokens anywhere in the document is done on constant time cost is shown as the experiments in Chapter 6 demonstrate.

4.3.2 Deletion of a Token:

Deletion of existing tokens from any location or level in the XML document can be done also with constant cost. This deletion process follows the following rules:

Note: The maxTokenID field will not be changed (i.e. not decremented), since no relabeling of the tokens within the document will be done.

a. Deletion of a token T between two siblings, S1 and S2:

1) *RightID(S1) = RightID(T)*
2) *LeftID(S2) = LeftID(T)*
3) *prevID(s2) = prevID(T)*

b. Deletion of a token from the left side of a subtree, to the left of S1:

1) *LeftID(S1) = 0*
2) *prevID(followT) = prevID(T)*

c. Deletion of a token from the right of a subtree, to the right to S1:

1) *RightID(S1) = 0*
2) *prevID(followT) = prevID(T)*

d. Deletion of a complex element:

Deletion of a subtree can be handled as a single token by one of the previous three cases, but all its descendants should also be deleted.

4.4. Retrieving and Querying XML Data Stored in Relational Database

Mapping XML documents into relational database is not just for storage and back-up. This data is stored so as to be efficiently updated and retrieved. In our proposed method, the XML Path Language (XPath) is used as a source tool for retrieving and querying the XML data stored in the relational database. The XPath expressions will be translated into its equivalent SQL statements in order to get the results from the relational database.

“XpathQuery table” is used as a temporary table to isolate XPath query results at run time from the database main tables. Its content is the result of walking through the tree side by side according to the XPath command, filtered as required, and getting the records (nodes) while doing so. This method has minimal cost, since in path methods we have to select the records too. It is different from Path table methods since those approaches building a table of all expected queries in the DBMS during the mapping time will result in increasing the database size (O'Neil et al., 2004; Jiang et al., 2002; Yoshikawa et al., 2001),.

In the following sub-section, a discussion for XPath axes (i.e. parent, child, ancestor, descendant, following, following-sibling, preceding, and preceding-sibling), translating of XPath expression to SQL statements, and building their results in XML format are also presented.

4.4.1 XPath Axes

XPath has mainly 8 axes used for retrieving XML document content. ***Figure 4.8*** gives a clearer view of these axes. Consider G as a candidate node, and the nodes:

- Node B is a parent of G.
- Nodes H and I are children of G.
- Nodes A and B are ancestor of G.
- Nodes H, I and J are descendant of G.
- Nodes C, D, E and F are preceding of G.
- Nodes C and E are preceding-sibling for G.
- Nodes K, M, L and N are following for G.

- Nodes K and M are following-sibling for G.

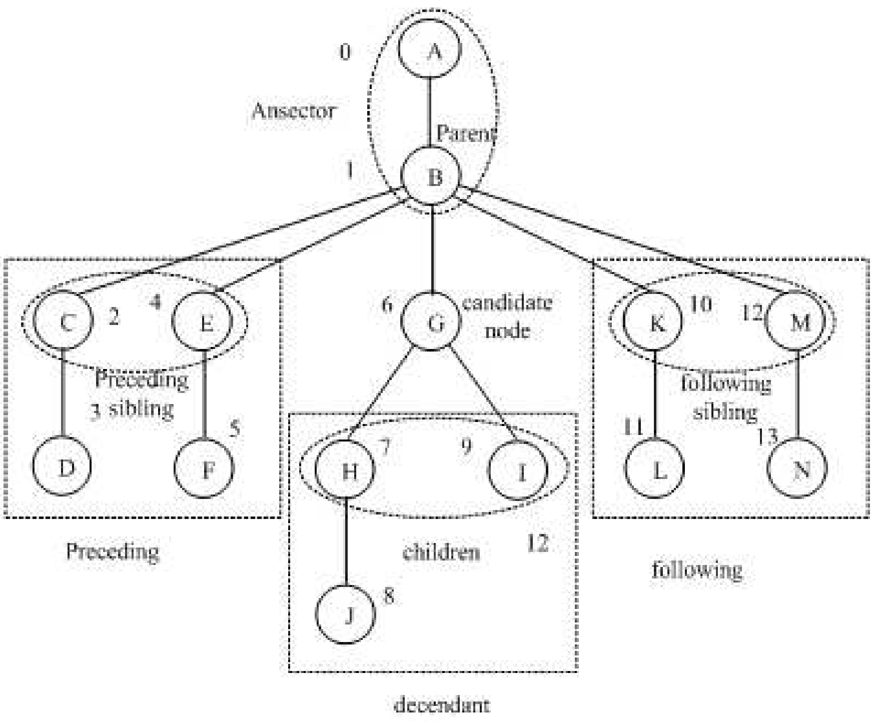


Figure 4.8 Nodes relationship in XML tree structure

Here is an explanation of how MAXDOR labelling method supports these axes. Given x and y as nodes in the XML document n:

I. Parent and child axes: node x is a parent of node y if and only if its tokenID is assigned as parentID of node y and its level is greater than its parent level by 1.

Document ID	TokenID	ParentID	LeftID	RightID	TokenName	TokenValue	TokenType
n	1	0	0	1	x	B	1
n	6	1	4	10	y	G	1

II. Ancestor axis: All ancestor nodes of node x can be retrieved as nested parent axes starting from node x in reverse order. All ancestor nodes of x are formulated and located on the same path.

III. Descendant axis: All descendant nodes of a node x can be retrieved as nested parent-child axes. They are retrieved recursively from left to right, as each of its children is a subree. The left most child of x has leftID equal to zero. Move right until the right most child having rightID = 0.

IV. Following axis: All nodes following a node x can be retrieved as follows:

1. If RighID of x is not equal to zero, then the right node of x is considered as a starting node to be retrieved, and retrieve all of its following nodes. The process applies in the same way building the whole document, but the resultant XML document may not be well-formed.

2. If RightID of x is equal to zero, then find the node whose prevID equals the ID of node x. If it exists, consider it as the starting node to be retrieved and retrieve all of its following nodes. As in case 1, the process applies to building the whole document, but resultant XML document may not be well-formed

V. Preceding axis: All preceding nodes of a node x can be retrieved as follows:

   1. The process goes through the candidate node path starting from it up to the root node (i.e., node x and its ancestor nodes in reverse order), check the left-sibling of each parent. If its leftID is not equal to zero then push it to the stack.

   2. The parent nodes starting from the top are popped from the stack, and their preceding-sibling nodes are retrieved along with their descendant nodes according to cases VII and III.

VI. Following-sibling axis: All following-sibling nodes of a node x that has the same parent as x can be retrieved from rightID link, starting from rightID node of node x, as a sequence, until the right most sibling node(with rightID equals to zero) is reached.

VII. Preceding-sibling axis: All preceding-sibling nodes of node x, which has the same parent as x, can be retrieved from leftID link, starting from leftID node of node x, as a sequence, until the left most sibling node (with leftID equals to zero) is reached. In this case a stack data structure can be used to retrieve preceding sibling from left to right instead of right to left.

VIII. Attribute Axis: All attributes of a node x can be retrieved as an attribute whose parentID equals X's tokenID. To retrieve them from left to right, start by the attribute of leftID= 0, and move right by following rightID links until the right most attribute is reached.

Other Axes, as ancestor-or-self axis and descendant-or-self axis can be processed as ancestor or descendant axes including the context node.

4.4.2 XPath Syntax:

XPath uses path expression to select a node or a set of node from the XML document by following a path or a step. According to W3C (Berglund *et al.*, 2007), this selection can be performed as follows:

1- Nodes selection: This selection can be done through some expressions which appear in ***Table 4.1***.

Table 4.1: XPath expressions (Berglund et al., 2007)

Expression	Description	Comments
nodename	Selects children nodes of the named node	
/	Selects nodes from the root node	This expression is considered as absolute expression
//	Selects document nodes from the current node no matter where they are	This expression is considered as relative expression
.	Selects the current node	
..	Selects the parent of current node	
@	Selects attributes of the current node	

In this case retrieving XPath results can be dealt with as follows:

a- Expression of one step like /books or //name, the process can directly use the tokens table as:

```
SELECT t1.* FROM tokens as t1
WHERE t1.tokenName = x
And t1.documentId=n;
```

Where x is the specified token name and n represents the current document ID. But if the expression has multi step as /books/book or //book/authors/author, then nested inner joins on "Tokens" table and relation between “XPathQuery” table is needed to retrieve the desired tokens which have this path.

b- For “.” expression to select current node, the SQL statement for this expression is:

SELECT t1.tokenName, t1.tokenValue FROM tokens as t1
WHERE t1.tokenID = x And t1.documentId= n;

Where x represents the current token ID

c- For “@” expressions, the SQL can be:

SELECT t1.tokenName, t1.tokenValue FROM tokens as t1, token AS t2 WHERE t1.parentId = t2.tokenID
And t2.tokenID = x
And t1.documentID = n
And t1.tokenType= 2;

2- Path expression including predicates: Predicates in XPath are used to find a specific node, or a node that has a specific value. Usually, predicates are surrounded by square brackets. ***Table 4.2*** shows some path expressions that use predicates. In this case retrieving XPath results can be dealt with as follows:

a- For path expressions number 1 and 2, a nested inner joins between “XPathQuery” table and “Tokens” table are used to retrieve the desired elements that have this path. Left link or right link of selected tokens is also retrieved. In expression 1, leftID should be zero for the selected token, while in expression 2 rightID should be zero for the selected token. The following two SQL statements represent expression 1 and expression 2 respectively.

Table 4.2: Path expressions with predicate

No.	Path expression	Description
1.	/books/book[1]	Select the first book element that is a child of books element.
2.	/books/book[last()]	Select the last book element that is a child of books element.
3.	/books/book[last()]/isbn	Select the isbn element of last book element that is a child of books element.
4.	/books/book[isbn='42516 8']/title	Select the title element of book element which its isbn = '425168' and is a child of books element

b- For path expressions number 3 and number 4, the process goes for expression 1 and 2, and after identifying the desired tokens. Then a selection of tokens whose parent is in the selected set will be performed.

Where x represents the left part of "[", and y represents the right part of "]".

For both expressions 3 and 4 farther step is needed to identify the desired child.

An implementation of mapping XPath expressions into SQL statements algorithm is described in Chapter 5 as *frmQuery* class. And some experiments on different forms of XPath expressions are conducted in Chapter 6.

4.4.3 XML Sub-tree Reconstruction (Query's Result Translation to XML)

A user query result could be a group of separated single elements, or attributes or nested elements that can be consider in this case as a subtree. In case of group of separated elements or attributes, a "starting tag" and "closing tag" can be used to group the results as a single level tree which rooted by query result. This procedure helps in forming the result as a well-formed document. In case the result is a nested element, it can be built the same way as building an entire document, starting by the lowest level element as a root node instead of the document element. In both cases, the algorithm that is used for reconstructing XML document from relational database can be used also for building queries' results from relational format into XML format.

4.5. Chapter Summary

In this Chapter, a detailed description is given for the MAXDOR model. The description includes: the theory used for this model and tree facilities for representing and accessing XML document as the two structures are both hierarchical and nested. The labelling method used in designing the MAXDOR model is represented as Four Dimensional Links (FDLs), since a multiple linked list is used in our case. The links are used for parent node, left node, right node and previous node in the structure. Those links make the insertion time cost of new element or attributes anywhere in the document realized with a constant number of operations. Also the retrieving process of document contents can be done smoothly as the relations between its nodes are identified through those links. For example, left-sibling can be identified by leftID, right-sibling by rightID, parent by parentID, complex element by previous ID. The relational model used in the model is introduced as Entity-Relational diagram and relational schema. As three relational tables are used, two used to store document metadata, which are "documents table" and "XpathQuery table" while the third one "tokens table" is used to store document contents. A description of mapping XML document into relational database algorithm has been given with the data structure that is used to optimize the process. The rebuilding of XML document from relational database algorithm has been presented with two options. The first option is to reconstruct updated documents and the other to reconstruct documents that are updated within insertion or deletion processes. The update of XML document contents in relational database include inserting, deleting and allocating processes algorithm have been presented, and operations on different locations in the document have been done to show that the updating time cost is constant. At the end, a querying and retrieving document contents algorithm has been presented with some XPath axes and XPath expressions translated into SQL statements.

An implementation of MAXDOR model will be offered in the next chapter, (i.e. Chapter 5). This includes system architecture, the tools used, software needed for implementation, classes implemented for the model, data structure used for enhancing the model performance and the XML data set used for testing.

CHAPTER 5 SYSTEM ARCHITECTURE AND IMPLEMENTATION

This chapter presents the system architecture, and implementation tools used for evaluating the MAXDOR model. The chapter also presents the main classes created to demonstrate the methodology for Mapping XML document into relational database, Rebuilding XML document from relational database, Updating the content of XML document stored in relational database, and XPath-To-SQL query translation, and building the result in XML format. Application on a case study is also presented. XML data sets from selected XML bench marks and XML data repository will be identified to be used for testing and evaluating the model. Finally, the chapter concludes with a summary.

5.1. System Architecture and the Used Tools

5.1.1 System Architecture

System architecture consists of four main components each of which represents one of the project requirements. Those components are:

1- Mapping XML document into relational database: the system loads the XML document and parses it using XML SAX parser as a sequence of events, shreds the document content into tokens, and inserts these tokens into predefined relational database schema. Detail of the relational schema has been given in chapter 4.

2- Reconstructing XML document from relational database: this component goes through the relational tables and reconstructs the requested XML document to check the method for lossless of XML document information in Part one or to exchange or export the document to other location.

3- Updating XML document stored in relational database: by this component, the user is given the facility to update the XML document stored in relational database. Update includes: inserting new tokens as element or element attributes, delete tokens or tokens' re-allocation within the document, and modify tokens' name and values.

4- Retrieving and querying XML document stored in relational database: throughout this component, XPath queries are translated into SQL statements. The resultant SQL statements are fired against the database engine so as to retrieve XML data results. The retrieved results are reconstructed as XML hierarchical format and returned to the user. ***Figure 5.1*** gives an overview of the system architecture.

The above listed components will be tested and evaluated in Chapter 6 for the MAXDOR model described in chapter 4. The following points are taken into consideration during system design; i.e. components of the system should be:

- Testable against requirements - every requirement should be easy to test.
- Structured – the system structure should be clear, read and its code should be easy and understandable.
- Reusable – the system design should be reusable and repeatable.

5.1.2 Tools Used

The tools used in the project can be classified into:

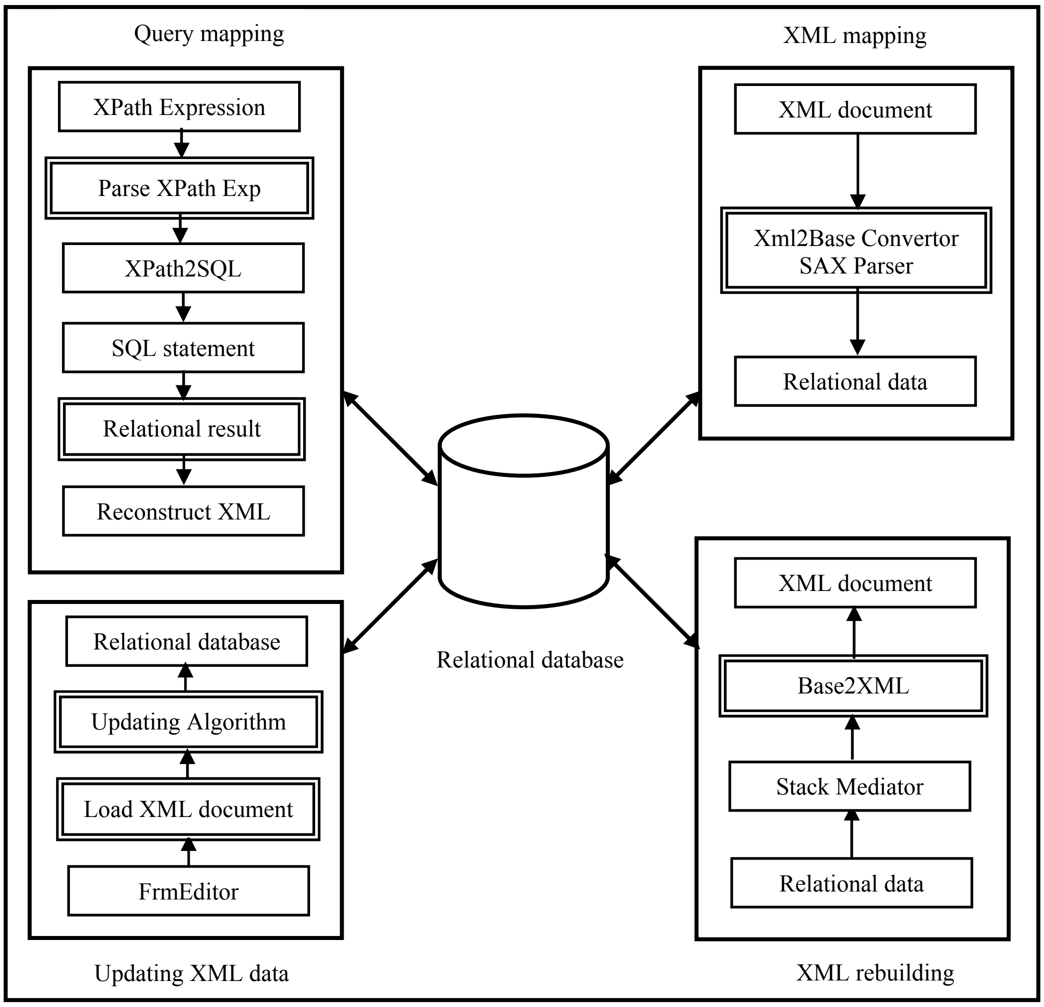


Figure 5.1: MAXDOR Architecture

1- XML interface: both input and output documents are XML format and relational database technology is a target tool for storing XML documents' contents and structure; so, the relational database capabilities are used for internal processing of data.

2- XPath or XQuery as source languages provided for users to represent their requests. SQL query language is a target language used against relational database to answer users' queries. XPath is used for the following two reasons:

- XPath is simpler than XQuery, and hence would be better to achieve the objective of testing our model in current situation.
- Its structure is included in XQuery, so it is easier to be upgraded into XQuery.

3- Visual Basic 6.0 programming language is used as a tool to create the GUI and to implement the system components. It is used for the following reasons:

- VB structure is simple, mainly as to the executable code.
- VB is easy for building graphical user interfaces.
- VB application is easily connected with Microsoft Access database.

4- Microsoft Office Access is used as a relational database management system (RDBMS). It is used for the following reasons:

- It can be easily used with visual basic programming language.
- Access database can be easily sited on a website for access by remote users. Simple screens can be built in Access, Data Access Pages. Or it can be employed using Active Server Page (ASP) scripting.

5.2. System Implementation

5.2.1 Requirements for System Implementation

1- Microsoft Office 2003 or 2007 is required since we are using Microsoft Access as the development DBMS for the system.

2- Microsoft Windows 2000 with service pack 3 (sp3), Windows XP, or later, because Microsoft Office Access 2003 is used, and this is its minimum requirement of the operating system.

3- Minimum hardware requirements are given in ***Table 5.1.***

Table 5.1: Hardware requirements for Microsoft Office 2003 (Corporation, 2009)

Computer and processor	Personal computer with an Intel Pentium 233-MHz or faster processor (Pentium III recommended)
Memory	128 MB of RAM or greater
Hard disk	150 MB of available hard-disk space; optional installation files cache (recommended) requires an additional 200 MB of available hard-disk space
Drive	CD-ROM or DVD drive
Display	Super VGA (800 × 600) or higher-resolution monitor

5.2.2 Classes of the MAXDOR Model

In the following sections a description of the main classes used for system implementation is given. These classes are *xbsXml2Base*, *xbsBase2XML*, *dbxTokens* classes and *frmQuery*.

5.2.2.1 XbsXml2Base Class

The data model used for the mapping algorithm uses the W3C's Simple Application Program Interface for XML (SAX parsing) (Megginson, 2004). A stack is also used to traverse the XML document. Each child of the element is pushed to preserve and identify nodes' order, element siblings and parent-child relationship. SAX parser fires actions on many events including document start, document end, element start, element end, characters, element attributes, and processing instruction. These events help in shredding XML document and store its contents into relational database tables. Four more links are added to token description, its parent ID, left sibling ID, right sibling ID, and previous token ID in the document structure. Left and right sibling IDs are used to make the time needed for future insertion in the document constant since these IDs could be updated as new node or subtree is added or relocated in the document. Previous token ID helps in rebuilding XML document with minimum cost because the document is built in sequential order, on top-down bases, (i.e. moving down through parent-child relationship and forward through sibling relationship).

A description of xbsXML2Base class is given below. The class takes XML document as input and generates its relational database tables as output. It mainly depends on the XMLSAX *Contenthandler* class (i.e. a custom class implementing the IVBSAXContentHandler interface). ***Figure 5.2*** shows the state transition diagram for this class. Few private methods are added to the class and their description is given below. The coding of the class is presented in Appendix B.

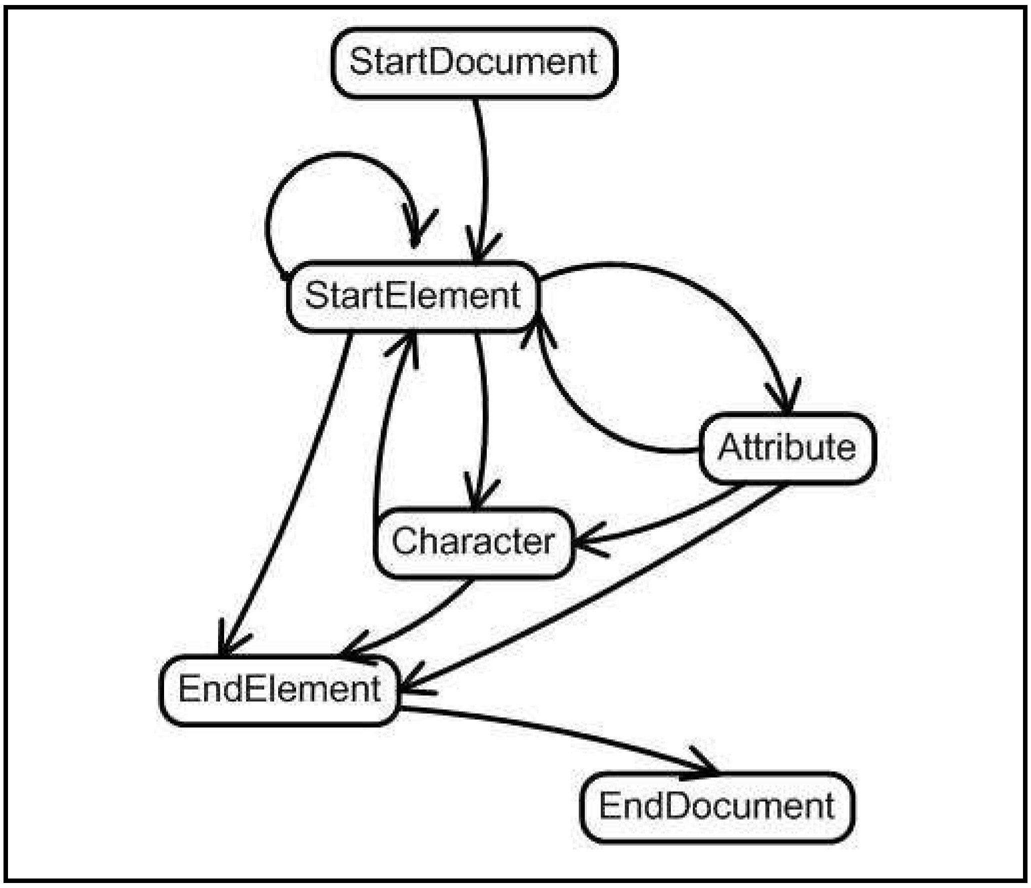


Figure 5.2: State transition Diagram for xbsXml2RDB class

The startDocument method is called just one time for each document. As it is called by SAX parser, the relational database tables are prepared to receive document information in “document table” and “tokens table”.

The three methods, *startElelment, characters* and *endElement* are called back by SAX parser depending on the document contents and contents sequences.

1- The *startElement* method is called by SAX parser whenever it encounters an XML start tag as <book id=”bk210”>. The parser gives tag name and the list of attributes if any. In this method, a stack data structure is used to manage document structure and build relations between document contents in our model.

2- The *characters* method is called whenever text content is seen as input in the document.

3- The *endElement* method is called whenever the corresponding end tag </book> is seen. In this method, a return to previous level is performed.

If SAX parser encounters Document end, it calls *endDocument* method. In this method, all pending elements in the stack are inserted into "Tokens table".

The stack data structure in this class is used to preserve parent-child and sibling relationships. The stack is used to hold the tokens' information of all elements for one path of the document, (i.e., ancestors' nodes of the current node). And that path identifies the size of the stack since the path size depends on the tree level or depth. In this case, the relation between the path size and the stack size can be considered proportional, and may decrease the performance of the method for documents with very deep levels.

5.2.2.2 XbsBase2Xml Class

The class is used to rebuild XML documents back from relational database to create new XML document from scratch since original document contents could be updated. The class depends mainly on two methods: *DirectBuild* and *BuildProps*. ***Figure 5.3*** shows the main processes of this class. A brief description for these processes is given below and the coding of the class is presented in Appendix B:

- The *select document elements* process is used to select the entire candidate document elements from "*tokens table*".
- *" Open output file"* process is used to open an output XML file for writing the candidate document contents in XML format.

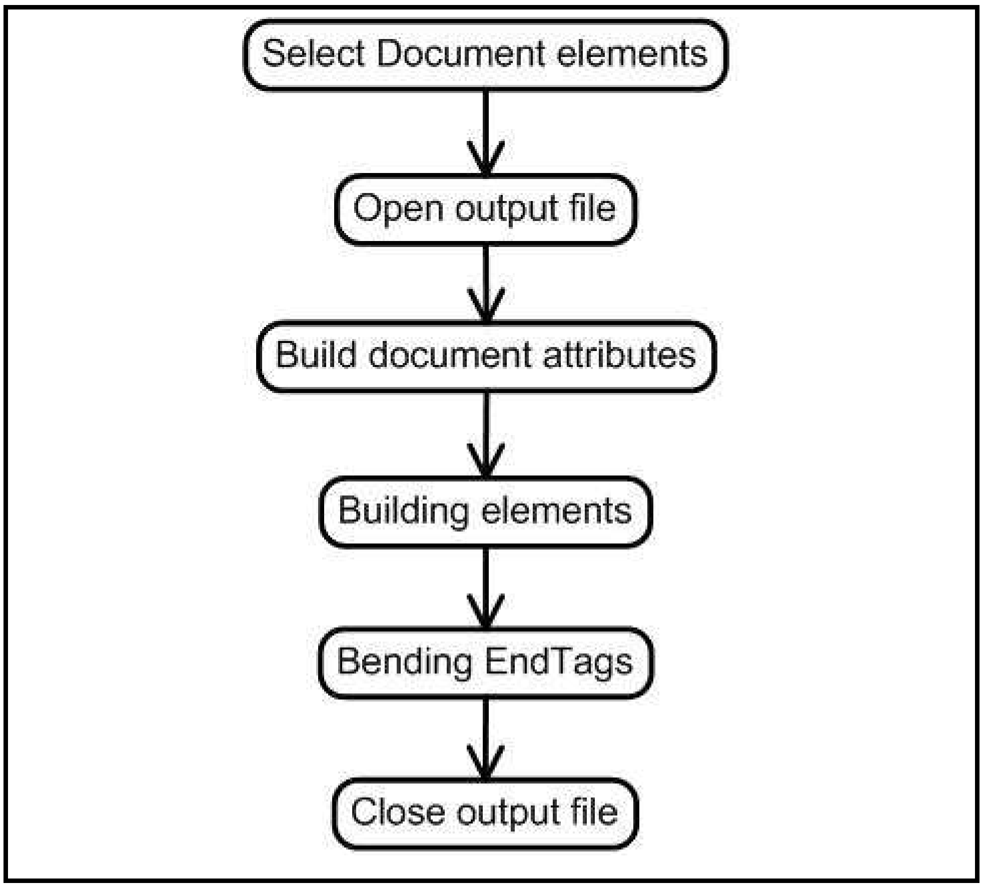


Figure 5.3: Rebuilding XML document from relational database state diagram

- *Buildprops method:* This method is used to read all attributes and put them on a stack for later use in a form of *"attribute vectors"*; each vector corresponds to a unique element (i.e. the attributes parent) and is composed of an ordered list of attributes in their original order.
- The *Building Elements* process: This is used to rebuild document elements depending on the *prevID* and *treeLevel* for elements in order to identify elements sequence and parent-child relation. The element starting tag, its attributes (if any is found on the *"attribute vectors"*), and value are written to *output file*. If the current element has children, its closing tag will not be written out but put onto a

stack till all sub-children are processed. The process uses a stack called *Clpending* to temporarily hold the elements that can not be written directly to XML file since newly inserted elements would break the sequence order of the labels.

- The *Bending EndTags* process: This is used to write all bending closing tags off the stack to the output files.
- The *Closing output file* process: This is used to close the output file and terminates the building process.

5.2.2.3 DbxTokens Class

This class consists of three groups of subroutines for editing XML documents: *inserting*, *updating* and *deleting* the tokens. ***Figure 5.4*** shows the state transition diagram for this class and the coding of the class is presented in Appendix B.

The editing process starts by loading the XML document contents from relational database using the *"frmeditor"*. When the document is loaded into the editor and a candidate element is selected, any one of the following processes can be performed:

I. Adding (i.e. inserting) new elements: Four different methods are used to perform this process depending on the position of insertion which are *InsTagBefore*, *InsTagAfter*, *InsTagBelow* and *InsTagAbove*. These four methods are used to insert new elements as left-sibling, right-sibling, child and parent respectively for the candidate element. The methods are different since different links have to be updated depending on the position of insertion. Insertion of new elements as a parent needs to update all descendent tokens level of candidate element.

II. Update element: Candidate element's name and value can be updated in this process.

III. Deleting selected token: This process is used to delete a candidate element and its entire descendant tokens (if any).

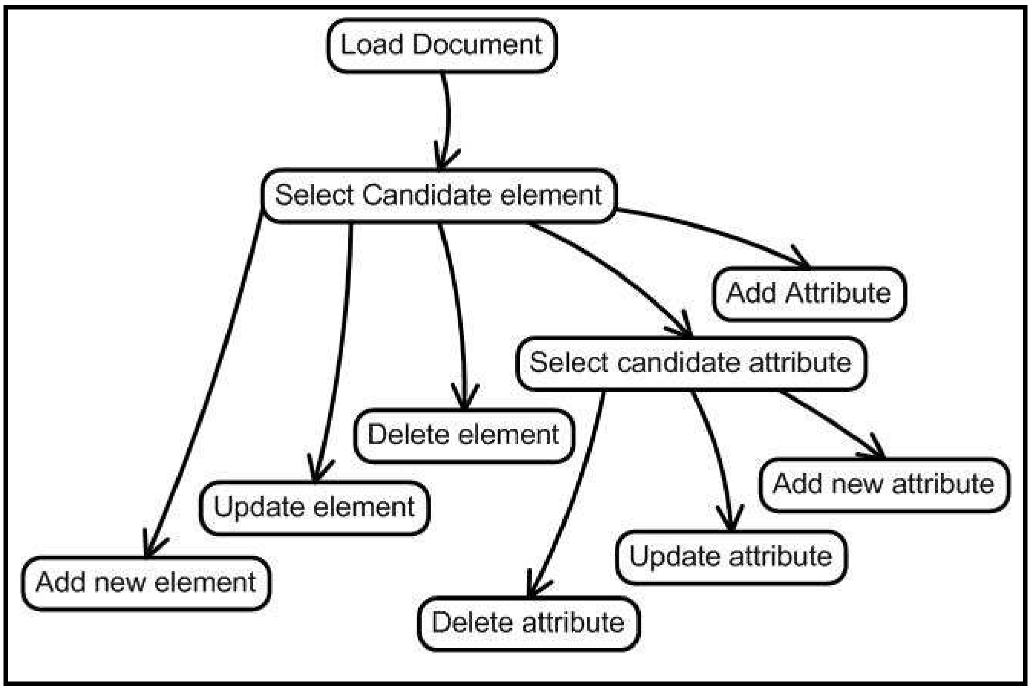


Figure 5.4: State transition diagram for updating the XML document

IV. Adding an attribute: This process is used to add an attribute to the candidate element.

V. Select candidate attributes for selected elements. In this process any one of the following can be performed:

1- Add (i.e. insert) new attribute before or after the candidate attribute.

2- Delete the candidate attribute.

3- Update the candidate attributes name or value.

5.2.2.4 FrmQuery Form:

Executing XPath queries pass through four stages: validating XPath expression, parsing XPath expression, generating “XPathQuery table”, and building the results in XML tree format. ***Figure 5.5*** shows the state transition diagram for these main processes and the coding of the form is presented in Appendix B.

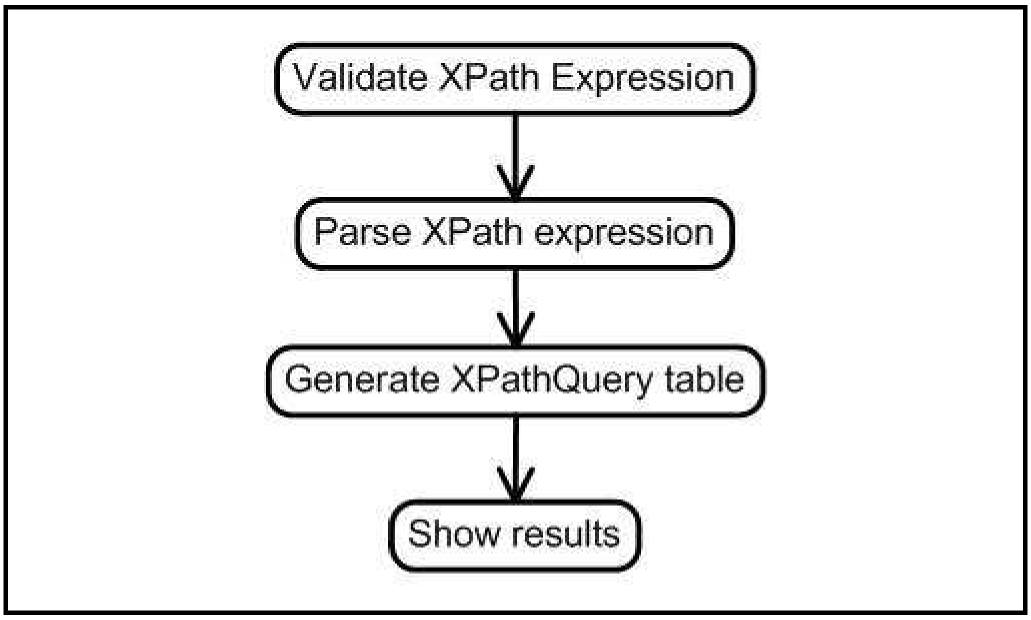


Figure 5.5: Main processes of XPath expression Execution

A brief description of these processes is given below:

I. “*Validate XPath expression*” process is to ensure that the given XPath expression conforms to XPath expression structure rules before parsing it.

II. “*Parsing XPath expression*” process is used to parse and simplify the XPath expression into multiple steps and identify relevant conditions in order to create equivalent SQL statements. These SQL statements will be used to generate the output into a temporary dummy table for the next stage (i.e. cursors alternative).

III. *“Generate XPathQuery table process”* is used to dynamically create result subtree(s) on the fly using the records from the temporary table generated on the previous step.

IV. *“Show results process”* is used to show the query results in two forms, grid view and tree view. In grid view, the results of the query, (i.e. *XPathQuery table* contents) are displayed in tabular format which shows tokenIDs, parentID, token name and token value. While in tree view, the results are shown in a tree-like format representing the XML structure.

5.3. Case Study

In this section, a case study is presented to illustrate the implementation of MAXDOR model. Consider the sample XML document (i.e. *books.xml*) in ***Figure 5.6***. The hierarchical structure of XML document makes it possible to represent it as a rooted, labelled tree. *Figure 5.7* presents an XML tree for the XML document in ***Figure 5.6***. Our approach gives each node a global label in pre-order traversal in the first scan while any new inserted token is given an identification label following the last label used for the document. This label can be taken from *maxTokenID* from *“documents table”*. So the label of a token does not reflect its location in the document structure. Consequently, a label, in our approach, is used to identify a token where each token represents an element or attribute of the XML document. Other researchers use a label to represent the structure of the contents of a document and nodes order (cf. Tatarinov *et al.*, 2002; Torsten et al., 2004; Soltan and Rahgozar, 2006).

After mapping, a single record is assigned for this document in *“documents table”*, for example with *documentID* = 1, as in *Figure 5.8*, and document elements and elements’ attributes are represented as records in the *“tokens* *table"*, as shown in *Figure 5.9*. Each record gives a full description of an element or element's attribute and its structure.

```
<books>
   <book id="bk210" >
      <author id="a1" >M. John</author>
      <title>C++ </title>
   </book>
   <book id="bk211">
      <subject>Math</subject >
      <title> Calculus </title>
      <price> 45.50 </title>
   </book>
</books>
```

Figure 5.6: XML document

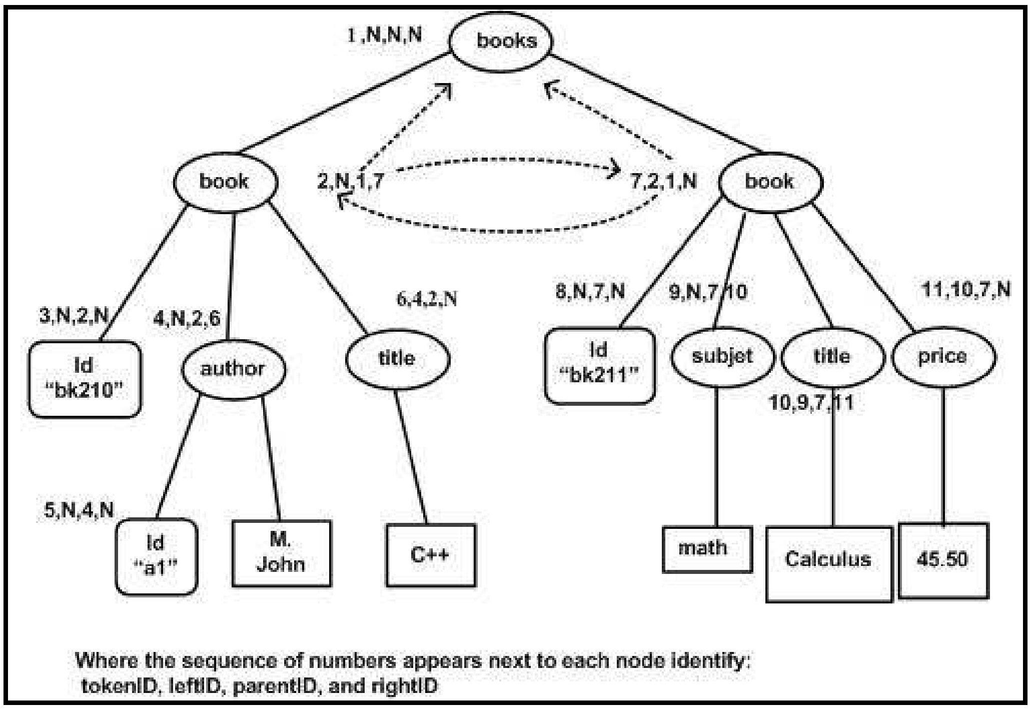


Figure 5.7: A tree representations for XML document

documentID	documentName	docElement	maxTokenID
1	Catalog	Books	11

Figure 5.8: Documents table

After storing XML document content and structure in a relational database, MAXDOR gives the ability to update document contents. Update includes inserting new elements or elements' attributes, deleting elements or attributes, modifying elements' names or values and modifying attributes names or values in a way to keep the document in well formed condition. The update is performed on the relational database version of the document. Thus, there will be no need to keep the original XML document as it does not reflect the contents of the relational database.

documentID	tokenId	leftID	parentID	RightID	prevID	treeLevel	tokenName	tokenValue	tokenType
1	1	0	N	0	0	1	books	Null	1
1	2	0	1	7	1	2	book	Null	1
1	3	0	2	0	2	3	id	bk210	2
1	4	0	2	6	3	3	author	M. John	1
1	5	0	4	0	4	4	id	a1	2
1	6	4	2	0	5	3	title	C++	1
1	7	2	1	0	6	2	book	Null	1
1	8	0	7	0	7	3	id	bk211	2
1	9	0	7	10	8	3	subject	Math	1
1	10	9	7	11	9	3	title	Calculus	1
1	11	10	7	0	10	3	Price	45.50	1

Figure 5.9: Tokens table

Inserting new element: Inserting a new element can be executed in four locations in reference to the selected element; these locations can be as a child, parent, left-sibling or right-sibling. The following discussion shows how to insert new "book" element between the two existing ones, (i.e. before token # 7). *Figure 5.10* shows the XML element "book", *Figure 5.11* reflects the XML modification after inserting the new element. It is a complex element (i.e. subtree) of one attribute and 2 simple elements. Subtree tokens (i.e. elements and attributes) are assigned new IDs that succeed the last assigned label in the previous shredding process for initial mapping or element insertion. For example, the "Book" elements' tokenID becomes 12, and the "book id" elements' tokenID will be 13, the "author" tokenID will be equal to 14, while the "title" elements' tokenID will be equal to 15. *Figure 5.12* shows the equivalent relational tuples for the "book" element and the required updated links for this operation. The right sibling of token number 2 points to the new element which is 12 and the left sibling of node 7 points to the new element which is 12. The left sibling of the new element (i.e. subtree root) points to the element whose TokenID equals to 2 and the right sibling of the new element points to the element whose TokenID equals to 7. PrevID of token 7 is changed to point to the last token in the new subtree which is 15. And prevID of token number 12 points to token number 6. Other tokens' links of the new complex element are shown in *Figure 5.12*.

```
<book id="bk106">
   <author>Mike</author>
   <title>Applied Geometry </title>
</book>
```

Figure 5.10: XML document element (subtree)

The process is trivial for updating selected element’s name or value as this process does not involve updating of document structure. To delete selected element, just update its left and right sibling links and ensure that all its descendant tokens are also deleted.

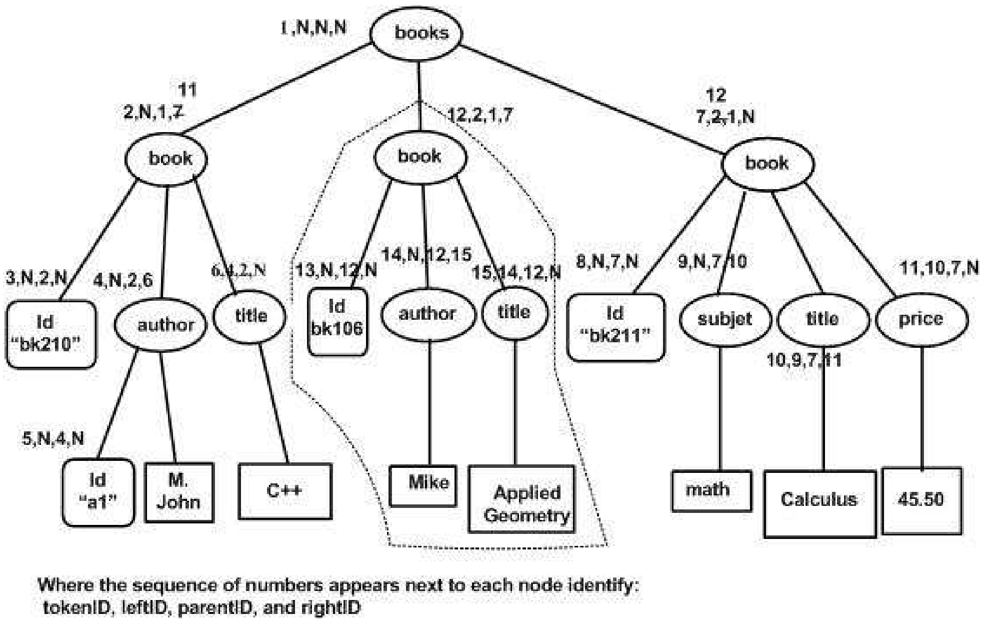


Figure 5.11: A tree representation for updated XML document

documentId	tokenId	leftId	parentId	RightId	prevId	treeLevel	tokenName	tokenValue	tokenType
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
1	2	0	1	12	1	2	book	Null	1
…	…	…	…	…	…	…	…	…	…
1	7	12	1	0	15	2	book	Null	1
…	…	…	…	…	…	…	…	…	…
1	12	2	1	7	6	2	book	Null	1
1	13	0	12	0	12	3	Id	bk106	2
1	14	0	12	15	13	3	Author	Mike	1
1	15	14	12	0	14	3	title	Applied Geometry	1

Figure 5.12: Updated "Tokens table"

5.4. XML Data Sets Used for Testing the Model

In order to assess the usability and efficiency of our MAXDOR model, three XML benchmarks are used: XML benchmark from Washington University (Washington University, 2002), XMark benchmark (Busse et al., 2002) and Michigan XML benchmark (Runapongsa et al., 2006). XML document generator *XMLgen* from XMark is used to create documents of different sizes using factors of the original one.

"Tree-bank" document is taken from Washington benchmark, "Auction documents" from XMark, and "Xbench-TCSD-small" and "Xbench-TCSD-normal" from Michigan benchmark. These documents characteristics are shown in ***Table 5.2***:

Table 5.2: XML data sets of equally sizes

Document	Size(MB)	# of Token	# of Paths	Max depth
Auction11	11	200358	502	12
Xbench-TCSD-small	11	283312	26	8
Auction82	82	1485699	502	12
Tree-bank	82	2437667	168123	36
Auction107	107	1946203	502	12
Xbench-TCSD-Normal	107	2757084	26	8

Michigan XML benchmark data sets are used for evaluating the performance of the model against the complicated characteristics of XML documents such as depth, fan-out in "tree-bank" document. The tree depth has significant effect on performance in cases, of creating and evaluating containment relationships between nodes, namely identifying nodes with ancestor-descendant relations. Nodes fan-out can affect the way in which the DBMS stores data, and affect queries based on retrieving children in precise order, such as the first or last child of a node (Runapongsa *et al.*, 2006). Scaling a benchmark data set in the relational model is done by increasing the number of records. Scaling in XML, however, can be done by increasing depth, number of nodes, or fan-outs. The data sets in ***Table 5.2*** and ***Table 5.3*** are used to evaluate the model performance and usability in both directions, for mapping the documents into relational database and for rebuilding the mapped documents from relational database.

Table 5.3: XML datasets of equal depths and different sizes

Document Name	**Factor used**	**Document Size (MB)**	**Max depth**	**# of nodes**
Auction_1	0.1	11.3	12	206130
Auction_2	0.2	22.8	12	413111
Auction_3	0.3	34.0	12	616229
Auction_4	0.4	45.3	12	820438
Auction_5	0.5	56.2	12	1024073
Auction10	1.0	113.0	12	2048193

For evaluating the update performance of our model, we used th set of documents in ***Table 5.4***. The documents are created from auction document using *XMLgen.* We choose small factor between 0.001 and 0.006 to get small size document that can be managed by our editor. Many experiments can be performed to insert new tokens in different places: In the beginning, in the middle and at the end. They can also have different relationship with the candidate element such as parent, child, left-sibling and right-sibling.

Table 5.4: Auction documents of small factor

Document Name	**Factor used**	**Document Size (KB)**	**# of nodes**
Auction_0.001	0.001	115	2086
Auction_0.002	0.002	210	3684
Auction_0.003	0.003	318	6284
Auction_0.004	0.004	457	7957
Auction_0.005	0.005	567	10492
Auction_0.006	0.006	682	11911

For evaluating the query performance of our model, a set of XPath queries are selected from different resources, ***Table 5.5*** shows those XPath queries and the features which they evaluate.

Table 5.5: XPath expression sets

XPath expression	name	Used for
/root/listing	Q1	Short simple path
/root/listing/auction_info/higher_bidder/bidder_rating	Q2	Long simple path
//higher_bidder/bidder_name	Q3	Regular expression, single '//'
//auction_info//bidder_rating	Q4	Regular expression, double '//'
/root/listing/seller_info[seller_rating='2']	Q5	Text matching
/root/listing[last]	Q6	index
/root/listing/seller_info[seller_rating='2']/ seller_name	Q7	Text matching

5.5. Chapter Summary

In this chapter, a description of the system architecture, and the tools used in building the project are given in section 5.1. These tools include XML tools for generating XML documents, XPath tools for querying and retrieving an XML document or parts of it. RDBMS tools (i.e. Microsoft Access) for storing XML document and SQL for retrieving XPath expression from relational database. To this end, Visual Basic programming language is used as a programming tool.

System implementation description is given in section 5.2. Software and hardware requirements for system implementation have been presented. A description of the classes implemented in Visual Basic for the four main components of the project is offered. *XbsXml2Base* Class is used for mapping XML document into relational database. *XbsBase2XML* Class is used for rebuilding XML document from relational database. *DbxTokens* Class is used for editing XML document contents within a relational database. That includes update, insert or delete of document element's name, element's value, attribute's name or value. "*fmrquery*" form is used for parsing XPath expression, formulating of equivalent SQL statement, getting the results and building it in XML tree format.

Section 5.3 presents theory implementation on a sample case study which shows the process of mapping an XML document into relational database and the process of how to update the XML document within the relational database.

Section 5.4 shows different XML data sets from various XML benchmarks and XPath expression sets for testing and evaluating the usability and performance of MAXDOR model.

The experiments and their resultant assessments will be given in the next chapter, Chapter 6.

CHAPTER 6 EXPERIMENTS AND THEIR ASSESSMENT

In this chapter we will give a description of the experiment setup consisting of experiment environment and performance measurement. We will perform experiments on mapping XML document into relational database, building XML document from relational database, updating XML document stored in relational database and retrieving document content from relational database using XPath expressions. These experiments will be done to check the scalability and effectiveness of our model. Then we will compare our model with the Global Encoding model (Tatarinov *et al.*, 2002) and the Accelerating XPath model (Torsten *et al.*, 2004). The comparison consist of four stages: mapping, building, updating and retrieving, as most of other studies just took one or two stage and forgot the others. Some of them took retrieving, others took updating and others took updating and retrieving, but most of them did not consider mapping and rebuilding.

6.1. Experiment Setup

6.1.1 Experiment Environment

All experiments tests are conducted on a PC of an Intel Core2 Quad Q9550 2.83 GHz CPU, 4.00 GB RAM, running Windows 7 Professional. Visual Basic 6 programming language is used to implement MAXDOR model, and Microsoft Access 2007 is used as a target relational database for storing XML document contents on local hard drive. In addition, a disk file is named with document number in the document table and with an XML extension created for reconstructed XML document from relational database.

6.1.2 Performance Measurement

- Mapping XML document into RDB execution time.
- Rebuilding of XML document from RDB execution time.
- Dealing with any document size.
- Inserting nodes processing time (number of nodes to be relabelled).
- Query processing execution time.

The execution time is used as an evaluation scale in this research rather than storage space since the former is crucial nowadays for the users, while storage space is available in a very huge size with reasonable prices.

6.2. Testing Strategies

6.2.1 Mapping XML Document into Relational Database Performance.

The experiment is performed as follows:

Face 1, **scalability test:** An XML document generator from XMark (Busse *et al.*, 2002) is used to create documents of different sizes with factors of 0.1, 0.2, 0.3, 0.4 and 0.5. The documents characteristics are shown in ***Table 6.1***. In this experiment, our model shows performance in a linear and scalable manner as document size is increasing. The mapping result over different sizes of the same document is shown in ***Figure 6.1***.

Table 6.1: Different sizes of Auction document

Document Name	Factor used	Document Size (MB)	# of nodes
Auction_1	0.1	11.3	206130
Auction_2	0.2	22.8	413111
Auction_3	0.3	34.0	616229
Auction_4	0.4	45.3	820438
Auction_5	0.5	56.2	1024073

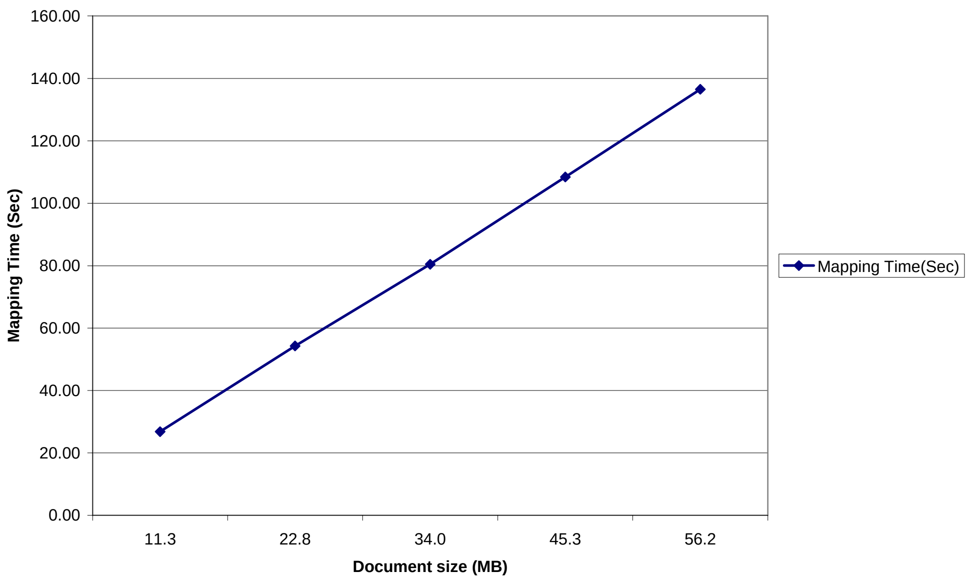


Figure 6.1: Mapping time for dataset in Table 6.1

Face 2, effectiveness test: Three groups of documents of different sizes 11MB, 82MB and 107MB but with different structure and different numbers of token are included in this experiment. ***Table 6.2*** shows documents properties and their mapping and rebuilding time. ***Figure 6.2*** shows the time required for mapping XML documents into relational database which consistently increases as the number of tokens increases in the document.

Considering the results shown in ***Figure 6.1*** for homogenous documents and those shown in ***Table 6.2*** and ***Figure 6.2*** for heterogeneous documents coupled with calculating the correlation coefficient between document size and mapping time in the two cases r_1=0.99988 and r_2=0.8751 on the one hand, and the number of tokens and mapping time in the two cases r_3= 0.99991 and r_4=0.9991 on the other hand, we can conclude that the time required for mapping the document largely depends on the number of tokens (i.e., elements and attributes) in the document, the document size and document depth (r=0.1752) respectively.

Table 6.2: XML Dataset of different structures

Document	Doc Size (MB)	# of Token	# of XPath	Mapping (Sec)	Building (Sec)
Auction11	11	200358	502	25.50	13.41
Xbench-TCSD-small	11	283312	26	36.75881	17.39469
Auction82	82	1485699	502	186.7157	141
Tree-bank	82	2437667	168123	325.2331	150
Auction107	107	1946203	502	260.3572	200
Xbench-TCSD-Normal	107	2757084	26	376.7195	181

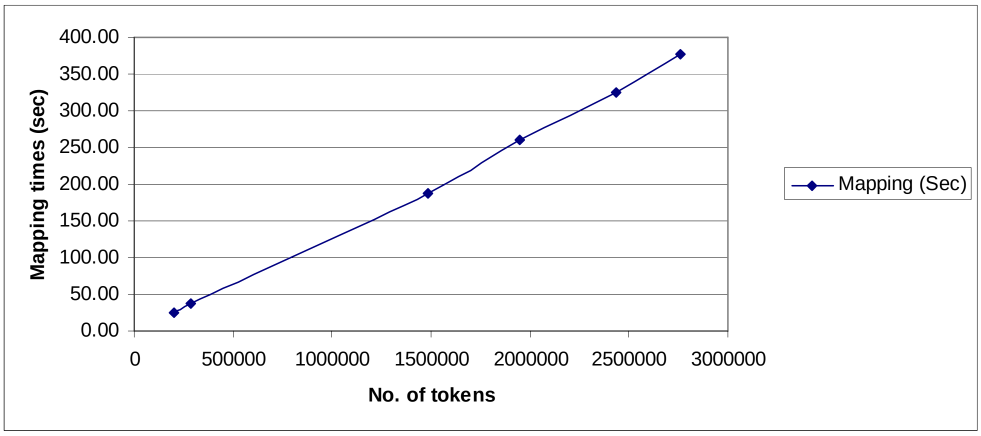


Figure 6.2: Mapping time for documents in Table 6.2

Now let us compare MAXDOR model with Global Encoding for (Tatarinov *et al.*, 2002) and Accelerating XPath for (Torsten *et al.*, 2004), since the three models are using the same general number encoding to identify the XML document of elements and attributes (tokens). A detail description for Global Encoding and Accelerating XPath is given in Chapter 3.

The three models use one scan to shred the document contents, assign an identifier for each token, reserve node information, (i.e. token name and token value) to store them in one tuple in relational database. Global Encoding adds another table for tokens path from the document element passing through until the candidate token. MAXDOR and Accelerating XPath are similar in using just one table to store documents contents. Both also use a stack collection to manage post-order label in Acceleration XPath and RightID link in MAXDOR.

Based on previous experiment, one finds that mapping time mainly depends on the number of tokens in the document. Based on that, we may consider the following assumptions:

$$T=\begin{cases} t & \textit{for both MAXDOR and Accelerating XPath,} \\ t+tp & \textit{for Global encoding,} \end{cases} \qquad (6.1)$$

where T is the mapping time and tp is the time required to process the tokens path.

$$tp = (t/n)*m \qquad (6.2)$$

where n is the number of tokens in the document and m is the number of distinct paths in the document.

Now we can use the results of experiments 1 and 2 for mapping XML documents into relational database and compare our model with the other two models.

From ***Table 6.3*** and ***Figure 6.3*** we can see that MAXDOR and Accelerating XPath are identical while Global Encoding is closed to the other two models in homogeneous documents where the number of paths is small and the gap becomes larger for heterogeneous documents where the number of paths becomes very large as in tree_bank document.

Table 6.3: Mapping time for MAXDOR, Accelerating XPath and Global Encoding in seconds

Doc. Size (MB)	**# of Token**	**Different path**	**M-MAXDOR**	**M-Accel**	**M-Global**
11	200358	502	25.4965	25.4965	25.56038
11	283312	26	36.7588	36.7588	36.76218
82	1485699	502	186.7157	186.7157	186.77879
107	1946203	502	260.3572	260.3572	260.42436
82	2437667	168123	325.2331	325.2331	347.66404
107	2757084	26	376.7195	376.7195	376.72305

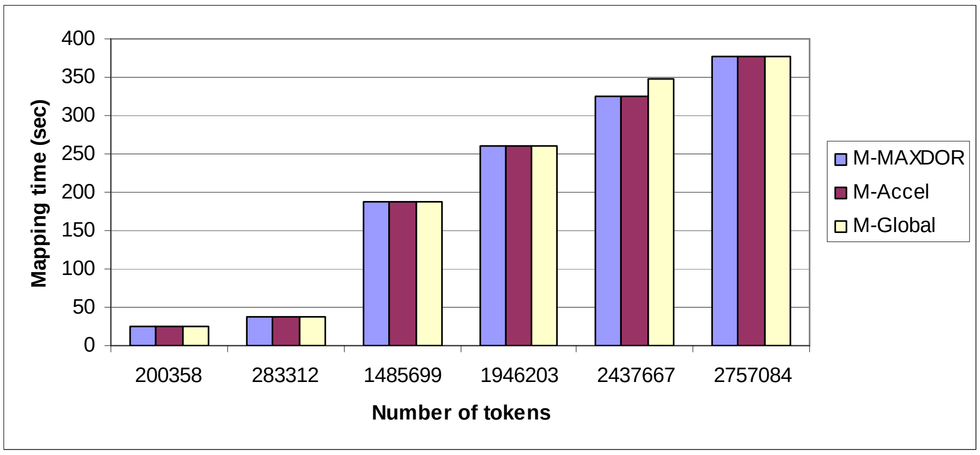


Figure 6.3: Mapping Comparison between MAXDOR, Accelerating XPath and Global Encoding

6.2.2 Rebuilding XML Document from Relational Database Performance

The experiment is done at different stages as follows:

Face 1, scalability test: the auction documents in ***Table 6.1*** mapped before will be built in this experiment to see the scalability of MAXDOR in rebuilding XML documents from relational database.

From the results shown in ***Figure 6.4***, we find that our model performs well for rebuilding the XML document. The time for rebuilding a document of 11.3MB size is 14.14 seconds and for 56.2MB size is 88.00 seconds. This shows that the relation between rebuilding time and document size is approximately linear as it passes through the origin and is given as follows:

$$t = 1.644989\ s \qquad (6.3)$$

where t is the time in seconds for rebuilding the document and s is the size of the document in MB.

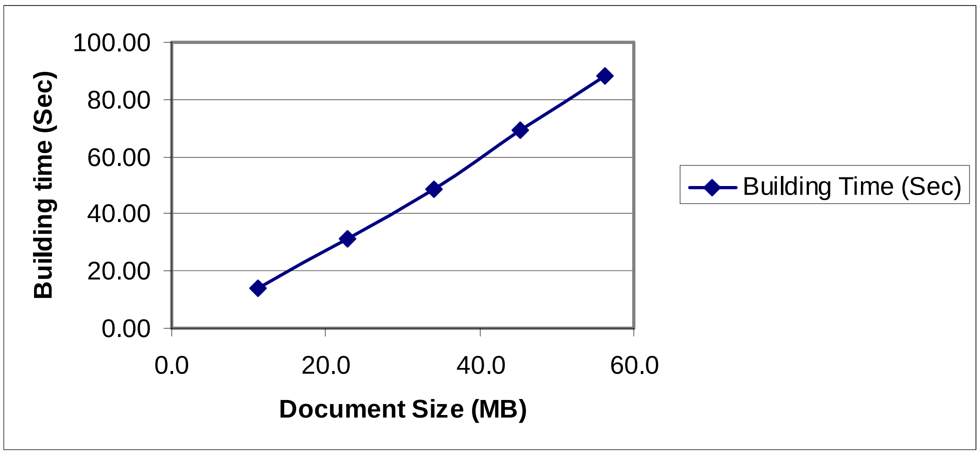


Figure 6.4: Building time for documents in Table 6.1

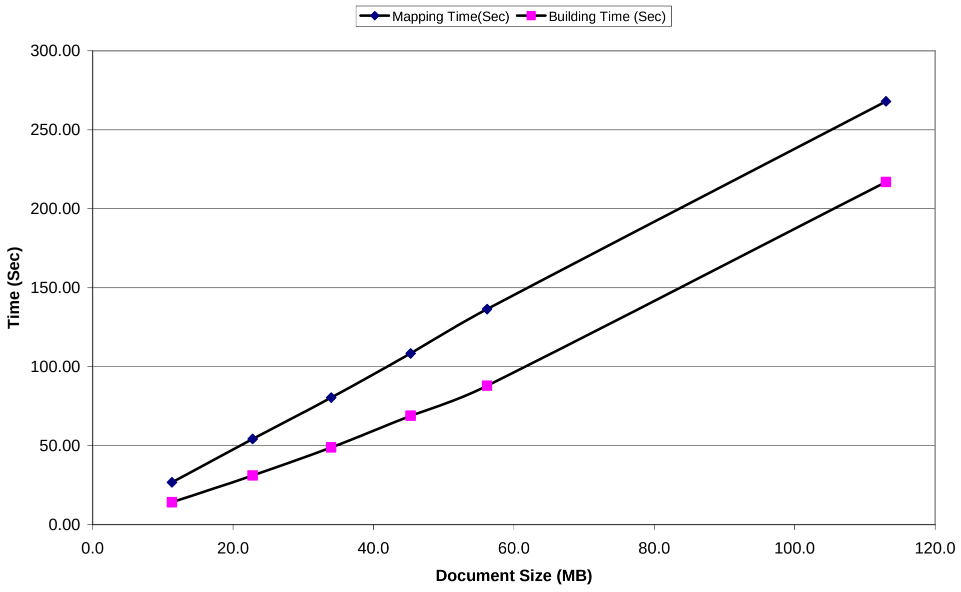


Figure 6.5: Mapping and building time for XML documents of different sizes

Figure 6.5 is a combination of ***Figure 6.1*** and ***Figure 6.4*** for mapping and rebuilding of the same XML documents, in addition to an extra document which is the original auction document of 113.0MB. From the Figure we can conclude that our model still behaves linearly for both mapping and rebuilding of large sizes of documents.

Face 2, effectiveness test: The same sets of documents from ***Table 6.2*** are also used to check the ability of MAXDOR in dealing with different XML document types. The documents are grouped by size and every two have the same size.

From the experiments done and results shown in ***Figure 6.6***, it can be concluded that the time of rebuilding the document is influenced by the number of tokens formulating the document because two documents of the same size need different amounts of time for rebuilding.

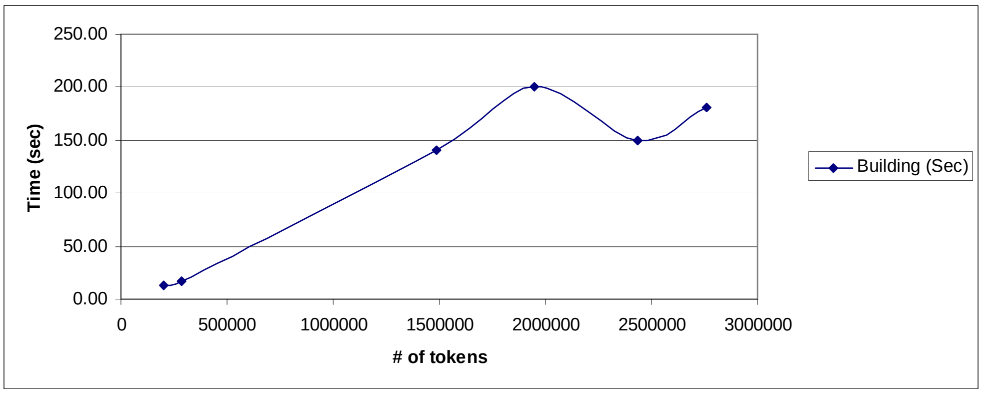


Figure 6.6: Building time for documents in Table 6.2

From the results shown in ***Figure 6.4*** for homogenous documents and results shown in ***Table 6.2*** and ***Figure 6.6*** for heterogeneous document, and after calculating the correlation coefficient between document size and rebuilding time (r1= 0.998795203, r2= 0.926455747), and number of tokens and rebuilding time (r3= 0.999311324, r4= 0.308485455), we can conclude that the time required for rebuilding the document mainly depends on the document size, the number of tokens (elements and attributes) that exist in the document and the document depth (r= 0.214860654) respectively.

Face 4, Building XML document after the insertion of elements in three locations:

1. At the beginning of the document.

2. In the middle of the document

3. At the end of the document.

Table 6.4 and ***Table 6.5*** show results of rebuilding auction document of several values of *n*, where *n* is the number of tokens in the document. In ***Table 6.4***, column 3 shows the time required for rebuilding the documents before any update, column 4 after inserting a token at the beginning of the documents, column 5 after inserting a new token at the middle and column 6 after inserting a new token at the end of the documents. ***Table 6.5*** shows the difference between the required time for rebuilding the document after inserting the defined location and the rebuilding time of the original document and the percentages of that difference.

The averages of percentages are different. The cost of rebuilding the document depends mainly on the location of inserting the new tokens. The cost decreases from $1.24*t$ at location L_1 to t at location L_3, where L_1 denotes token number 2 and L_3 denotes token number $n + 1$, and t represents the time required for rebuilding the original document before any insertion.

Next, we will compare our model with the models of Tatarinov *et al.* (2002) and Torsten *et al.* (2004). The comparison will be based on the rebuilding document cost in time (*BCDT*) and the time of inserting a new token (element or attribute) (*ICDT*). The comparison will make use of the discussion above. In the following results, we will give the expected value of the *BCDT* and *ICDT* for the models under study.

Theorem-6.1:

(a) Under the following assumptions:

1- We will assume that the locations of insertion have the same probability,

$$P[X = x] = 1/n, \ x = 2, 3, \dots n+1 \qquad (6.4)$$

where X denote the location of insertion.

2- We will assume that the time decreases from $1.24*t$ at location 2 to t at location $n+1$ uniformly, i.e.

$$P[Y= 1.24 - [0.24*(y-2)/(n-1)]*t] = 1/n, y= 2,3 \dots n+1 \qquad (6.5)$$

where Y denotes the time required to build the document after inserting a new token at position y, we have:

$$E_{11}=E_{MAXDOR}[BCDT] = 1.24*t - 0.12*t\,(n-1)/n \qquad (6.6)$$

$$\text{(b) } E_{12} = E_{Blobal}[BCDT] = t \qquad (6.7)$$

$$\text{(c) } E_{13} = E_{Acc}[BCDT] = t \qquad (6.8)$$

where E_{model} denotes the expected value of $BCDT$ under the model. The Proof of Theorem 6.1 (a) will be given in Appendix A.

For E_{12} and E_{13}, in both cases the tokens there are sorted in sequential order and the time needed for building the document is equal to t

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in ***Table 6.4*** and ***Table 6.5***.

Table 6.4: Building time after update

Document Size (KB)	# of Tokens	Before insertion	Insertion Location		
			Begin	Middle	End
115	2086	0.1256	0.1598	0.1384	0.12623
210	3684	0.2264	0.2759	0.2474	0.22581
318	6284	0.3854	0.4799	0.4341	0.37913
457	7957	0.4963	0.6134	0.5671	0.49238
567	10492	0.6419	0.8116	0.7307	0.64538
682	11911	0.7295	0.8924	0.8245	0.73666

Table 6.5: Differences in building time

Document Size (KB)	Differences			Percent		
	Begin L1	Middle L2	End L3	Begin	Middle	End
115	0.03425	0.01281	0.00067	27%	10%	1%
210	0.04950	0.02100	-0.00056	22%	9%	0%
318	0.09450	0.04869	-0.00631	25%	13%	-2%
457	0.11719	0.07088	-0.00388	24%	14%	-1%
567	0.16969	0.08875	0.00344	26%	14%	1%
682	0.16291	0.09497	0.00716	22%	13%	1%

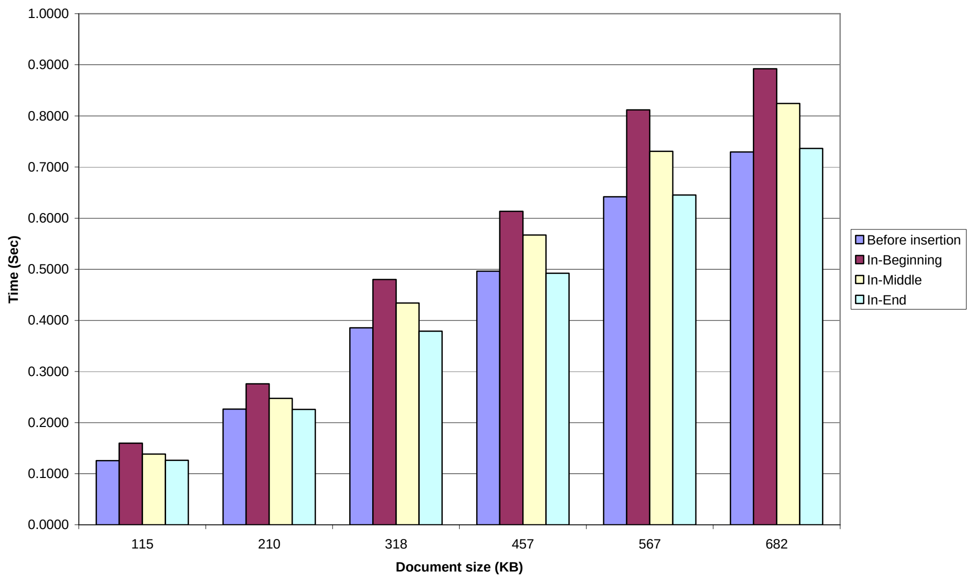


Figure 6.7: Comparison of building after insertion in different location

6.2.3 Updating Performance

To evaluate our model updating performance, the experiment is performed as follows:

a. Inserting a child node in different location in the document and at different levels.

b. Inserting a preceding-sibling (i.e. before) node in different locations in the document and at different levels.

c. Inserting a following-sibling (i.e. after) node in different locations in the document and at different levels.

d. Inserting a parent node in different location in the document and for different levels.

Table 6.6 shows the time in seconds needed to process the inserting nodes in documents that have 2086, 3684, 6284 tokens. The figures in the table show that the number of tokens (i.e. size of the document) has an influence on the processing time wherever the insert on process occurs, in the beginning of the document, in the middle or at the end. For cases of inserting a token as a child or before (i.e. left-sibling), the time cost is constant, but for the other two cases, parent and after (i.e. right-sibling), the cost is variable. For the parent node since we have an identifier for token level in the document, all descendant nodes tree level should be updated (i.e. incremented by 1). While for after nodes (right-sibling) we should look at descendant nodes for the proper PrevID link for the new node. That means, there is an increase in the cost of insertion time depending on the size of the candidate node (i.e. number of descendant nodes) for the two cases. The differences in cost for *parent* and *after* tokens are shown in ***Table 6.6***.

Table 6.6: Time cost of insertion of a token in different location

Location in Document	**Insert Location (time in Sec)**				# of token in Document
	Parent	**Child**	**Before**	**After**	
In-Beginning	0.046875	0.015625	0.015625	0.015625	2086
At-Middle	0.015625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.078125	
In-Beginning	0.0625	0.015625	0.015625	0.03125	3684
At-Middle	0.015625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.015625	
In-Beginning	0.046875	0.015625	0.015625	0.03125	6284
At-Middle	0.0625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.015625	

Theorem-6.2:

(a) Under the following assumptions:

1- We will assume that the locations of insertion have the same probability,

$$P[X = x] = 1/n, x = 2, 3 \ldots n+1 \qquad (6.9)$$

where X denotes the location of insertion.

2- We will assume that the time decreases from $n{*}t_0$ at location 2 to t_0 at location $n+1$ uniformly, i.e.

$$P[Z= t_0 [n - z + 2] = 1/n, y= 2,3 \ldots n+1, \qquad (6.10)$$

where Z denotes the time required to insert the new node at position z, we have:

$$E_{22} = E_{Global}[ICDT] = t_1 n/2 + t_0/(n+1) \qquad (6.11)$$

$$\text{(b) } E_{21}=E_{MAXDOR}[ICDT] = t_0 \qquad (6.12)$$

$$\text{(c) } E_{23} = E_{Acc}[ICDT] = t_1 n + t_0/(n+1) \qquad (6.13)$$

where E_{model} denotes the expected value of $ICDT$ under the model:

Proof of Theorem 6.2 (a) will be given in Appendix A.

b) For E_{21}, since there is no relabeling needed after insertion of a new token. Then, the cost of inserting a new node is equal to $t_{0.}$

c) For E_{23}, since there is a need to update the pre-order and post-order label, the cost of update will be double the cost of update one of label after insertion of a new token.

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in ***Table 6.4*** and ***Table 6.5***.

6.2.4 Query Performance

To evaluate the query performance of our model, we execute the following XPath expressions against the stored XML document in relational database. After that, we will compare the results with the other two models, Global Encoding and Accelerating XPath. To make sure that our experiments run in reproducable form, we create different sizes of XML documents from auction document using the generator *XMLgen* from XMark benchmark (Busse *et al.*, 2002). ***Table 6.7*** shows these documents and their characteristics.

For each XPath expression in ***Table 6.8***, we run the experiment for each document in ***Table 6.7***.

Table 6.7: XML documents sizes and # of tokens in them

Document size (MB)	**Number of Nodes**	**Factor value**
0.11	2086	0.001
0.22	3684	0.002
0.44	7956	0.004
0.55	10492	0.005
0.66	11911	0.06
1.1	21051	0.01
11.0	200358	0.1

Table 6.8: XPath expressions under evaluation

XPath expression	**name**
/site/regions	Q1
/site/regions/Africa/item/location	Q2
/site/regions/Africa/item[@id="item1"]/location	Q3

Table 6.9: XPath traversals for query Q1

Document size (MB)	# Result Nodes	# of interest result	t(ms)
0.11	2	1	7.8125
0.22	2	1	4.882813
0.44	2	1	6.835938
0.55	2	1	7.8125
0.66	2	1	5.859375
1.1	2	1	7.8125
11.0	2	1	7.8125

Table 6.10: XPath traversals for query Q2

Document size (MB)	# Result Nodes	# of interest result	t(ms)
0.11	5	1	7.8125
0.22	5	1	11.23047
0.44	7	2	11.23047
0.55	7	2	7.8125
0.66	9	3	11.23047
1.1	13	5	11.23047
11.0	113	55	15.625

Table 6.11: XPath traversals for query Q3

Document size (MB)	# Result Nodes	# of interest result	t(ms)
0.11	5	1	23.4375
0.22	5	1	31.73828
0.44	5	1	38.08594
0.55	5	1	46.875
0.66	5	1	45.89844
1.1	5	1	70.3125
11.0	5	1	60.15625

For Q1, we can see that the execution time is almost the same, since there are just two select statements to get the desired results of one token. For Q2, we can see from the ***Table 6.10***, there is a difference between the number of selected nodes and the number of interest nodes. This difference becomes as a result of selecting the ancestors on the desired result, and the cost will become high for large homogeneous documents. For Q3, the execution time increases as the document size increases, since there is more time needed to execute the condition.

6.3. Model Analysis and Comparison

We will compare the models, MAXDOR, Global encoding and Accelerating XPath using the total expectations of the cost of building the document (*BCDT*) and the cost of insertion of a new token (*ICDT*) (whose expression are given in Theorems 6.1 and Theorem 6.2) as follows:

$$E_1 = E_{11} + E_{21} = 1.24\, t - 0.12\, t\, (n-1)/n + t_0 \tag{6.14}$$

$$E_2 = E_{12} + E_{22} = t + (t_1\, n/2 + t_0/(n+1)) \tag{6.15}$$

$$E_3 = E_{13} + E_{23} = t + (t_1\, n + t_0/(n+1)) \tag{6.16}$$

Where t denotes the time in seconds required for building the document, t_0 denotes the time in seconds required for inserting the new token and t_1 denotes the time required to update the label.

In ***Table 6.12***, we calculated the total expectation time for building XML documents from relational database and for inserting new tokens in different positions in the document with probability *1/n*, where *n* is the number of tokens in the document. t_0 is the time required to insert a new token, t_1 is the time required to update the label and t is the time required to build the document, E_1, E_2 and E_3 which is the total expectation time for MAXDOR, Global Encoding and Accelerating XPath respectively.

Table 6.12: Total expectation time for building and inserting tokens for the three models (in Sec)

n	t_0	t	t_1	E_1	E_2	E_3
2086	0.015625	0.1256	0.00488	0.15882	5.21734	10.30907
3684	0.015625	0.2264	0.00488	0.27373	9.21870	18.21099
6284	0.015625	0.3854	0.00488	0.45499	15.72405	31.06270
7957	0.015625	0.4963	0.00488	0.58142	19.91858	39.34086
10492	0.015625	0.6419	0.00488	0.74740	26.25188	51.86185
11911	0.015625	0.7295	0.00488	0.84726	29.80312	58.87674

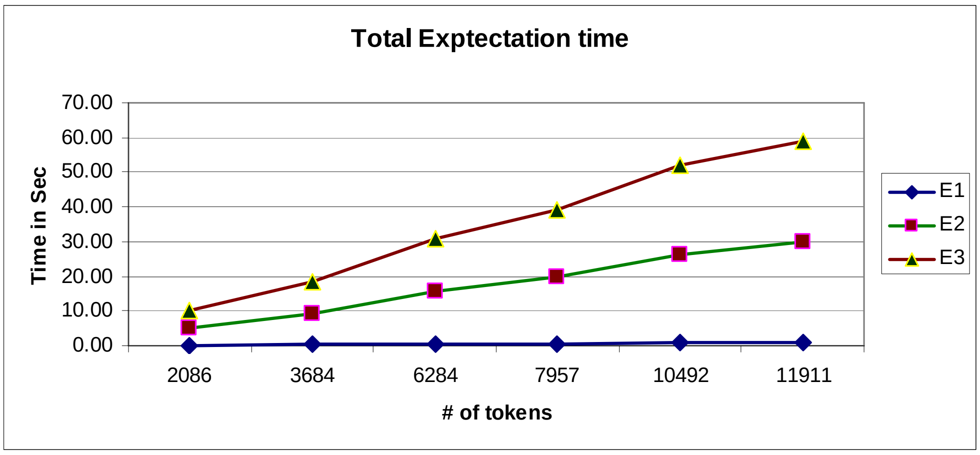
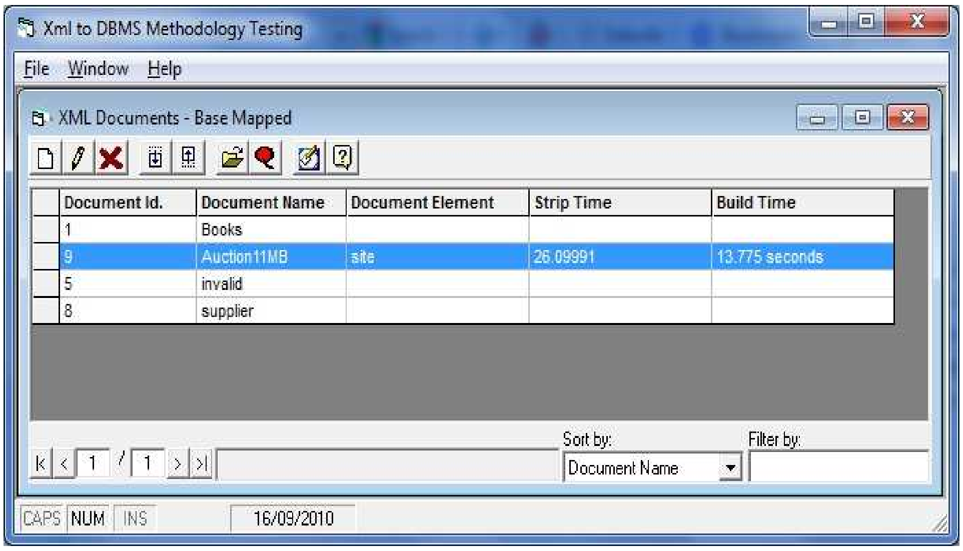


Figure 6.8: Total expectation time for the three models, MAXDOR, Global Encoding, and Accelerating XPath

From ***Table 6.12*** and ***Figure 6.8*** we can see that our model MAXDOR out perform the two models for the total expectation time. And the difference becomes large for a large number of tokens n.

In the following figures, snapshots for some run of MAXDOR system are shown. ***Figure 6.9*** shows snapshot for a run to map and rebuild Auction XML document of size 11MB. The time in seconds for mapping and rebuilding is also shown.



Xml to DBMS Methodology Testing

XML Documents - Base Mapped

	Document Id.	Document Name	Document Element	Strip Time	Build Time
	1	Books			
	9	Auction11MB	site	26.09991	13.775 seconds
	5	invalid			
	8	supplier			

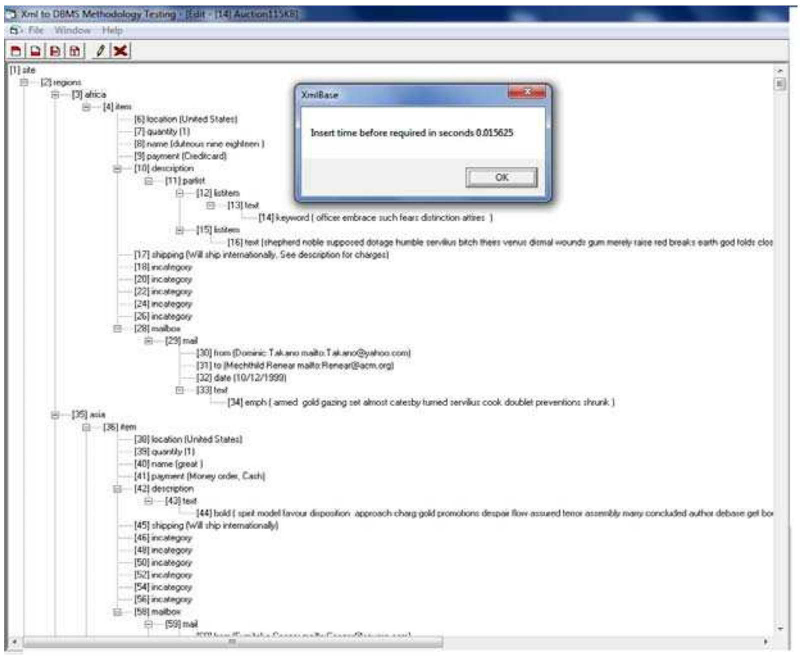
Sort by: Document Name

Filter by:

16/09/2010

Figure 6.9: Snapshot for mapping and building of XML document

Figure 6.10 and ***Figure 6.11*** show snapshots for inserting new element before and after element “africa” in the auction document respectively. The time required for both processes is displayed as messages on the screen.

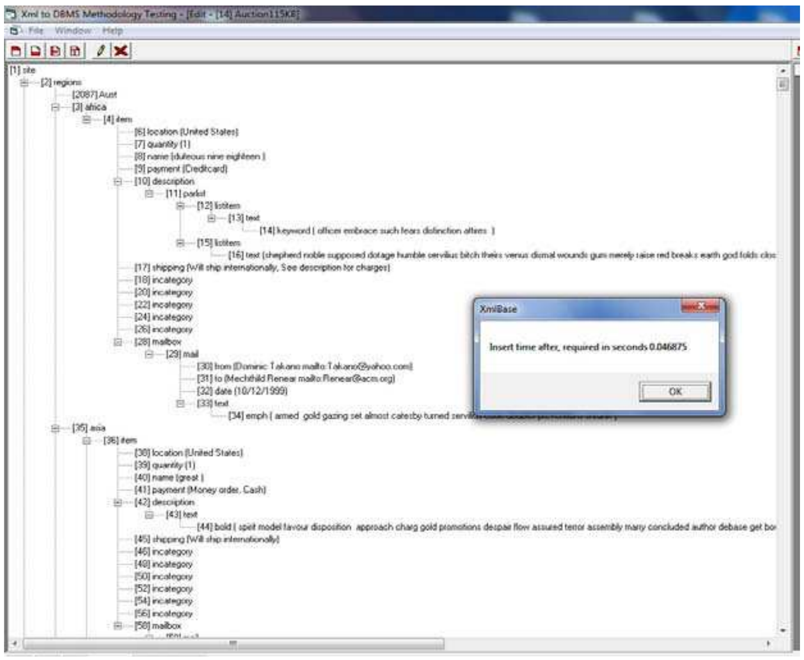


```
Xml to DBMS Methodology Testing - [Edit - [14] Auction115KB]
[1] site
  [2] regions
    [3] africa
      [4] item
        [6] location (United States)
        [7] quantity (1)
        [8] name (duteous nine eighteen )
        [9] payment (Creditcard)
        [10] description
          [11] parlist
            [12] listitem
              [13] text
                [14] keyword ( officer embrace such fears distinction attires  )
            [15] listitem
              [16] text (shepherd noble supposed dotage humble servilius bitch theirs venus dismal wounds gum merely raise red breaks earth god folds clos
        [17] shipping (Will ship internationally, See description for charges)
        [18] incategory
        [20] incategory
        [22] incategory
        [24] incategory
        [26] incategory
        [28] mailbox
          [29] mail
            [30] from (Dominic Takano mailto:Takano@yahoo.com)
            [31] to (Mechthild Renear mailto:Renear@acm.org)
            [32] date (10/12/1999)
            [33] text
              [34] emph ( armed  gold gazing set almost catesby turned servilius cook doublet preventions shrunk )
    [35] asia
      [36] item
        [38] location (United States)
        [39] quantity (1)
        [40] name (great )
        [41] payment (Money order, Cash)
        [42] description
          [43] text
            [44] bold ( spirit model favour disposition  approach charg gold promotions despair flow assured terror assembly many concluded author debase get bo
        [45] shipping (Will ship internationally)
        [46] incategory
        [48] incategory
        [50] incategory
        [52] incategory
        [54] incategory
        [56] incategory
        [58] mailbox
          [59] mail
```

XmlBase

Insert time before required in seconds 0.015625

Figure 6.10: Snapshot for inserting new element before candidate one


Xml to DBMS Methodology Testing - [Edit - [14] Auction115KB]
File
Window
Help

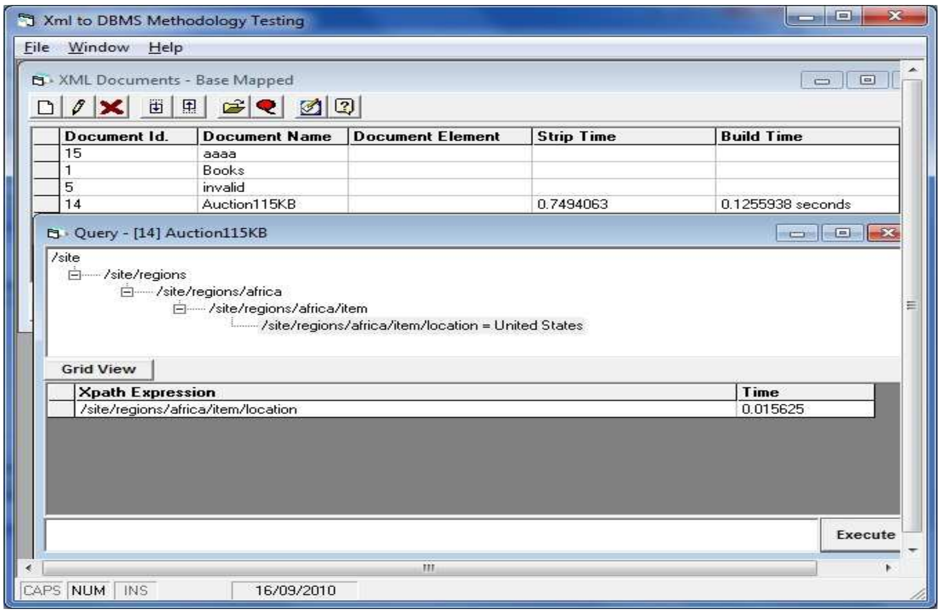
```
[1] site
  [2] regions
      [2087] Aust
      [3] africa
          [4] item
              [6] location (United States)
              [7] quantity (1)
              [8] name (duteous nine eighteen )
              [9] payment (Creditcard)
              [10] description
                  [11] parlist
                      [12] listitem
                          [13] text
                              [14] keyword ( officer embrace such fears distinction attires )
                      [15] listitem
                          [16] text (shepherd noble supposed dotage humble servilius bitch theirs venus dismal wounds gum merely raise red breaks earth god folds clo
              [17] shipping (Will ship internationally, See description for charges)
              [18] incategory
              [20] incategory
              [22] incategory
              [24] incategory
              [26] incategory
              [28] mailbox
                  [29] mail
                      [30] from (Dominic Takano mailto:Takano@yahoo.com)
                      [31] to (Mechthild Renear mailto:Renear@acm.org)
                      [32] date (10/12/1999)
                      [33] text
                          [34] emph ( armed  gold gazing set almost catesby turned servi
      [35] asia
          [36] item
              [38] location (United States)
              [39] quantity (1)
              [40] name (great )
              [41] payment (Money order, Cash)
              [42] description
                  [43] text
                      [44] bold ( spirit model favour disposition  approach charg gold promotions despair flow assured terror assembly many concluded author debase get bo
              [45] shipping (Will ship internationally)
              [46] incategory
              [48] incategory
              [50] incategory
              [52] incategory
              [54] incategory
              [56] incategory
              [58] mailbox
```

XmlBase

Insert time after, required in seconds 0.046875

Figure 6.11: Snapshot for inserting new element after candidate one

Figure 6.12 and ***Figure 6.13*** Show snapshots for an execution of an XPath expression (q2) against the auction document. The figures show the results in tree view and grid view respectively.


Xml to DBMS Methodology Testing
File
Window
Help
XML Documents - Base Mapped

Document Id.	Document Name	Document Element	Strip Time	Build Time
15	aaaa			
1	Books			
5	invalid			
14	Auction115KB		0.7494063	0.1255938 seconds

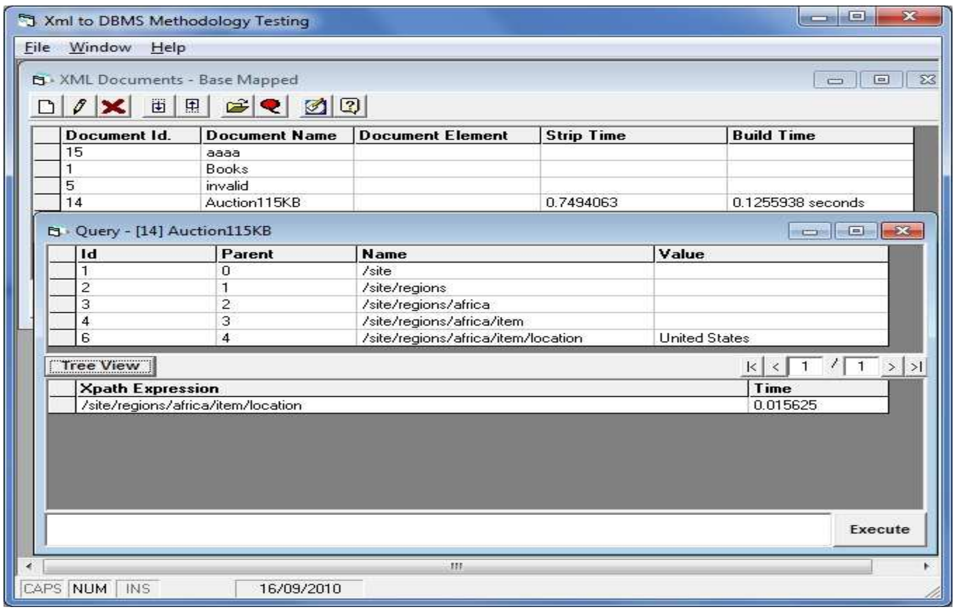
Query - [14] Auction115KB

```
/site
  /site/regions
      /site/regions/africa
          /site/regions/africa/item
              /site/regions/africa/item/location = United States
```

Xpath Expression	Time
/site/regions/africa/item/location	0.015625

Execute
CAPS
NUM
INS
16/09/2010

Figure 6.12: Snapshot for executing XPath in tree view


Xml to DBMS Methodology Testing
File Window Help
XML Documents - Base Mapped
Query - [14] Auction115KB
Tree View
Execute
CAPS NUM INS 16/09/2010

Document Id.	Document Name	Document Element	Strip Time	Build Time
15	aaaa			
1	Books			
5	invalid			
14	Auction115KB		0.7494063	0.1255938 seconds

Id	Parent	Name	Value
1	0	/site	
2	1	/site/regions	
3	2	/site/regions/africa	
4	3	/site/regions/africa/item	
6	4	/site/regions/africa/item/location	United States

Xpath Expression	Time
/site/regions/africa/item/location	0.015625

Figure 6.13: Snapshot for executing XPath in tree view

CHAPTER 7 : CONCLUSIONS AND FURTHER RESEARCH

In this thesis, we have characterized a new model for mapping XML documents into relational database. The model examined the problem of solving the structural hole between ordered hierarchical XML and unordered tabular relational database to enable us to use the relational database systems for storing, updating and querying XML data. We have introduced and implemented a mapping system called MAXDOR to solve the problem.

7.1 Contributions

The following are the main contributions presented throughout this thesis:

XML Document mapping into relational database: a novel method is introduced to partition XML document into tokens (i.e. element and attributes). It relies on assigning a tuple in a relational table for each token information and relations with its neighbours. The method works efficient and performs well for large XML documents.

Building XML document from relational database: a novel method is introduced to build original XML document or update one from relational database. It relies on retrieving document contents depending on token links and token levels which formulate XML document as a group of subtrees.

Updating XML document contents: a novel method is used to update (i.e. insert new token or modify its name or value) XML document contents stored in relational database. It is based on creating links for each token with its neighbours to maintain document structure without a need to relabel or re-index document contents.

Querying and retrieving many XPath axes of XML document: a novel method is introduced to access most of XPath axes preceding-sibling, following-sibling and descendant without storing all possible XPath information for document contents (Tatarinov et al., 2002; O'Neil et al., 2004) . It relies on dynamically creating result subtree(s) on the fly using a temporary table "XPathQuery table" for the desired XPath expression storing all interested tokens.

7.2 Advantages

- **High Flexibility of updating:** MAXDOR approach performed updating processes of inserting new tokens in any location in the document and at any level of relevance to the candidate element (i.e. parent, child, left-sibling and right-sibling), updating token name and value at constant cost of execution time since there is no need to relabel following tokens IDs or overwrite tokens paths.
- **Stability:** The approach worked fine in both directions; mapping and rebuilding for large documents: *"Auction"* document with 600MB size and 9244050 tokens can be processed without trouble.

7.3 Recommendations:

1. Our model is strongly recommended for a system where XML document contents needs to be updated very frequently.
2. Our model is strongly recommended for a system where maintaining document structure is important as in document-centric documents.

7.4 Drawbacks and Limitations

- Loss of Information: Our mapping algorithm does not consider some information in the original XML document such as processing instructions, comments, CDATA sections and external entities. Furthermore, it needs an enhancement to consider multiple occurrences of texts in one element.

- Since XPath query expression is used for retrieving information from XML document, it ascribes two limitations to our approach:
    1. Only one query upon one document will be applied at the time.
    2. XPath language doesn't have commands to insert or update an XML document content that enforces us to add an editor to manage updating process. The editor can manage small documents only.
- Our approach uses fixed schema in relational database and one table "tokens table" is used to store document contents. In addition, maximum table size in Microsoft Access is limited to 2GB including System Objects and indexes. These limitations restrict the maximum XML document size to be mapped in our approach to 600MB approximately

7.5 Further Research

There is still room enough for improvement. This includes:

- Enhancing our document editor to manage large XML documents.
- Conducting further study on XPath parser in order to evaluate our model for the querying and retrieving parts since it is not finalized yet.
- Using of XQuery Language for the retrieving and updating contents of XML documents.
- Using MSSQL, MYSQL or Oracle as an alternative to Microsoft Access to solve the problem of maximum document size of around

550M to achieve faster response in building XML documents on the fly XPath queries using DBMS memory cursors.

- Since multiple links are used in our model, an optimization of labels sizes may reduce the size of "Tokens Table" and indexes used for these links.
- Other performance measurement for evaluation needs to be considered such as storage space and mapping accuracy.
- Ancestor-descendant relationship is executed indirectly through multi parent-child relationship. This increases the execution time for accessing XPath expression of this form. Looking for an efficient solution to decrease this cost becomes necessary.

Enhance our model to consider multiple occurrences of texts in one element and other document information like processing instructions, external entities, and CDATA sections.

REFERENCES:

Chen, Q., Lim, A., Ong, K. W. & Tang, J. Q. (2006). Indexing Graph-Structured XML Data for Efficient Structural Join Operation. Data & Knowledge Engineering 58: 21.

Fujimoto, K., Yoshikawa, T., Kha, D. D., Yashikawa, M. & Amagasa, T. (2005) A Mapping Scheme of XML Documents into Relational Databases Using Schema-based Path Identifiers. International Workshop on Challenges in Web Information and Integration (WIRI'05).

Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E. & Zhang, C. (2002) Storing and Querying Ordered XML using a Relational Database System. SIGMOD pp. 204-215).

Torsten, G., Keulen, M. V. & Jens, T. (2004). Accelerating XPath Evaluation in Any RDBMS. ACM Transactions on Database Technology 29: 40.

Jagadish, H., Al-khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos, S., Patel, J., Srivastava, D., Wiwatwattana, N., Wu, Y. & Yu, C. (2003) TIMBER: A Native XML Database. In: SIGMOD, San Diego, CA.

M. Grinev, A. Fomichev & Kuznetsov, S. (2004) Sedna: A Native XML DBMS. MODIS ISPRAS.

Chung, S. M. & Jesurajaiah, S. B. (2005) Schemaless XML document management in object-oriented databases. Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on pp. 261-266 Vol. 261).

Zhang, H. & Tompa, F. W. (2004a) Querying XML documents by dynamic shredding. In: Proceedings of the 2004 ACM symposium on Document engineering. ACM, Milwaukee, Wisconsin, USA.

Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D. J. & Naughton, J. F. (1999) Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB pp. 302–314).

O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G. & Westbury, N. (2004) ORDPATHs: insert-friendly XML node labels. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, Paris, France.

Tan, Z., Xu, J., Wang, W. & Shi, B. (2005) Storing Normalized XML Documents in Normalized Relations. the Fifth International

Conference on Computer and Information Technology (CIT'05) pp. 123-129).

Leonardi, E. & Bhowmick, S. S. (Nov 2005). XANDY: A scalable change detection technique for ordered XML documents using relational databases. Data & Knowledge Engineering 59: 32.

Atay, M. (2006) XML2REL: An Efficient System for Storing and Querying XML Documents Using Relational Databases. In: Graduate School, p. 127. Wayne State University, Detroit, Michigan.

Atay, M., Chebotko, A., Liu, D., Lu, S. & Fotouhi, F. (2007a) Efficient schema-based XML-to-Relational data mapping, pp. 458-476. Elsevier Science Ltd.

Min, J.-K., Lee, C.-H. & Chung, C.-W. (2008). XTRON: An XML data management system using relational databa. Information and Software Technology 50: 18.

Yun, J.-H. & Chung, C.-W. (2008). Dynamic interval-based labeling scheme for efficient XML query and update processing. Journal of Systems and Software 81: 56-70.

Ahlgren, P. & Colliander, C. (2009). Document-document similarity approaches and science mapping: Experimental comparison of five approaches. Journal of Informetrics 3: 49-63.

Dweib, I., Awadi, A., Alrahman, S. E. F. & Lu, J. (2008) Schemaless approach of mapping XML document into Relational Database. Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on pp. 167-172).

Torsten, G. (2002) Accelerating XPath location steps. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM, Madison, Wisconsin.

Soltan, S. & Rahgozar, M. (2006). A Clustering-based Scheme for Labeling XML Trees. IJCSNS International Journal of Computer Science and Network Security 6: 84-89.

Li, Q. & Moon, B. (2001) Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Data Bases pp. Pp: 361 - 370).

O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G. & Westbury, N. (2004) ORDPATHs: Insert-Friendly XML Node Labels. In: SIGMOD, Paris, France.

Chen, Y., Davidson, S., Hara, C. & Zheng, Y. (2003) RRXS: redundancy reducing XML storage in relations. In: Proceedings of the 29th international conference on Very large data bases - Volume 29. VLDB Endowment, Berlin, Germany.

Amer-Yahia, S., Du, F. & Freire, J. (2004) A comprehensive Solution to the XML-to-Relational Mapping Problem. In: WIDM'04, Washington, DC, USA.

Xing, G., Xia, Z. & Ayers, D. (2007a) X2R: a system for managing XML documents and key constraints using RDBMS. In: Proceedings of the 45th annual southeast regional conference. ACM, Winston-Salem, North Carolina.

Atay, M., Chebotko, A., Liu, D., Lu, S. & Fotouhi, F. (2007b). Efficient schema-based XML-to-Relational data mapping. Information Systems 32: 458-476.

Zhang, H. & Tompa, F. W. (2004b) Querying XML Documents by Dynamic Shredding. In: DocEng'04, Milwaukee, Wisconsin, USA.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (2007) Extensible Markup Language (XML) 1.0 (Fourth Edition). W3 Consortium.

Bansal, V. & Alam, A. (2001) Study and Comparison of Techniques to Efficiently Store and Retrieve XML Data.

Berglund, A., Boag, S., Chamberlin, D., Fernández, M., Kay, M., Robie, J. & Siméon, J. (2007) XML Path Language (XPath) 2.0. W3 Consortium.

Boag, X., Chamberlin, D., Fernández, M., Florescu, D., Robie, J. & Siméon, J. (2007) XQuery 1.0: An XML Query Language. W3 Consortium.

Fallside, D. & Walmsley, P. (2004) XML Schema Part 0: Primer Second Edition. W3 Consortium.

Thompson, H. S., Beech, D., Maloney, M. & Mendelsohn, N. (2004) XML Schema Part 1: Structures Second Edition. W3 Consortium.

Murata, M., Walsh, N. & McRae, M. (2001) TREX and RELAX Unified as RELAX NG, a Lightweight XML Language Validation Specification.

Møller, A. (2005) Document Structure Description.

Jelliffe, R. (2006) Resource Directory (RDDL) for Schematron 1.5.

Ray, E. T. (2003). Learning XML: [creating self-describing data] O'Reilly.

OASIS (2002) SGML: General Introductions and Overviews.

Clark, J. (2001) Tree Regular Expressions for XML (TREX).

Makoto, M. (2002) RELAX (Regular Language description for XML).

Klarlund, N., Møller, A. & Schwartzbach, M. (2000) DSD: A Schema Language for XML. In: . FMSP'00. . ACM, Portland, Oregon.

Hégaret, P. L., Whitmer, R. & Wood, L. (2005) Document Object Model (DOM). W3 Consortium.

W3C (2005) Document Object Model (DOM).

www.Altova.com/XMLSpy, w. M. c. (2008) SAX & DOM.

Bourret, R. (2005) XML and Databases.

Vakali, A., Catania, B. & Maddalena, A. (2005). XML Data Stores: Emerging Practices. IEEE computer March-April 2005.

Xing, G., Xia, Z. & Ayers, D. (2007b) X2R: A System for Managing XML Documents and Key Constraints Using RDBMS. ACMSE Winston-Salem, North Carolina, USA.

Grinev, M., Fomichev, A. & Kuznetsov, S. (2004) Sedna: A Native XML DBMS.

Codd, E. (1971) A database sub-language founded on the relational calculus. ACM SIGFIDET Workshop Data Description, Access and Control pp. 35-61).

Codd, E. "". Communications of the ACM, Vol. 13, No. 6, June, pp. 377-387. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13: 11.

Delobel, C. (1978) Normalization and hierarchical dependencies in the relational data model, pp. 201-222. ACM.

Codd, E. (1983) A relational model of data for large shared data banks, pp. 64-69. ACM.

Codd, E. (1970) A relational model of data for large shared data banks, pp. 377-387. ACM.

Zhou, J., Zhang, S., Wang, M. & Sun, H. (2006). XML-RDB Driven Semi-Structure Data Management. Journal of Information and Computing Science 1: 9.

Wang, H. & Meng, X. (2005) On the Sequencing of Tree Structures for XML Indexing. 21st International Conference on Data Engineering (ICDE 2005).

Jiang, H., Lu, H., Wang, W. & Yu, J. X. (2002) XParent: An Efficient RDBMS-Based XML Database System. ICDE pp. 335-336).

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: 32.

Knudsen, S. U., Pedersen, T. B., Thomsen, C. & Torp, K. (2005) RelaXML: Bidirectional Transfer between Relational and XML Data. 9th International Database Engineering &#38 (IDEAS'05) pp. 151-162).

Lee, Q., Bressan, S. & Rahayu, W. (2006) XShreX: Maintaining Integrity Constraints in the Mapping of XML Schema to Relational. 17th International Conference on Database and Expert Systems Applications (DEXA'06).

Oracle (n. a.) Oracle XML DB Developer's Guide 10g.

Wu, X., Lee, M. L. & Hsu, W. (2004) A prime number labeling scheme for dynamic ordered XML trees. Data Engineering, 2004. Proceedings. 20th International Conference on pp. 66-78).

Kobayashi, K., Wenxin, L., Kobayashi, D., Watanabe, A. & Yokota, H. (2005) VLEI Code: An Efficient Labeling Method for Handling XML Documents in an RDB. Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on pp. 386-387).

DB, O. X. (n.a) Oracle XML DB. Oracle Technology Network.

Pal, S., Fussell, M. & Dolobowsky, I. (December 2005) XML Support in Microsoft SQL Server 2005.

Ramanath, M. (2006) Schema-based Statistics and Storage for XML. In: Faculty of Engineering, p. 175. INDIAN INSTITUTE OF SCIENCE, BANGALORE – 560 012, INDIA.

Oosten, J. v. (July 2002) Basic Category Theory, Department of Mathematics, Utrecht University, The Netherlands.

Megginson, D. (27-April 2004) Simple API for XML. SAX.

Corporation, M. (2009) Microsoft Office Word 2003 system requirements. Microsoft Corporation.

Washington University, C. S. E. R. (2002) XMLData Repository.

Busse, R., Carey, M., Florescu, D. & Kersten, M. (2002) XMark- An XML Benchmark Project.

Runapongsa, K., Patel, J. M., Jagadish, H., Chen, Y. & Al-Khalifa, S. (2006). The Michigan benchmark: towards XML query performance diagnostics. Information Systems 31: 73-97.

APPENDIX A

Theorem-6.1:

(a) Under the following assumptions:

1- We will assume that the locations of insertion have the same probability, *t*

$$P[X = x] = 1/n,\ \ x = 2, 3, \ldots n+1$$

where X denote the location of insertion.

2- We will assume that the time decreases from *1.28 t* at location 2 to *t* at location *n+1* uniformly, i.e.

$$P[Y= 1.24 - (0.24*(y-2)/(n-1)]\ t] = 1/n, y= 2,3 \ldots n+1$$

where *Y* denotes the time required to build the document after insertion new token at position *y*. We have:

$$E_{11}=E_{MAXDOR}[BCDT] = 1.24\ t - 0.12\ t\ (n-1)/n$$

(b) $E_{12} = E_{Blobal}[BCDT] = t$

(c) $E_{13} = E_{Acc}[BCDT] = t$

Where E_{model} denotes the expected value of BCDT under the model:

Proof: (a)

$$E11 = \sum_{y=2}^{n+1}\left(1.24 - 0.24\ \frac{y-2}{n}\right) t\ \frac{1}{n}$$

$$= 1.24\ t - \frac{0.24\ t}{n^2}\sum_{y=2}^{n+1}(y-2)$$

$$= 1.24\ t - \frac{0.24\ t}{n^2}\sum_{z=0}^{n-1}(z)$$

$$= 1.24\ t - \frac{0.24\ t}{n^2}\ \frac{n(n-1)}{2}$$

$$= 1.24\,t - \frac{0.12\,t(n-1)}{n}$$

b) For E_{12}, E_{13}, in both cases the tokens there are sorted in sequential order and the time needed for building the document is equal to t

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in ***Table 6.4*** and ***Table 6.5.***

Theorem-6.2:

(a) Under the following assumptions:

1- We will assume that the locations of insertion have the same probability, t

$$P[X = x] = 1/n, x = 2, 3 \dots n+1$$

where X denote the location of insertion.

2- We will assume that the time decreases from $n\ t_0$ at location 2 to t_0 at location $n+1$ uniformly, i.e.

$$P[Z= t_0\,[n - z + 2] = 1/n, y= 2,3 \dots n+1$$

where Z denotes the time required to insert the new node at position z.

We have:

$$E_{22} = E_{Global}[ICDT] = t_0\,(n+1)/2$$

(b) E_{21}=$E_{MAXDOR}[ICDT] = t_0$

(c) $E_{23} = E_{Acc}[ICDT] = t$

where E_{model} denotes the expected value of $ICDT$ under the model:

Proof:

a)

$$E_{22} = \sum_{k=1}^{n} (n+k-2)\frac{t_1}{n+1} + t_0 \times \frac{1}{n+1}$$

$$= \frac{t_1}{n+1}(n^2 - \sum_{k=1}^{n}(k-1) + t_0 \times \frac{1}{n+1}$$

$$= \frac{t_1}{n+1}(n^2 - \sum_{z=0}^{n-1} z) + t_0 \times \frac{1}{n+1}$$

$$= \frac{t_1}{n+1}(n^2 - \frac{(n-1)n}{2}) + t_0 \times \frac{1}{n+1}$$

$$= \frac{t_1}{n+1}(\frac{n^2}{2} + \frac{n}{2}) + t_0 \times \frac{1}{n+1}$$

$$= \left(\frac{t_1}{n+1}\right)\frac{n}{2}(n+1) + t_0 \times \frac{1}{n+1}$$

$$= \left(\frac{t_1}{n+1}\right)\frac{n}{2}(n+1) + t_0 \times \frac{1}{n+1}$$

$$= \frac{n\,t_1}{2} + \frac{t_0}{n+1}$$

b) For E_{21}, since a relabeling is not needed after insertion of new token. Then, the cost of inserting new node is equal to t_0.

c) For $\boldsymbol{E}_{23}$, since a relabeling is needed after insertion for both pre-order and post-order then the equation will become as for *XML2RDB*, but the time needed for update is multiplied by 2, as follows:

$$E_{23} = \frac{n\,t_1}{2} \times 2 + \frac{t_0}{n+1}$$

$$E_{23} = n\,t_1 + \frac{t_0}{n+1}$$

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in ***Table 6.4*** and ***Table 6.5***.

APPENDIX B

Source program in Visual basic 6 for mapping XML documents into relational database, rebuilding, updating and querying document contents from relational database.

It is available as a digital copy attached with the thesis.