A Labeling DOM-Based Tree Walking Algorithm for Mapping XML Documents into Relational Databases

Dissertation Submitted in Fulfillment of the Requirements for the Degree of Doctor of Philosophy in the Faculty of Mathematical Sciences University of Khartoum

By:

Seif El.Duola Fath EL Rhman El Haj El.Bashair

Supervisor:

Dr. Izzeldin Kamil Amin

Co supervisor:

Dr. Mohamed Ahmed Al-Affendi

Aug 2011

DEDICATION

To my parents, brother Rabiea, my wife, and my children and all my family with love.

ACKNOWLEDGEMENTS

I am very thankful to my supervisors, Dr. Izzeldin K. Amin and Dr.Al- Affendi M. A., for their guidance, encouragement and help in fulfilling the requirements of this dissertation. Their invaluable advice and constant guidance has helped me improve myself as a researcher and guide me to improve my written skills.

I would like to extend my special thanks to the faculty and the staff of the Department of Computer Science, University of Khartoum, Sudan.

Above all, I would like to express my appreciation and thanks to my wife and my son Mohammed and Ahmed and my daughter Renad for their love, support, understanding, and patience.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract

XML has emerged as the standard format for representing and exchanging data on the World Wide Web. For practical purposes, it is found to be critical to have efficient mechanisms to store and query XML data, as well as to exploit the full power of this new technology. Several researchers have proposed to use relational databases to store and query XML data. With the understanding the limitations of current approaches, this thesis aims to propose an algorithm for automatic mapping XML documents to RDBMS with XML-API as a database utility. The algorithm uses best fit auto mapping technique, and dynamic shredding, of a specified selected XML document type (data-centric, document-centric, and mixed documents).e. The propose algorithm use DOM(Data Object Model) as a warehouse and stack as a data structure to mapping the XML document into relational database and reconstructing the XML document from the relational database. The experiment study show that the algorithm mapping document and reconstructing it again well. Finally, the algorithm compare with other algorithms the result is good in time and efficiency, also the algorithm complexity is O(11n+2).

ملخص الدراسة

تعتبر قوالب XML كتقنية حديثة الوسيط الأفضل في عملية عرض وتبادل البيانات عبر الانترنت وقد تم اعتماد هذا الوسيط كأحد الوسائل القياسية لتبادل البيانات مما أدى لكشف بعض القصور في إمكانية استرجاع البيانات والاستفسار عن بعض المعلومات، كما أنها واجهت صعوبات كبيرة في عملية التخزين، فهي لم تصمم كوسيلة للتخزين. أصبحت عملية تخزين واسترجاع البيانات هي التحدي الذي واجه المهتمين. أجريت دراسات عديدة على استخدام قواعد البيانات العلائقية في التخزين والاستفسار عن مستندات ال XML وللوصول لهذا كان لابد من التوفيق بين الطبيعة الهرمية لـXML والطبيعة المسطحة لقواعد البيانات.

طورت هذه الدراسة خوارزمية لتحويل البيانات من XML إلي قواعد البيانات العلائقية وإعادة بناء مستند XML مرة أخري. استخدمت هذه الخوارزمية النموذج الكائني (Data Object Model) كوسيط تخزين و استخدمت المكدسة كبنية بيانات لدفع البيانات من مستند XML إلي قواعد بيانات علائقية كما أنه تم استخدام الطريقة القياسية لفهرسة شجرة كائنات مستند XML و استفادة من الفهرسة في عملية استرجاع البيانات مرة أخري لمستند XML. جربت هذه الخوارزمية مع جميع أنواع البيانات في XML وكانت نتيجة قياس كفاءة الخوارزمية O(11n+2) كما إنها تتزايد بتناسب خطية مع حجم المستند المراد تحويله.كما انه تم مقارنة هذه الخوارزمية مع مجموعة من الخوارزميات فأثبت كفأتها من حيث التنفيذ و الزمن.

CHAPTER 1 THE PROBLEM

1.1 Introduction

XML is widely accepted as a standard medium for representing data exchanged between businesses on Internet since 1998. However, it was not designed for efficient storage and retrieval of [1]. As a result, seeking an efficient storage and query medium of XML documents is an attractive area of research in the database community.

For that, Mapping XML documents to RDBMS has been studied for the last few years to leverage the powerful, reliability, concurrency control, integrity, crash recovery and multi-user access of RDBMS [2, 3, 4, 5, 6, 7, 8, 9, 10, 11], which are not available in XML technology until now. These studies are trying to bridge the technology gap between XML hierarchical ordered structured and RDBMS tabular unordered structure. Existing Mapping techniques from XML-to-relational can be generally classified into two tracks: the first one is the structured-centric technique, which depends on the XML document structure to guide the mapping process [9,17,18,19], and the second track is the schema–centric, which makes the use of schema information such as DTD or XML schema to derive an efficient relational storage for XML documents [2, 3, 4, 6, 7, 8].

None of the above mapping XML-to-Relational technique gave an ideal solution to all the types of XML documents, which are data-centric, document-centric, and mixed documents of the previous two.

The aim of this research is to propose an algorithm for automatic mapping of XML documents to RDBMS with XML-API as a database utility. The algorithm uses best fit auto mapping technique, and dynamic shredding, of a specified selected XML document type (data-centric, document-centric, and mixed documents). The proposed algorithm will be used to overcome the database vendor dependency and XML document types and information loss stored in the original XML documents due to the shredding process. Also, the XML-API as a database utility will simplify the mapping process for loading the XML documents, selecting the best fit mapping technique, querying, retrieving, and managing the XML documents stored in the Database.

1.2 Problem Statement:

XML is becoming the standard medium for data exchange and representation over the web that can be shared between business partners. It is not designed to facilitate efficient retrieval of data or data storage [20]. On the other hand, Relational Database Management System (RDBMS) is one of the successful database management systems which are currently the most widely deployed data-storage system, particularly for large-scale databases. It's Scalability, reliability, integrity, multi-user, concurrency control, recovery mechanism, and easy implementation [21], makes it the best choice to store and retrieve XML documents.

To bridge the gap between the two technologies, there have been many studies done to map the XML documents to the RDBMS. Some of them are using the shredding approaches with indexing techniques to store data-centric XML documents in RDBMS [22, 23, 24, 25], while others store document-centric XML documents as Character Large Objects (CLOBs) in a relational database [26]. But a few of them are trying to deal with all types of XML documents (data-centric, document centric and mixed documents). [23]

As a result, there is still a need for efficient algorithm to take the advantages of the XML and RDBMS technologies and deal with unspecified XML documents types.

Until now, the research efforts are directed towards three areas:

a) The First is using Relational Data Base Management System (RDBMS) [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]as a well established medium to store and retrieve data in the business area.

b) The second is using Object Relational Database Management System (ORDBMS) [12], to take advantages of the facilities in RDBMS and some of OORDB.

c) The third is using new approach dedicated for XML documents to create a native XML database [13, 14, 15, 16]. This approach [1] includes modules XLink, XPath, Xquery, XSLT, and SOAP which are built from scratch for specific purpose to store and query XML documents. However, the approach is still short to reach the powerful capabilities of existing relational database system and also does not have complex search tools like relational database tools.

1.3 Objectives of research:

The research aims to achieve the following objectives:

1) Propose an automatic conversion of XML documents to relational database in order to:
    a) Save the time by reducing human interference and the need of user's previous knowledge of the mapping process
    b) Enhance the quality of mapping process.
2) Propose a dynamic mapping technique for an XML document in order to:
    a) Deal with any XML document types, structured, unstructured, and semi-structured documents.
    b) Improve the performance of the mapping process.
    c) Overcome the limitation on size of the XML documents.
    d) Reduce the loss of information results from fixed shredding process.
3) Build a high-level XML-API utility for database in order to:
    a) Load XML documents to be stored in a relational database
    b) Manage in an efficient way the querying ofan XML document.
    c) Manage XML documents by storing in a database, deleting, and retrieving the names of XML documents.

d) Reduce the need for support of vendor-specific feature.

e) Have a good GUI application to avoid users command entry syntax errors and maintain semantic consistencies among all the tabular data in a relational databases.

1.4 Main Contribution:

In this research, we propose an automatic mapping technique of XML documents to RDBMS with XML-API for a database. This technique will leverage the advantages of mature relational database features and the strength of XML in data representation and exchange on the Internet. To accomplish this goal the research will propose a new Dynamic shredding mapping technique for the mapping XML-to-relational to overcome the issues of the XML documents size, loss of information stored in the original documents, and mixed XML document types. The new Technique is carried out in two step:

1.4.1 Data mapping:

The data mapping module takes a valid XML document and map it into relational tuples, which are then loaded to the relational database. The data mapping module uses the XML-to

– relational mapping information to form relational tuples and to decide on the relation where these tuples will be loaded to database tables.

1.4.2 Database Reconstruction:

The reconstruction module retrieves the descendant of each XML element in the relational database. Then reconstructs the XML subtree corresponding to these element by putting the proper tags around the relations.

1.5 Thesis Tools:

To get the above objective, the following tools will be used during the research:

1) One of the most popular Relational Database Managemnt Systems with high share in the market (e.g. SQL server, Access DB2, … etc) for testing purposes[27].

2) An XML editor software to create XML documents (e.g. Composer, XMLSpy 2006, TurboXML, , … etc) [28], which can be made available.

3) A Programming language to implement the techniques and the XML API. Vbasic 6 was selected for the following reasons:

- VBasic 6 is being familiar to the researcher.
- It is object-oriented language that offers a standard library of a number of classes with varied functionalities.
- The basic functionality includes the reading, manipulating, and generating of XML text, which are the core features required to form the building blocks for developing fully functional, XML based application

4) Standard XML processor such as XQuery, XPath, and XSLT to be integrated with the mapping processor to provide the maximum amount of query and transformation flexibility on the database data.

5) Data for testing and analysis to be selected at that time with arrangement of the supervisors.

6) A benchmark for XML Data Management for evaluating the performance mapping technique and XML API data management systems (XMatch-1, XMark … etc), for comparison purposes, which could be justified at the time of testing and evaluation.

1.6 Thesis Organization:

The rest of the dissertation is organized as follows: chapter 2 present information retrieval for XML documents. Chapter 3 gives a Historical brief on the development in XML technology and its Patterns. Chapter 4 covers the Historical development in RDBMs and its Patterns. Chapter 5 related work and the current mapping approaches in the literature are analyzed and discussed. In chapter 6, we propose an algorithm with its analysis and design. In chapter 7, we give the results and experimental discussion and a conclusion and proposal of future work. All reference and appendix are given in Appendix A.

CHAPTER 2: INFORMATION RETRIEVAL USING XML

2.1 Introduction:

From a general web-surfer point of view, clearly, a human maintained index allows quick and accurate search of the web; in addition, the result of the queries are semantically organized. The indexing function is normally assigned a catalogue service module which is sometimes, overloaded. As a consequence of that, some important sites are not included and indexed; and the directories are not always up to date both due to the enormous growth of information sources available on the Internet and for the inherently time varying nature of the web pages. This fundamental issue of imprecise search results arises due to the representation of the data on the web. The semantic approach makes use of metadata descriptions to add meaning to a particular document's content. These metadata descriptions provide a greater probability of ascertaining what the user really desires when entering a search query.

XML messaging is at the heart of Web services, providing the flexibility required for their deployment, composition, and maintenance. Yet, current approaches to Web services development hide the messaging layer behind Java or C# APIs, preventing the application to get direct access to the underlying XML information.

XML [29] is an emerging standard for the representation and exchange of Internet data. It is obvious that relational, object relational, or object-oriented data models, do not suffice to integrate data from several data sources in the web. To support this, semi-structured data models have been proposed. The nature of this semi-structured data is that it is self-descriptive, and that it incorporates an optional XML definition (DTD) Document Type Descriptor.

One of three alternative data models can be deployed for the persistent storage of semi-structured data (i.e., XML documents). First, the development of specialized data management systems can be noted, such as Rufus [30], Lore [31, 32], and Strudel [33]. These are tailored to store and retrieve XML documents using special purpose indices and techniques of query optimization. Second, for an object-oriented database management system[34], or Object store, can be used to store XML documents because of the rich capability of this database system. Third, when a relational database management system (RDBMS) is employed

XML data is mapped into relations and queries posed in a semi structured query language which is then translated into SQL queries.

It is not possible to reliably predict which of these three approaches will be widely accepted. The first, the use of a specialized or special purpose database system, may work best, once needs are met concerning scalability and the level of maturity required for the handling of huge amounts of data. The second, an object-oriented database system, seems well-suited to complex data like XML, but vulnerable in the area of evaluating queries addressed to a very large database[35].

The third approach, (RDBMS), provides maturity, stability, portability, and scalability [35]. Furthermore, since a majority of the data on the web currently resides in and will continue to be stored in RDBMS, the opportunity arises for constructing a system using a RDBMS to store XML documents, making it possible to seamlessly query of data with one system and one query language. Given all these advantages, we believe that a RDBMS will be a viable option.

2.2 Information Retrieval on the Web:

Early definitions, dating from 1960, emphasize the very general nature of the task. For example, in Salton's classic textbook[36]:

"Information retrieval (IR) is the field concerned with the structure, analysis organization, storage, searching, and retrieval information."

In that textbook, information retrieval is assumed also to include database systems and question answering systems, and information is construed to mean document, references, text passages, or facts.

Over the 1970's and 1980's, much of the research in IR was focused on document retrieval, and the emphasis on this task in the text retrieval conference (TREC) evaluation of the 1990's has further reinforced the view that IR is synonymous with document retrieval. Web search engines are, of course the most common example of this type of IR system.

The huge success of Web search engines, such as Google, might lead some to question the need for extensive IR research. There are a number of possible answers to this question, but here are some major point:

- Web search and IR are not Equivalent. As mentioned previously, IR encompasses many types of information access. Web search is only part (although an important part) of this spectrum of information systems.
- Web queries do not represent all information needs. A broad range of information access technologies are being created to address the diversity of information needs of people in different contexts. If we focus only on the current mix of queries in Web search engine loge, many of those information needs will not be addressed.
- Web search engine are effective for some types of queries in some contexts. Retrieval experiments in the TREC environment, and commercial success, demonstrate that, for a very popular type of query (find the right home page), retrieving the pages containing all the query words and then ranking them according to other features based on links, anchor text, URLs, and HTML tags is very effective. For other type of queries, and in other environment (e.g. corporate), this approach to ranking is less successful.

XML could actually hinder some of the processes involved in the functioning of the system. Let us examine this issue further. Firstly, although XML has had a significant impact on information management for the Web, it is still unclear whether XML will be used primarily as a data exchange format, or also as a data storage format [36].

Since XML is a document format and not a data model, we need the ability to map XML-encoded information into a true data model [36]. More generally, we need to resolve the various conflicts that arise when we try to mix the concepts of documents and databases. For example, while some applications may wish to view a large set of XML documents as exactly that-a set of documents-other applications may prefer to think of each document as a database "load file," where all document contents are merged into a single large database [36].

In fact, we may wish to simultaneously view a body of XML information in both ways. But there are a number of questions that arise when translating a conceptual model of a database into an XML encoding. For example, when should attributes be used and when should sub-elements be used [36].

Efficient physical layout and indexing mechanisms are required for large stores of XML data. Random searching on an XML file is equivalent to key-based searches on any flat sequential file and can take an exorbitant amount of time. Also, we sometimes want to be able to provide the illusion of an XML data store when the data actually is stored elsewhere (such as in a traditional DBMS), and make the two modes work together [36]. This leads to the consideration of issues in information storage and management in XML and possible alternatives.

2.3 difference between standard information retrieval and XML retrieval:

The fundamental difference between standard information retrieval and XML retrieval is the unit of retrieval. In traditional IR, the unit of retrieval is fixed: it is the complete document. In XML retrieval, every XML element in a document is a retrievable unit. This makes XML retrieval more difficult: besides being relevant, a retrieved unit should be neither too large nor too small. The research presented here, a comparative analysis of two approaches to XML retrieval, aims to shed light on which XML elements should be retrieved. The experimental evaluation uses data from the Initiative for the Evaluation of XML retrieval. (INEX 2002).

2.4 XML RETRIEVAL APPROACHES:

Most full-text information retrieval systems ignore the information about the document structure and consider whole documents as units of retrieval. Such retrieval systems take queries that often represent a bag of words, where phrases or logical query operators could also be included. The final list of answer elements usually comprises ranked list of whole documents sorted in a descending order according to their estimated likelihood of relevance to the information need in the query. Accordingly, it is expected that for Content – And – Structure (CAS) retrieval topics in the first category a full-text information retrieval system would be able to successfully retrieve highly relevant articles.

Most native XML databases support XML-specific retrieval technologies, such as found in XPath and XQuery. The information about the structure of the XML documents is usually incorporated in the document index, allowing users to query both by document content and by document structure. This allows an easy identification of elements that belong to the XML documents, either by the path they appear in the document or by certain keywords they contain. Accordingly, it is expected that a native XML database would be suitable for CAS retrieval topics that belong in the second category.

In an effort to support a content-and-structure XML retrieval that combines both CAS topic categories, we develop a hybrid XML retrieval system that uses a native XML database to produce final answers from those documents that are estimated as likely to be relevant by a full-text information retrieval system.

The following sections describe the XML retrieval approaches implemented in the respective systems, together with some open issues that arise when a particular retrieval approach is applied.

2.5 Full Text Information Retrieval Approach

The efficient inverted index structure is first used with Zettair [38] to index the INEX (INitiative for the Evaluation of XML retrieval) XML document collection, which is a first indexed by using its efficient indexing scheme. This index stores the information about the parsed elements within articles together with the information about the attributes and all word occurrences; its size is roughly twice as big as the total collection size. The term postings file is stored in a compressed form on disk, so the size of the Zettair index takes roughly 26% of the total collection size. The time taken to index the entire INEX collection on a system with a Pentium4 2.66GHz processor and a 512MB RAM memory running Mandrake Linux 9.1 is around 70 seconds. A topic translation module is used to automatically translate an INEX CAS topic into a Zettair query. For INEX CAS topics, terms that appear in the <Title> part of the topic are used to formulate the query. Up to 100 <article> elements are then returned in the descending order according to their estimated likelihood of relevance to the CAS topic. One retrieval issue when using Zettair, which is in particular related to the XML retrieval process, is that it is effective retrieval scheme [37]. For the INEX XML document collection, we calculated the optimal slope parameter in the pivoted cosine ranking formula by using a different set of retrieval topics (those from the previous year, INEX 2002). When using terms from <Title> part of INEX topics while formulating Zettair queries, we found that a slope parameter with a value of 0.25 yields highest system effectiveness (although when longer queries are used, such as queries that contain terms from the <Keywords> part of INEX topics, a different value of 0.55 would be better [38]). Consequently, for INEX 2003 CAS topics we use the value of 0.25 for the slope parameter in the pivoted cosine ranking formula in Zettair.

2.6 Native XML Database Approach

With eXist, the INEX XML document collection is first indexed by using its efficient indexing scheme. This index stores the information about the parsed elements within articles together with the information about the attributes and all word occurrences; its size is roughly twice as big as the total collection size. The time taken to index the entire INEX collection on a system with a Pentium 4 2.6GHz processor and a 512MB RAM memory running Mandrake Linux 9.1 is around 2050 seconds.

A topic translation module is used to automatically translate an INEX CAS topic into two eXist queries: AND and OR. For INEX CAS topics, the terms and structural constraints that appear in the <Title> part of the CAS topic are used to formulate eXist queries. The query symbol operators &=, denoting logical "and" operation, and |=, denoting logical "OR" operation are used with eXist while formulating the above queries, respectively. The AND and OR eXist queries are depicted in solid boxes in Figure 2.1 where the elements to be retrieved are specified explicitly.

For an INEX CAS topic, our choice for the final list of answer elements comprises matching elements from the AND answer list followed by the matching elements from the OR answer list that do not belong to the AND answer list. If an AND answer list is empty, the final answer list is the same as the OR answer list. In both cases it contains (up to) 100 matching articles or elements within articles. The equivalent matching elements are also considered during the retrieval process.

We observed two retrieval issues while using eXist, which are in particular related to the XML retrieval process.

1. For an INEX CAS topic that retrieves full articles rather than more specific elements within articles, the list of answer elements comprises full articles that satisfy the logical query constraints. These articles are sorted by their internal identifiers that correspond to the order in which each article is stored in the database. However, there is no information about the estimated likelihood of

relevance of a particular matching article to the information need expressed in the CAS topic.26

2. For an INEX CAS topic that retrieves more specific elements within articles rather than full articles, the list of answer elements comprises most specific elements that satisfy both the content and the granularity constraints in the query. eXist orders the matching elements in the answer list by the article where they belong, according to the XQuery specification.

However, there is no information whether or not a particular matching element in the above list is likely to be more relevant than other matching elements that belong to the same article. Accordingly, ranking of matching elements within articles is also not supported.

The following sections describe our approaches that address both of these issues.

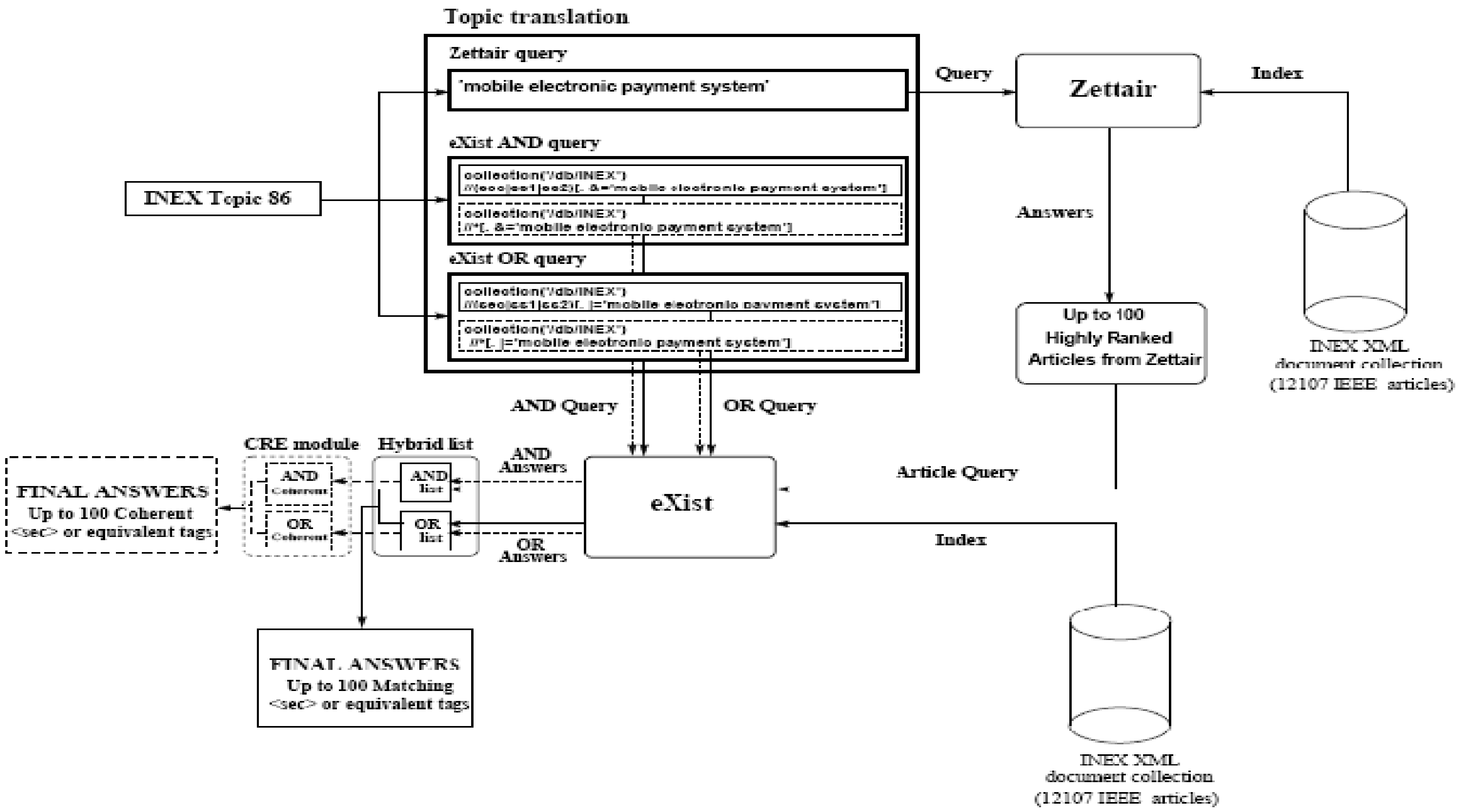

Topic translation
Zettair query
'mobile electronic payment system'
eXist AND query
collection('/db/INEX')
//(sec|ss1|ss2)[. &='mobile electronic payment system']
collection('/db/INEX')
//*[. &='mobile electronic payment system']
eXist OR query
collection('/db/INEX')
//(sec|ss1|ss2)[. |='mobile electronic payment system']
collection('/db/INEX')
//*[. |='mobile electronic payment system']
INEX Topic 86
Query
Zettair
Index
Answers
Up to 100 Highly Ranked Articles from Zettair
INEX XML document collection (12107 IEEE articles)
AND Query
OR Query
Article Query
CRE module
Hybrid list
AND Coherent
OR Coherent
AND list
OR list
AND Answers
OR Answers
eXist
Index
FINAL ANSWERS Up to 100 Coherent <sec> or equivalent tags
FINAL ANSWERS Up to 100 Matching <sec> or equivalent tags
INEX XML document collection (12107 IEEE articles)

Figure 2.1: A hybrid XML retrieval approach to INEX Content And Structure (CAS) topics

2.7 Hybrid XML Retrieval Approach

Our hybrid system incorporates the best retrieval features from Zettair [38] and exists. . Figure 2.1 shows the hybrid XML retrieval approach as implemented in the hybrid system. We use the CAS topic 86 throughout the example. Zettair is first used to obtain (up to) 100 articles likely to be considered relevant to the information need expressed in the CAS topic as into a Zettair query. For each article in the answer list produced by Zettair, both AND and OR queries are then applied by eXist, which produce matching elements in two corresponding answer lists. The answer list for an INEX CAS topic and a particular article thus comprises the article's matching elements from the AND answer list followed by the article's matching elements from the OR answer list that do not belong to the AND answer list.

The final answer list for an INEX CAS topic comprises (up to) 100 matching elements and equivalent element tags that belong to highly ranked articles as estimated by Zettair. The final answer list is shown as Hybrid list in Figure 2.1.

Figure 2.1 also shows queries and other parts of our hybrid system depicted in dashed boxes, where we also explore whether or not using CO-type queries could improve the CAS retrieval task. This can equally be applied to the hybrid approach as well as to the native XML database approach, since they both use eXist to produce the final list of matching elements. The next section explores this retrieval process in detail.

The hybrid XML retrieval approach addresses the first retrieval issue observed in a native XML database approach. However, because of its modular nature we observe a loss in efficiency. For a particular CAS topic, up to 100 articles firstly need to be retrieved by Zettair. This article list is then queried by eXist, one article at a time. In order to retrieve (up to) 100 matching elements, eXist may need to query each article in the list before it reaches this number. Obviously, having an equally effective system that produces its final list of answer elements much faster would be more efficient solution. The second retrieval issue observed in a native XML database approach still remains open, since for a particular article our hybrid XML retrieval system also uses exist to produce its final list of answer elements.

CHAPTER 3: An Overview of XML Standard Technology

3.1 XML Technology

The Extendible Markup Language (XML), a W3C Recommendation for marking up data as a standard medium for representing and exchanging structured and semi-structured data between business applications on the Internet, was published on 10th February 1998 as a First Edition, and its Second Edition XML 1.1 was published on 4th February 2004 [52]. It was designed to improve the functionality of the Internet by providing flexible information structuring. XML is extensible because it is not a fixed format like Hypertext Markup Language (HTML) (W3C 1999b) but a meta language for describing other languages. XML can be utilized to design customized markup languages for different types of documents. XML is a subset of Standard Generalized Markup Language (SGML) (ISO 1986), with some exceptions. SGML is a standard for defining descriptions of the structure of an electronic document. SGML is very powerful but complex, whereas XML is a lightweight version of SGML cleansed of all the features that make SGML too complex for the Internet. SGML is very comprehensive, which makes it hard to learn and expensive to implement.

Newly standardized applications to complete the data processing capabilities of XML were developed. XML Schema was approved as a W3C recommendation on 2nd May 2001 [52] aimed at replacing Document Type Definition (DTDs) as the official schema language for XML documents. Other XML schema languages, DSD was proposed by Klarlund in 2000 [52], and RELAX NG was proposed by OASIS in 2001 [52]. These schema languages are using to define and validate the structure and data of XML documents.

The XML Path Language (XPath) was approved as a W3C recommendation on 16th November 1999 [52] for addressing parts of an XML document. XML Query Language (XQuery) is recommended by W3C on 3rd November 2005 as XQuery 1.0 [52]. 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. A common feature of XPath and XQuery languages is a possibility to formulate paths in the XML graph. Such paths are a sequence of element or attribute names from the root element to a leaf.

XML Linking Language (XLink) Version 1.0 was approved as a W3C recommendation on 27th June 2001, which allows elements to be inserted into XML documents in order to create and describe links between resources [52].

The Extensible Stylesheet Language for Transformations (XSLT) is a W3C Recommendation in 16th November 1999 [52]. It is a language in XML markup designed to transform an XML document into another XML or plain text document.

3.1 XML Technology

The Extendible Markup Language (XML), a W3C Recommendation for marking up data as a standard medium for representing and exchanging structured and semi-structured data between business applications on the Internet, was published on 10th February 1998 as a First Edition, and its Second Edition XML 1.1 was published on 4th February 2004 [52]. It was designed to improve the functionality of the Internet by providing flexible information structuring. XML is extensible because it is not a fixed format like Hypertext Markup Language (HTML) (W3C 1999b) but a meta language for describing other languages. XML can be utilized to design customized markup languages for different types of documents. XML is a subset of Standard Generalized Markup Language (SGML) (ISO 1986), with some exceptions. SGML is a standard for defining descriptions of the structure of an electronic document. SGML is very powerful but complex, whereas XML is a lightweight version of SGML cleansed of all the features that make SGML too complex for the Internet. SGML is very comprehensive, which makes it hard to learn and expensive to implement.

Newly standardized applications to complete the data processing capabilities of XML were developed. XML Schema was approved as a W3C recommendation on 2nd May 2001 [52] aimed at replacing Document Type Definition (DTDs) as the official schema language for XML documents. Other XML schema languages, DSD was proposed by Klarlund in 2000 [52], and RELAX NG was proposed by OASIS in 2001 [52]. These schema languages are using to define and validate the structure and data of XML documents.

The XML Path Language (XPath) was approved as a W3C recommendation on 16th November 1999 [52] for addressing parts of an XML document. XML Query Language (XQuery) is recommended by W3C on 3rd November 2005 as XQuery 1.0 [52]. 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. A common feature of XPath and XQuery languages is a possibility to formulate paths in the XML graph. Such paths are a sequence of element or attribute names from the root element to a leaf.

XML Linking Language (XLink) Version 1.0 was approved as a W3C recommendation on 27th June 2001, which allows elements to be inserted into XML documents in order to create and describe links between resources [52].

The Extensible Stylesheet Language for Transformations (XSLT) is a W3C Recommendation in 16th November 1999 [52]. It is a language in XML markup designed to transform an XML document into another XML or plain text document.

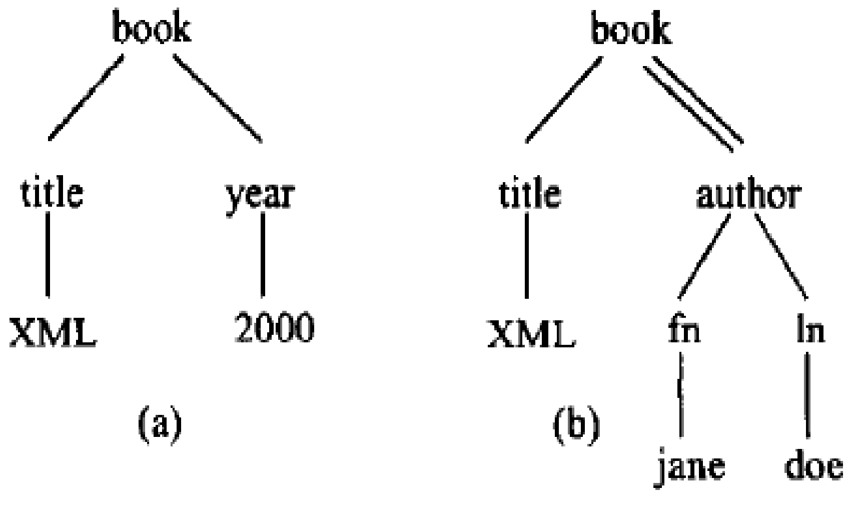


Figure 3.1: Query twig patterns

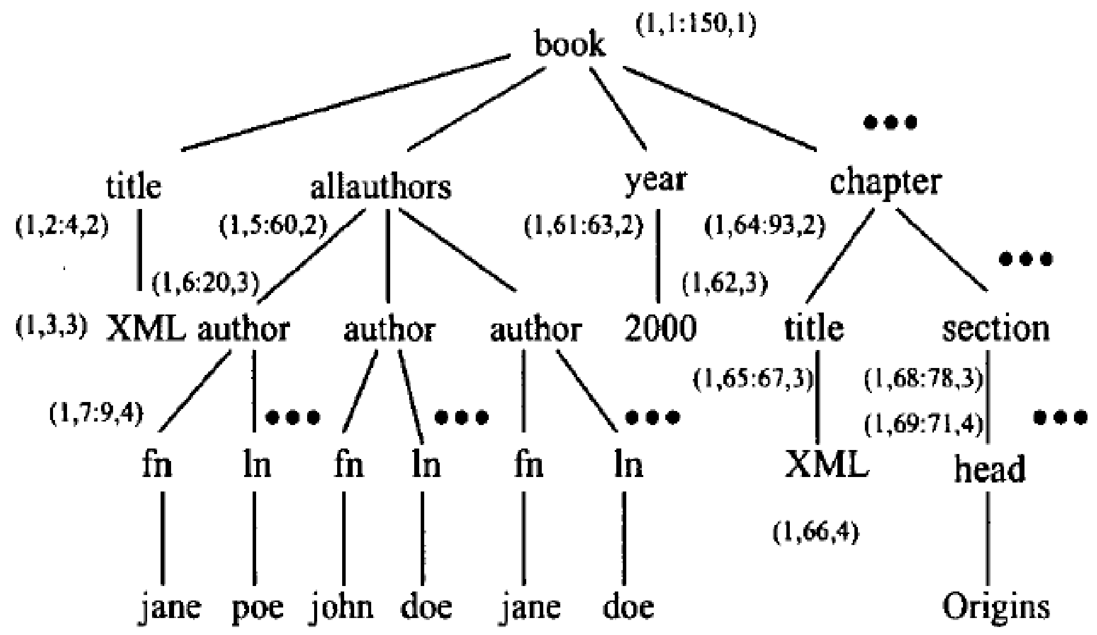


Figure 3.2: A sample of XML Tree representation [57]

New query languages are designed to extract information from XML documents. These query approaches not only use the contents of the XML document but also the structure of it. There are some existing algorithms proposed for XML tree pattern matching to present efficient techniques for querying XML data [53-54], beside XPath [52] and XQuery [52] approaches of W3C organization.

PathStack and TwigStack algorithms were proposed by Bruno et al., 2002, from the Columbia University and AT& T Labs-Research [57]. In their study, they tried to solve the limitation of decomposing of the twig pattern into binary structural ancestor-descendant relationships, which may generate large and possibly unnecessary intermediate results because the join results of individual binary relationships may not appear in the final results. However the approach is found to be suboptimal if there are Parent-Child (P-C) relationships in twig patterns. But, the method may still generate redundant intermediate results in the presence of P-C relationships in twig patterns [58].

TSGeneric+, twig join processing algorithm, was developed by Jiang et al., 2003, from the Hong Kong University of Science and Technology and the Chinese University of Hong Kong [59], for indexing XML documents, which makes use of a set of stacks to cache elements and a cursor interface that provides standard methods to return elements with possible matches in order to speed up the twig pattern match. Also, they proposed three edge-picking heuristics, top-down, bottom-up and statistics-based to select the first edge to start the processing. However, it still does not solve the problem of redundant intermediate results in the presence of P-C relationships [58].

ITwigJoin, a holistic twig Join algorithm, was proposed by Chen et al., 2005, from the National University of Singapore [58], which works correctly on any XML streaming scheme. They used the following recursive formula to determine the useful streams for evaluating a twig pattern Q using both Tag+Level schema and Prefix-Path Stream scheme. For a stream T of class q, they defined U_T to be the set of all descendant streams of T (including T) which are useful for the sub-twig of Q_q except that they only used stream T to match node q.

$$U_T = \begin{cases} \{T\} & \text{if } q \text{ is a leaf node;} \\ \{T\} \cup \{ \cup_{qi \in \text{child}(q)} C_i\} & \text{if none of } C_i \text{ is } \{\}; \\ \{\} & \text{if one of } C_i \text{ is } \{\}; \end{cases}$$

Where $C_i = \cup_{\text{Tc} \in \text{soln}(T,qi)} U_{Tc}$ [58]

Applied on Tag+Level scheme the algorithm can process Ancestor-Descendant (A-D) and Parent-Child (P-C) only twig patterns optimally, applied on Prefix-Path Stream (PPS) scheme the algorithm can process A-D only or P-C only or one branchnode only twig patterns optimally.

PRIX, PRufer sequences for Ind*e*x*i*ng XML, system was developed by Rao and Moon, 2006, from the University of Arizona [60], for indexing XML documents and processing twig queries. Their work is different from previous works, in that they tried to get further optimization for twig query processing without breaking a twig into root-to-leaf paths and merging the results.

TwigStackList, algorithm to process NOT-twig query, was proposed by Yu et al., 2006, from the National University of Singapore [61]. Also they developed a new concept Negation Children Extension to determine whether an element is in the results of a NOT-twig query.
An indexing framework, the layer index, and evaluation algorithms for performing the structural join operation on graph-structured XML data, was proposed by Chen et al., 2005, from the Hong Kong University of Science and Technology and University of California [55]. This approach constructed multiple nested layers of tree-structured indexes by recursively decomposing a graph into basic trees. Their study is different from Alkhalifa et al. [62] which adopted the representation, (DocID, LeftPos: RightPos) to index XML elements of a tree–structured model.

TJFast, holistic twig join algorithm, was proposed by Lu et al., 2005, from the National University of Singapore [54] based on their extending of labelling Dewey ID. Extended Dewey gives a powerful labelling scheme, since from the label of an element alone, all the elements names along the path from the root to the element can be derived. Algorithm TJFast is no longer guaranteed to be optimal in the case where the query contains parent-child relations between branching nodes and their children.

3.3 Validating: DTD and XSDL

An XML document can be validated against a Document Type Definition (DTD) or schema that is included in or referenced by the document. Since DTD and schemas describe the metadata of the document, they can be used to define a vocabulary that is a shared specification for documents in a particular domain of interest. Although DTDs are a part of XML Standard 1.0, they originate from SGML. A DTD specifies the structure of the XML document by defining elements of the document, one, zero-or-one, zero-or-more, and one-or-more occurrences of the elements and the hierarchical order between the elements. The DTD may define required and optional attributes of the elements and alternative values of the attributes. It may also contain references to other DTDs. Unfortunately; DTDs are not well-formed XML documents and provide little support for data typing, cardinality, and namespaces. A schema is an XML document for describing the structure of XML documents. XML Schema Definition Language (XSDL) (W3C 2001b), which is also known as XML Schema, is an XML language for schemas. XSDL offers a number of built-in data types and capabilities of defining data types. It allows applying data types to both element content and attributing values.

Although not all XML parsers are validating, the most popular ones enable that XML documents are validated against DTDs. In comparison, a number of XML parsers supporting validation against XSDL is small but increasing.

3.4 Parsing: SAX and DOM

There are two approaches for parsing XML documents. Simple API for XML (SAX) (SAX 2002) is an event-based application programming interface (API) that reports parsing events, such as the start and end tags, directly to the application through callbacks. The application implements handlers to deal with the different events. Since the original SAX did not support namespaces, SAX2 was developed. Document Object Model (DOM) (W3C 2002) is a tree-based API that converts an XML document into a tree structure. The application has access to navigate and manipulate this structure. It can also generate a well-formed XML document.

Comparing the parsing approaches, the SAX requires more programming due to handlers and makes it harder to visualize XML documents than the DOM. However, the SAX is faster and less memoryintensive because it does not load entire XML documents as tree structures into the memory.

There are several XML parsers for parsing XML documents. The most popular XML parsers support both SAX and DOM approaches.

3.5 Transforming: XSLT

XSL Transformation (XSLT) (W3C 1999a) is an XML language for transforming XML documents into other XML documents. XSLT is not intended as a complete general purpose XML transformation language but it is designed for use as a part of Extensible Stylesheet Language (XSL), which is a stylesheet language for XML. XSL includes a vocabulary for specifying formatting. For example, the block formatting represents the breaking of the content of a paragraph into lines.

A transformation expressed in XSLT describes the rules for transforming a source document into a result document. This stylesheet contains a set of template rules that consist of patterns and templates. This allows a stylesheet to be applicable to a wide class of documents that have structures similar to the source document. A pattern is matched against elements in the source document. A template is instantiated to create the part of the result document that is separate from the source document. In constructing the result, elements from the source can be filtered and reordered, and arbitrary structure can be added. Figure 3.3 a shows an example of an XSLT document and 4b is the output document of the transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0"
encoding="UTF-8"
indent="yes"/>
<xsl:strip-space elements="*"/>
<xsl:template match="PurchaseOrder">
<PurchaseOrder>
<BuyerPartyID>
<xsl:value-of select="BuyerParty/PartyID"/>
</BuyerPartyID>
<SellerPartyID>
<xsl:value-of select="SellerParty/PartyID"/>
</SellerPartyID>
```

```
<ProductID>
<xsl:value-of select="Product/ProductID"/>
</ProductID>
<Amount>
<xsl:value-of select="Product/Quantity"/>
</Amount>
</PurchaseOrder>
</xsl:template>
</xsl:stylesheet>
(a)
```

```
<?xmlversion="1.0" encoding="UTF-8"?>
<PurchaseOrder>
<BuyerPartyID>X</BuyerPartyID>
<SellerPartyID>Y</SellerPartyID>
<ProductID>ZZZ</ProductID>
<Amount>12.3</Amount>
</PurchaseOrder>     (b)
```

Figure 3.3: Example transformation from one format to another

Previously, an XSLT processor was a separate tool. Currently, many XML parsers are capable of XSLT processing.

3.6 XML Management Systems:

There are mainly two types of XML storage considered in the literature: relational database management systems and native XML technology. By relational storage, we mean that XML documents are mapped into relational tables. In contrast, in native storage, XML data can be stored in a versatile format and we can evaluate XML queries with algorithms that are tailored for XML. Figure 3.4 shows a classification of some existing proposals on managing XML data.

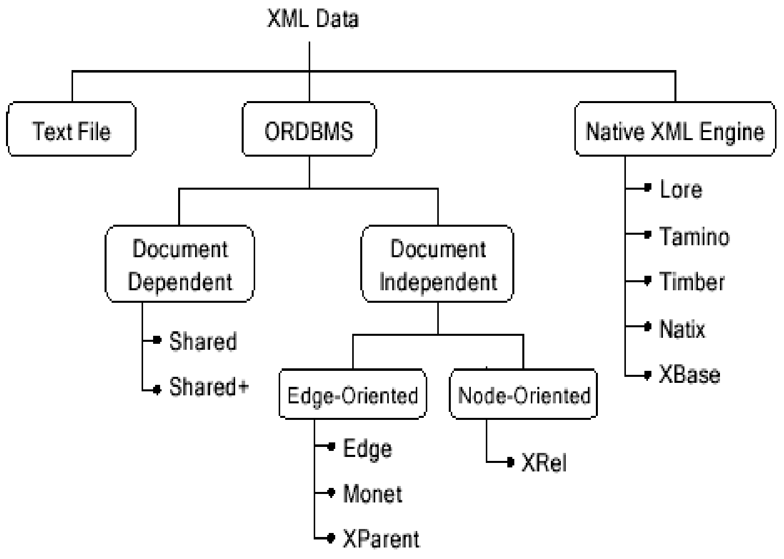


Figure 3.4: (Prototype) systems for managing XML documents

3.7 Relational Storage of XML Data:

Various techniques have been proposed to leverage the power of widely available object-relational databases for storing and querying XML data. The basic idea is that we shred XML documents into relational tables and access the data with SQL queries.

When XML data is stored in a relational database, a relational schema must be defined. The table schema can generate by either using or not using the schema information of an XML document to be stored. Such schema information could be given in the form of either a Document Type Definition (DTD) or an XML schema [63]. When the relational schema is generated based on the document schema, say DTD, different DTDs will lead to different table schemas, resulting in a document-dependent mapping. On the other hand, since any XML document can be modeled as an ordered tree, a single relational schema is able to describe the tree structure for all XML documents. No DTD information is required by this approach and all XML documents can share the same relational schema, resulting in a document-independent mapping.

3.8 Native XML Engines:

Native XML engines are systems that are specially designed for managing XML data. The storage and query processing techniques adopted by different systems may vary from each other in a noticeable way. One approach is to model XML documents using the Document Object Model (DOM) [64]. Internally, each node in a DOM tree has four filiation pointers and two sibling pointers. The filiation pointers include the first child, the last child, the parent, and the root pointers. The sibling pointers point to the previous and the next sibling nodes. The nodes in a DOM tree are serialized into disk pages according to depth-first order (filiation clustering) or breadth-first order (sibling clustering). Lore [65, 63] and XBase [66] are two instances of such a storage approach. The current release of TIMBER [67] transforms each node of the data tree into an internal representation and stores it into SHORE [68] as an atomic unit of storage. TIMBER is being engineered to package nodes in page-size containers due to SHORE's considerable overheads in dealing with small objects. Natix [69] uses a native storage format with the following features: (1) subtrees of the original XML document are stored together in a single (physical) record; (2) the inner structure of subtrees is retained; and (3) to satisfy special application requirements, the clustering requirements of subtrees are specifiable through a split matrix. Documents stored in Tamino [70,71] are grouped into collections. Within a collection, several document types can be declared and each incoming document validates against one of these types. The elements and attributes parsed from an incoming document can be stored in Tamino itself or in an external/built-in SQL database.

3.9 Index Structures:

Indexing structures used in relational databases are well-known and highly efficient. Using these indexing structures as a starting point for indexing XML documents, a natural evolution in the features and efficiency of said indexes has occurred and will continue to develop. This section starts by introducing a labeling scheme for nodes in a tree, presents preliminary index structures (B+-tree and XR-tree) used for XML documents, moves on to more sophisticated and efficient index methodologies (XB-tree, DataGuide, and ToXin), and finishes with a state-of-the-art indexing technique (constraint sequencing).

3.9.1. Node Labeling

When constructing a B+-tree, XR-tree, or XB-tree index on an OEM structure, the nodes must be labeled with a standard labeling scheme. Many labeling methods exist [72], but the most common and widely-used is an extension to Dietz’s numbering scheme (tree traversal order [73]) called extended preorder traversal [74]. Using this labeling method, each node in the tree is labeled with a pair of numbers <order,size>. This extension allows insertions to be made into the tree without the need for global reordering. It maintains the original idea of Dietz's scheme by imposing three conditions on the values for order and size.

1. For a tree node y and its parent x, order(x) < order(y) and order(y)+size(y) _ order(x)+ size(x). In other words, the interval [order(y), order(y)+size(y)] is contained in the interval [order(x), order(x) + size(x)].

2. For two sibling nodes x and y, if x is the predecessor of y in preorder traversal, then order(x)+ size(x) < order(y).

3. For any node x, size(x) _Xy size(y) for all y's that are a direct child of x. By using an arbitrarily large integer for size(x), future insertions into the structure can be made without the need for global reordering.

3.9.2. B+-Tree

In relational database systems, the B+-tree (a variation of the B-tree) is used to implement a dynamic multilevel index [75]. Offering advantages to indexed sequential files, a B+-tree does not require reorganization of the entire file to maintain performance. In other words, the tree will automatically reorganize itself with small, local changes when insertions and deletions occur. Due to its hierarchical nature, the B+-tree was used in an algorithm for processing XML structural joins

[76]. Although structural joins are discussed in greater detail in a later chapter, it is sufficient to mention that they require information about ancestors and descendants of a given element (possibly through multiple levels). For this reason, an algorithm and index structure that allows ancestors and descendants to be found and evaluated quickly will improve performance of structural joins. While it showed an improvement over a previous algorithm using R-trees for the same purpose, the B+-tree was later improved upon to produce the XR-tree and later the XB-tree.

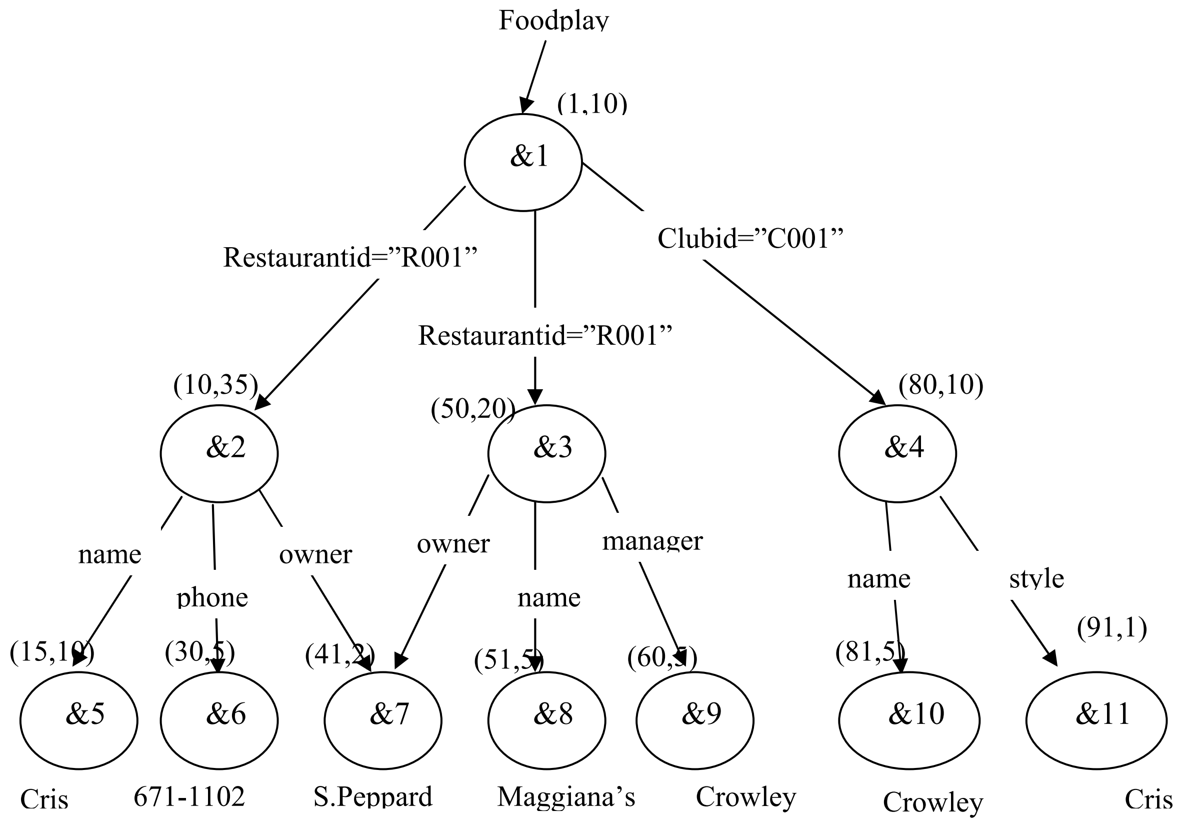


Figure 3.5: OEM Representation with Interval

3.9.3. XR-Tree

The XR-tree [77], known as the XML Region Tree, is a B+-tree that is built on the start points of the element intervals. Designed for strictly nested XML data, this type of index structure allows all ancestors and descendants for a given element to be identified with optimal worst case disk input/output cost. The XR-tree outperforms the B+-tree for processing structural joins, but it lacks the capability to handle highly recursive XML elements with the same efficiency [78].

3.9.4. XB-Tree

The XB-tree was developed by Bruno et al. [79] for use in processing holistic twig joins (a specialized version of structural joins). The XB-tree combines the structural features of both the B+-tree and the R-tree. It indexes the pre-assigned intervals of elements in the tree (similar to a one-dimensional R-tree) and then constructs the index on the start points of the intervals (similar to the standard B+-tree) [78]. The main difference is that the size portion of the <order,size> label must be propagated up the index. The main advantage of the XB-tree is that it quickly processes requests to find ancestors and descendants. A performance study [78] found that the XB-tree outperforms both the B+-tree and XR-tree for processing structural joins in XML documents.

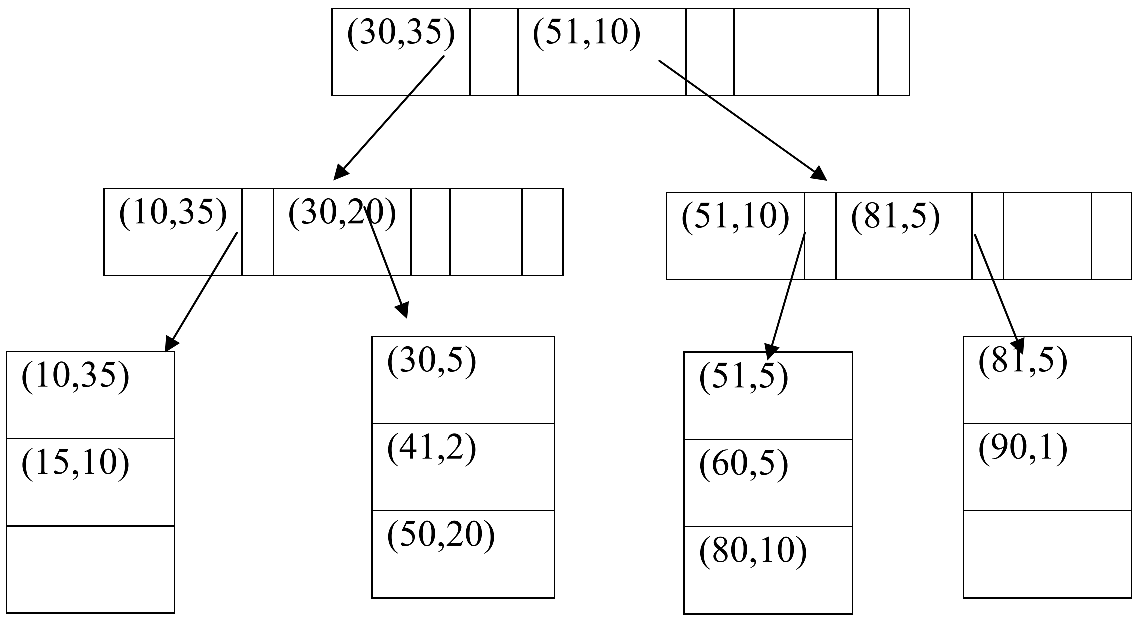


Figure 3.6: sample XB-tree

3.10 The XML Query Fundamentals

We present the background information of the XML query and notations used in this research. An XML document consists of nested elements enclosed by user-defined tags, which indicate the meaning of the content contained. Figure 3.7 shows an example of an XML document named "pub.xml", which contains some publication information. The hierarchical structure of an XML documents can be modeled as a tree. The XML documents on the Internet are a forest of XML trees and we call it an XML database.

```
<?xml version="1.0" ?>
```

```
<publication>
<journal title="DBMS">
<editor>Jack</editor>
<article>
<title>
Index Construction
</title>
<author>Smith</author>
</article>
</journal>
<journal title="Algorithm">
</journal>
</publication>
```

Figure 3.7: An example of an XML document

The semi-structured format of XML documents brings the possibility of using database technology to query the XML data instead of information retrieval techniques applicable only to plain text documents. However, the mature SQL queries can not be applied directly since XML documents do not necessarily conform to a predefined, rigid schema required by the traditional database system

[79]. Much research has been done on XML query languages. Although the query languages differ in detailed grammars, they share a common feature, that is: querying structure as well as the contents or values of elements. Queries in XML query languages make use of tree patterns to match portions of data in the XML database. For example, the following is a query expressed in Xquery [80] over the document in Figure 3.7 where "//" indicates ancestor-descendant relationship, and "/" indicates parent-child relationship. FOR $a IN document ("http://.../pub.xml")//journal/article $b IN $a/title WHERE $a/author ="Smith" RETURN <article>$b </article>

This query retrieves the titles of articles authored by "Smith" and published in a journal. It contains both structure and content information. In other words, this query will find all the matching of the tree pattern in the XML database.

3.11 XML Queries:

XPath and XQuery are the standard XML querying languages. An XML query specifies selection predicates for multiple elements or attributes that share some tree-based relationships (see Figure 3.8). In a query's tree-based representation, nodes represent an element tag, an attribute tag, or a value; edges represent hierarchical relationships between XML elements (ancestor–descendant, element– subelement, element–attribute, element–value, or attribute–value). Thus, both nodes and edges represent conditions that the retrieved XML documents must satisfy. We can classify XML queries in three ways:

• **Tree structure**. As Figure 3.8 shows, XML queries can be classified into simple path or branching path expressions. In the first case, the tree corresponds to a chain-path. In the second case, it contains branches and corresponds to a small tree, called a twig.

• **Starting node**. Total matching queries are those that start from the root of the document representation, whereas partial matching queries start from some internal node. For example, the document in Figure 3.8(a) does not satisfy the total matching query /cast/actor[@role= 'Leading actor]. It does, however, satisfy the partial matching query //cast/actor [@role='Leading actor].

• **Node types**. XML queries can contain nodes representing text associated with the father attribute or element node. We call such queries content-based queries because they check element or attribute content.

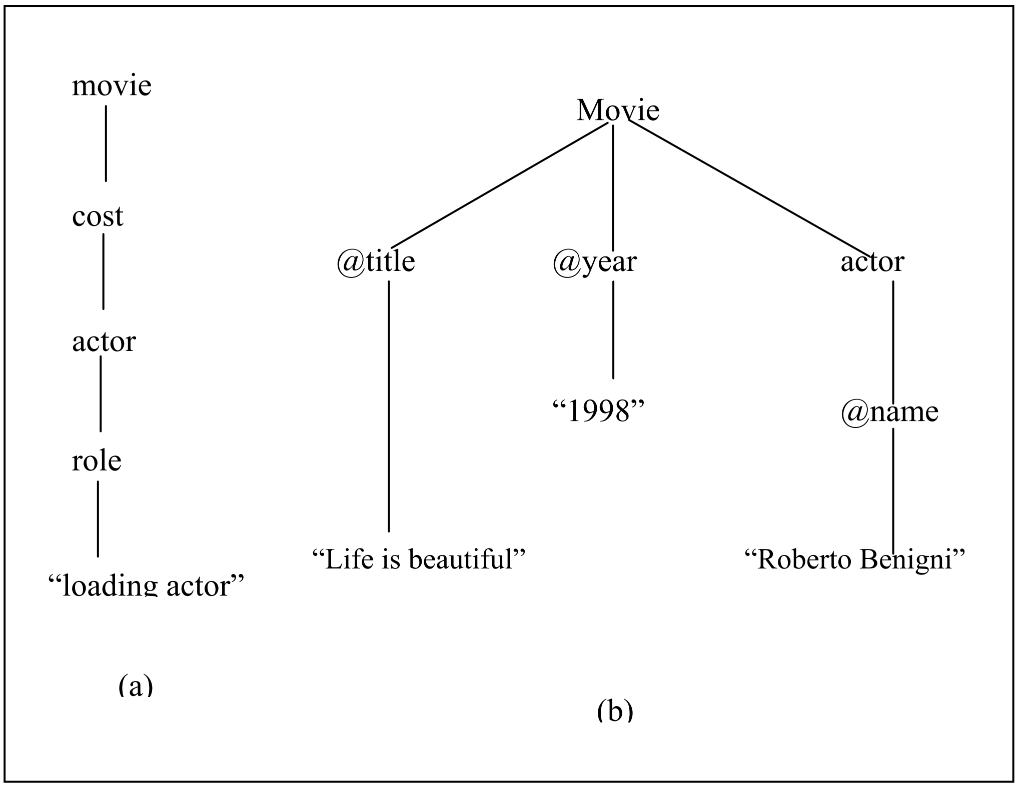


Figure 3.8: Tree Base Representation of queries

3.11.1 LOREL

LOREL was originally designed for querying semi structured data and has now been extended to XML data; it was conceived and implemented at Stanford University (S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener) and its prototype is at http://www-db.stanford.edu/lore. It is a user-friendiy language in the SQL\OQL style, it includes a strong mechanism for type coercion and permits very powerful path expressions, extremely useful when the structure of a document is not known in advance [81].

3.11.2 XML-QL

XML-QL was designed at AT&T Labs (A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu); its prototype is reachable at the url: http://www.research.att.com/sw/ toois/ xmlql as part of the Strudel Project. The XML-QL language extends SQL with an explicit CONSTRUCT clause

for building the document resulting from the query and uses the element patterns (patterns built on top of XML syntax) to match data in an XML document. XML-QL can express queries as well as transformations, for integrating XML data from different sources [82].

3.11.3 XML-GL

XML-GL is a graphical query language, relying on a graphical representation of XML documents and DTDs by means of labelled XML graphs. It was designed at Politecnico di Milano (S. Ceri, S. Comal, E. Damiani, P. Fraternali, S. Paraboschi and L. Tanca); an implementation is ongoing. All the elements of XML-GL are displayed visually; therefore, XML-GL is suitable for supporting a user-friendly interface (similar to QBE) [83].

3.11.4 XSL

The Extensible Style sheet Language (XSL) has facilities that could serve as a basis for an XML query language. An XSL program consists of a collection of template rules; each template rule has two parts: a pattern which is matched against nodes in the source tree and a template which is instantiated to form part of the result tree. XSL makes use of the expression language defined by XPath [79] for selecting elements for processing, for conditional processing and for generating text. It was designed by the W3C XSL working group (J. Clark editor) [84, 85, 86] .

3.11.5 XQL

XQL is a notation for selecting and filtering the elements and text of XML documents. XQL can be considered a natural extension to the XSL pattern syntax; it is designed with the goal of being syntactically very simple and compact (a query could be part of a UKL), with a reduced expressive power. It was designed by J. l~bie, Texcel Inc., J. Lapp, webMethods, Inc., and D. Schach, Microsoft Corporation [86, 87, 88,89, 90].

3.11.6. XPath

The simplest type of query in XML is an XPath expression . The XPath 1.0 [91] has been designed mostly as a navigation language that returns a subset of the nodes of a document. For instance, XSLT uses XPath heavily to match patterns that need transformation. When applied on a document, XPath returns a node set and not a sub-document. From the nodes, it is always possible to reconstruct the document (using the context to find the ancestors of the current node up to the root), but this is not the default behavior and the application using XPath needs to perform this reconstruction. Moreover, when the context is lost (e.g. data shipped from a remote site), this information is lost.

3.11.7 XQuery

XQuery is a query language for XML designed to be broadly applicable across many types of XML sources [52]. Designed to meet the requirements identified by the World Wide Web Consortium (W3C), XQuery operates on the logical structure of an XML document, and it has both human readable syntax and XML-based syntax. A grammar for XQuery is defined by the W3C [52]. While XQuery can successfully extract information from XML documents, there are no built-in optimization techniques that relate to the relational optimization techniques discussed earlier. XQuery [92] is an extension of the XPath language, sometimes called a superset of XPath. The most important extensions are the following:

1. XQuery introduces module definitions. A module can be a main module, which is a complete query program, or a library module that exports library functions and variables.
2. To facilitate more flexible control on query evaluation context, an XQuery module may contain a Prolog definition. Programmers can choose default namespaces at query time, import pre-defined schemas and library modules, bind global variables to some values, and define global functions.
3. A type-switch expression is a run-time dynamic type checking mechanism. Depending on the dynamic data type of an operand expression, a type-switch expression evaluates one of its case expressions and returns the expression as its own return value.
4. A constructor expression constructs an XML document fragment inside the query body. Query programmers can restructure XML data and produce different views.

3.12 Characterization of XML Query Styles:

This section presents a classification scheme for the different styles of XML queries. XML queries can be effectively categorized into three main operators: select, project, and join. Each of these operators can then be further decomposed into two distinct styles.

3.12.1 The Selection Operation

In relational algebra, the selection operator selects from a given table only those rows that satisfy a specified criteria or set of criteria. The returned value for the relational model is always an atomic value or set of atomic values (including the empty set). This definition can be applied to XML databases and queries. Due to the tree structure of an XML document, the path traversed during the execution of a selection can be either simple or complex.

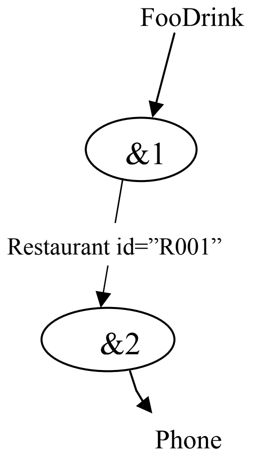


Figure 3.9: selection- Simple P

3.12.2 Selection: Simple Path

A simple path selection is shown in Figure 3.8. Stated in English, the query is asking for the phone number of Chili's (//restaurant[@name=''Chili's'']/phone in XPath). The path is considered simple because it does not cause the query tree to branch. The result (a phone number, 671-1102) is returned to the user.

3.12.3 Selection: Complex Path

In contrast to the simple path selection, a complex path selection causes the query tree to branch in order to return the requested values or objects. An example of a query that causes a complex path selection is to ask for the name of all restaurants owned by G. Peppard (given in XPath as //restaurant[@owner=''G. Peppard'']/name). All restaurants owned by G. Peppard (Chili's and Maggiano's) are returned to the user.

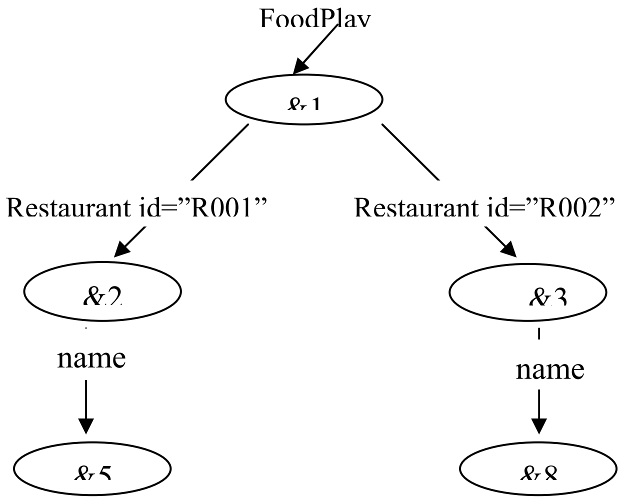


Figure 3.10: Selection-Complex Path

3.12.4 The Projection Operation

The projection operator in relational algebra retains certain columns from a given table and discards the others. Since XML documents have objects that are either atomic values or complex objects , projection operator in XML can return either a set of values or an object

3.13 XML advantage:

XML has been loaded with expectations. It has clear advantages over HTML and SGML. However, not all the features of XML are comparable with EDI. Some expectations are relevant in web publishing. The basic underlying ideas in XML are very simple: tags on data elements identify the meaning of the data rather than specifying how the data should be formatted (as in HTML), and the relationships between data elements are provided via simple nesting and references [93]. Yet the potential impact is significant: information content is separated from information rendering, making it easy to provide multiple views of the same data [93]. XML data files can be rendered via specifications in XSL, the Extensible Style sheet Language [93]. Generally, the types of web applications that will benefit from the use of XML are those that have any or all of the following features:

The following counterarguments point describes this advantage:

- ❖ XML is flexible: The ability to define other languages can potentially lead to problems because agreement on a common DTD or schema is not self-evident even in a small user community. Tens of e-business frameworks have been standardized using XML. This indicates that XML can be too flexible for this domain.
- ❖ XML is human-readable: If the XML document is indented for full-automated communication, human readability makes no sense. Even in semi-automated communication, it is easy to create quite unreadable XML documents. For example, the element ProNa may mean a product name. What about XML documents created in a different language?
- ❖ XML is self-describing: Although DTDs and schemas guarantee a certain amount of validity to XML documents, one may use a DTD, whereas another uses a schema to validate the document. How can it be ensured that the trading partners use the same version of DTDs or schemas?
- ❖ XML is structured: There are difficulties to store some characters, e.g. angle brackets, and binary data in XML documents. Since XML is structured text, it may take a lot of memory to store and a lot of time

to process this data. The possibility of specifying the contents is not free.

- ❖ XML is widespread and inexpensive: Processing data in XML documents does not necessarily stop at validation, parsing, or transformation of the documents but more steps are needed for many applications. For example, storing information in or retrieving it from the relational database is often necessary. The necessary widespread and inexpensive tools for all the steps that process the XML documents are not available.
- ❖ XML is platform-neutral and widely supported: Although XML is platform neutral and widely supported; the applications using XML are not guaranteed to be such. For example, a less-supported application may use a proprietary XML document format.
- ❖ XML-based systems have lower costs: Modification of legacy systems is not necessary because middleware can be built to transform data between XML and the native format. However, this does not eliminate the costs but shifts them from the legacy systems to the middleware.
- ❖ XML separates processing from content: Although XML separates processing from content; it depends on the developers to ensure that this separation really occurs. For example, if certain elements or

attributes require processing that is not supported by the basic XML technologies, these element or attribute names may need to be hard coded into the program. within the XML document.

CHAPTER 4: The Evolution of Relational Database Systems

4.1 Introduction

The earliest known use of the term ***'data base'*** was in June 1963, when the System Development Corporation sponsored a symposium under the title *Development and* Management of a Computer-centered Data Base. **Database** as a single word became common in Europe in the early 1970s and by the end of the decade it was being used in major American newspapers. (**Databank**, a comparable term, had been used in the Washington Post newspaper as early as 1966.)

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Bachman's early papers show that his aim was to make more effective use of the new direct access storage devices becoming available: until then, data processing had been based on punched cards and magnetic tape, so that serial processing was the dominant activity. Two key data models arose at this time: CODASYL developed the network model based on Bachman's ideas, and (apparently independently) the hierarchical model was used in a system developed by North American Rockwell, later adopted by IBM as the cornerstone of their IMS product.

The relational model was proposed by E. F. Codd in 1970. He criticized existing models for confusing the abstract description of information structure with descriptions of physical access mechanisms. For a long while, however, the relational model remained of academic interest only. While CODASYL systems and IMS were conceived as practical engineering solutions taking account of the technology as it existed at the time, the relational model took a much more theoretical perspective, arguing (correctly) that hardware and software technology would catch up in time. Among the first implementations were Michael Stonebraker's Ingres at Berkeley, and the System R project at IBM. Both of these were research prototypes, announced during 1976. The first commercial products, Oracle and DB2, did not appear until around 1980. The first successful database product for microcomputers was dBASE for the CP/M and PC-DOS/MS-DOS operating systems.

During the 1980s, research activity focused on distributed database systems and database machines, but these developments had little effect on the market. Another important theoretical idea was the Functional Data Model, but apart from some specialized applications in genetics, molecular biology, and fraud investigation, the world took little notice.

In the 1990s, attention shifted to object-oriented databases. These had some success in fields where it was necessary to handle more complex data than relational systems could easily cope with, such as spatial databases, engineering data (including software engineering repositories), and multimedia data. Some of these ideas were adopted by the relational vendors, who integrated new features into their products as a result.

In the 2000s, the fashionable area for innovation is the XML database. As with object databases, this has spawned a new collection of startup companies, but at the same time the key ideas are being integrated into the established relational products. XML databases aim to remove the traditional divide between documents and data, allowing all of an organization's information resources to be held in one place, whether they are highly structured or not.

4.2 A database:

is a collection of logically related data designed to meet the information needs of one or more users. A possible definition is that a database is a collection of records stored in a computer in a systematic way, so that a computer program can consult it to answer questions. For better retrieval and sorting, each record is usually organized as a set of data elements (facts). The items retrieved in answer to queries become information that can be used to make decisions. The computer program used to manage and query a database is known as a database management system (DBMS).[21].

The central concept of a database is that of a collection of records, or pieces of knowledge. Typically, for a given database, there is a structural description of the type of facts held in that database: this description is known as a schema. The schema describes the objects that are represented in the database, and the relationships among them. There are a number of different ways of organizing a schema, that is, of modeling the database structure: these are known as database models (or data models). The model in most common use today is the relational model, which in layman's terms represents all information in the form of multiple related tables each consisting of rows and columns (the true definition uses mathematical terminology). This model represents relationships by the use of values common to more than one table. Other models such as the hierarchical model and the network model use a more explicit representation of relationships.

A topic of great importance is to understand how to make a proper design of relational databases. The Relational model is not only very mature, but it has developed a strong knowledge on how to make a relational back-end fast and reliable, and how to exploit different technologies

The reason for my belief is that Relational Databases have a very well-known and proven underlying mathematical theory, a simple one (the set theory) that makes possible automatic cost-based query optimization, schema generation from high-level models and many other features that are now vital for mission-critical Information Systems development and operations.

4.3 Database models

Various techniques are used to model data structure. Most database systems are built around one particular data model, although it is increasingly common for products to offer support for more than one model. For any one logical model various physical implementations may be possible, and most products will offer the user some level of control in tuning the physical implementation, since the choices that are made have a significant effect on performance. An example of this is the relational model: all serious implementations of the relational model allow the creation of indexes which provide fast access to rows in a table if the values of certain columns are known.

A data model is not just a way of structuring data: it also defines a set of operations that can be performed on the data. The relational model, for example, defines operations such as selection, projection, and joins. Although these operations may not be explicit in a particular query language, they provide the foundation on which a query language is built.

4.4 Flat model

The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password that might be used as a part of a system security database. Each row would have the specific password associated with an individual user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is, incidentally, a basis of the spreadsheet.

4.5 Hierarchical model

In a hierarchical model, data is organized into a tree-like structure. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM. Most desktop computers also employ a hierarchical file system. This structure allows one 1:N relationship between two types of data. This structure is very efficient to describe some of the relationships in the real world. However, the hierarchical structure is inappropriate in many cases and is inefficient for certain database operations.

4.6 Network model

The network model (defined by the CODASYL specification) organizes data using two fundamental constructs, called *records* and *sets*. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets.

The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

Although it is not an essential feature of the model, network databases generally implement the set relationships by means of pointers that directly address the location of a record on disk. This gives excellent retrieval performance, at the expense of operations such as database loading and reorganization.[21].

4.7 Dimensional model

The dimensional model is a specialized adaptation of the relational model used to represent data in data warehouses in a way that data can be easily summarized using OLAP queries. In the dimensional model, a database consists of a single large table of facts that are described using dimensions and measures. A dimension provides the context of a fact (such as who participated, when and where it happened, and its type) and is used in queries to group related facts together. Dimensions tend to be discrete and are often hierarchical; for example, the location might include the building, state, and country. A measure is a quantity describing the fact, such as revenue. It's important that measures can be meaningfully aggregated - for example, the revenue from different locations can be added together.

In an OLAP query, dimensions are chosen and the facts are grouped and added together to create a summary.

The dimensional model is often implemented on top of the relational model using a star schema, consisting of one table containing the facts and surrounding tables containing the dimensions. Particularly complicated dimensions might be represented using multiple tables, resulting in a snowflake schema.

A data warehouse can contain multiple star schemas that share dimension tables, allowing them to be used together. Coming up with a standard set of dimensions is an important part of dimensional modeling.

4.8 Object database models

In recent years, the object-oriented paradigm has been applied to database technology, creating a new programming model known as object databases. These databases attempt to bring the database world and the application programming world closer together, in particular by ensuring that the database uses the same type system as the application program. This aims to avoid the overhead (sometimes referred to as the *impedance mismatch*) of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). At the same time object databases attempt to introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

A variety of these ways have been tried for storing objects in a database. Some products have approached the problem from the application programming end, by making the objects manipulated by the program persistent. This also typically requires the addition of some kind of query language, since conventional programming languages do not have the ability to find objects based on their information content. Others have attacked the problem from the database end, by defining an object-oriented data model for the database, and defining a database programming language that allows full programming capabilities as well as traditional query facilities.

Object databases suffered because of a lack of standardization: although standards were defined by ODMG, they were never implemented well enough to ensure interoperability between products. Nevertheless, object databases have been used successfully in many applications: usually specialized applications such as engineering databases or molecular biology databases rather than mainstream commercial data processing. However, object database ideas were picked up by the relational vendors and influenced extensions made to these products and indeed to the SQL language.

4.9 Relational model

The relational model was introduced in an academic paper by E. F. Codd in 1970 [20] as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. The data structures in these products are tables, rather than relations: the main differences being that tables can contain duplicate rows, and that the rows (and columns) can be treated as being ordered. The same criticism applies to the SQL language which is the primary interface to these products. There has been considerable controversy, mainly due to Codd himself, as to whether it is correct to describe SQL implementations as "relational": but the fact is that the world does so, and the following description uses the term in its popular sense.

A relational database contains multiple tables, each similar to the one in the "flat" database model. Relationships between tables are not defined explicitly; instead, *keys* are used to match up rows of data in different tables. A key is a collection of one or more columns in one table whose values match corresponding columns in other tables: for example, an *Employee* table may contain a column named *Location* which contains a value that matches the key of a *Location* table. Any column can be a key, or multiple columns can be grouped together into a single key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not originally intended to be one.[21].

A key that can be used to uniquely identify a row in a table is called a *unique key*. Typically one of the unique keys is the preferred way to refer to a row; this is defined as the table's primary key.

A key with an external real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), is sometimes called a "natural" key. If no natural key is suitable (think of the many people named *Brown*), an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that cannot break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

4.10 Relational operations

Users (or programs) request data from a relational database by sending it a query that is written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it is much more common for SQL queries to be embedded into software that provides an easier user interface. Many web sites, such as Wikipedia, perform SQL queries when generating pages.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables are combined into one, by doing a join. Conceptually, this is done by taking all possible combinations of rows (the Cartesian product), and then filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (which lists the rows in one table that are not found in the other), intersect (which lists the rows found in both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators - many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA - relation-valued attribute), then operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

The rows from a relational table are analogous to a record, and the columns to a field. Here's an example of a table and the SQL statement that creates the table:

```
CREATE TABLE ADDR_BOOK (
        NAME char(30),
        COMPANY char(20),
        E_MAIL char (25) )
```

NAME	COMPANY	E_MAIL
Haj fath	Haj Comp	Hajcompany@hotmail.com
Hafiz albarbari	Dal for Car	Dalcar@yahaoo.com

There are two basic operations you can perform on a relational table. The first one is retrieving a subset of its columns. The second is retrieving a subset of its rows. Here are samples of the two operations:

SELECT NAME, E_MAIL FROM ADDR_BOOK

NAME	E_MAIL
Haj fath	Hajcompany@hotmail.com
Hafiz albarbari	Dalcar@yahaoo.com

SELECT * FROM ADDR_BOOK WHERE COMPANY = ' Haj Comp '

NAME	COMPANY	E_MAIL
Haj fath	Haj Comp	Hajcompany@hotmail.com

You can also combine these two operations, as in:

SELECT NAME, E_MAIL FROM ADDR_BOOK WHERE COMPANY = 'Haj Comp'

NAME	E_MAIL
Sum song	sumsong@hotmail.com

You can also perform operations between two tables, treating then assets: you can make Cartesian product of the tables, you can get the intersection between two tables, you can add one table to another and so on. Later we'll present more details about these operations and how then can be useful.

Most set operations between tables are interesting but of limited use. After all, they will work as expected only when the tables have the same set of columns. The fun begins when you operate on tables that do NOT have the same set of columns.

4.11 Advantages of relational databases

The relational data model is appropriate for database applications requiring flexibility in the data structures and access paths of the database. Flexibility in the data structures allows the data to be stored as groups of logically similar data, with the groups being inter-linked as needed, rather than in a single, monolithic structure. Flexibility of the access paths permits the database to provide the exact data which each data consumer requires, in the most appropriate format for them. Relational databases are suitable both for applications under production control and for those in which there is a substantial need for ad hoc data manipulation by end users who are not computer professionals.

Relational Databases limit replication of data. By storing all the data pertaining to a particular item together, and then linking this collection of information to related objects, there is no need to store data about the original item in several different places. For instance, in a contact database, the information about each organization is stored in one place and information about individual contacts within that organization are linked to the relevant corporate information. There is therefore no need to store duplicate data.

By storing the data relating to an object in a single place, there is less likelihood of incorrect or incomplete data being stored or used. It only needs to be kept up-to-date in one place. If the data changes, it is only necessary to edit it in one place which saves time for those entering the data and reduces the likelihood of errors occurring on data entry. Data inconsistencies are thus more easily avoided.

Users of the data stored in the database do not have to be aware of the underlying structure. This permits the database designer to optimize the data storage while presenting the users with the data in the format which they need. For instance, in the contact database example, the individual and corporate information may be stored separately but a user who needs the address for an individual will be presented with a combination of the individual's information (name and title, say) along with the corporate data (company name and postal address).

Relational databases are very flexible. Because they can be used to present information in different ways, it is easy to add new views of the data as they are required. Inexperienced users can easily obtain the information they require without having to know anything about database design or implementation. Different components of the data can be maintained by different individuals so that the burden of keeping the data up-to-date can be spread over a number of people. Well-designed relational databases can provide appropriate data storage and retrieval facilities over a long timescale.

The relational database model is noted for its simplicity and expandability. The majority of large database applications are relational databases.

4.12 Understanding the application

The database designer must understand the application so that it is clear how the data will be maintained and how it will be used. Information to be ascertained at this stage of the design process must be the rules for creating and accessing data - how many users will perform each of these tasks, how much expertise they will have in using the system and how regularly the data will be updated or viewed. It is also important to understand the circumstances in which the data may be deleted and whether only specific users will be permitted to do this.

4.13 Organizing the data to form an initial conceptual model

At this stage, the database designer begins to identify the significant groups of data which are logically related. These logically distinct groups will correspond to database tables. Having recognized the differences between these sets of data, it is also necessary to recognize how the data sets will be linked.

4.14 Evaluating the conceptual model

The initial conceptual design should be used as a starting point for constructing "use cases" for each aspect of the database. Such "use cases" should cover different roles of user - administrator, manager, data provider, data consumer - as well as different scenarios of database use. By considering these scenarios in the context of the initial database design, it will become clear where shortcomings of the initial model lie.

4.15 Sequential Database Design

As a rule of thumb. the order in which constituent parts of the database should be constructed is as follows:

Identify tables

- Insert columns into tables
- Eliminate repeating groups
- Identify a primary key for each table
- Identify relationships between the tables
- Identify foreign keys

Repeating groups are groups of fields which recur in two or more tables. They indicate that the data has not been divided to a sufficiently atomic level and that this group should form the core of a new table with links to the tables requiring access to this information. Eliminating repeating groups in the database design will eliminate later problems in updating and accessing data.

This sequence of design stages should be iterated around until the design has been finalized.

Good Database Design :

- ❖ Mapping from a file-based process directly to a relational database is not good practice; the database should always be designed
- ❖ Several narrow fields are better than a single wide one
- ❖ Use character fields for numeric values such as phone numbers
- ❖ Do not store derived data in a record - only store raw, unprocessed data

4.16 The similarities and differences between XML pattern and RDBMS pattern

XML was originally proposed for representing and exchanging data between business applications on the Internet [94]. Where, RDBMS was proposed for storing and retrieving data [95].

XML can organize data in a hierarchical, object-oriented, and multidimensional way in the form of a tree with an arbitrary depth and width [96, 97]. While 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 (or rows) and the second level representing fields (or columns). An XML tree is clearly a more expressive way of representing data as no constraints are placed on either depth or width.

A comparison between XML technology and RDBMS technology shows that there is a technology gap between XML hierarchical ordered structure and RDBMS tabular unordered structure.

A Comparison between XML and RDBMS

XML	RDMS
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

Table 4.1: A Comparison between XML and RDBMS

CHAPTER 5: Related Works and Current Approaches

5.1 Introduction:

To store semi-structured data (i.e., XML documents) into persistence storage, three alternative approaches can be proposed: a special purpose database management system, an object-oriented database management system, and a relational database management system.

For researches to use a special purpose database system, Rufus [30], Lore [31, 32], and Strudel [33] report the development of research prototypes [39]. These are tailored to store and retrieve XML documents using special purpose indices and techniques of query optimization.

Insofar as an object-oriented database management system is concerned, the rich capability of such a database system permits the use of Object Orient (O2) [34] or Object store for the storage of XML documents (e.g. the MONET project [40]).

For a relational database management system one of two techniques can be considered. First, schema is extracted from XML documents based on semi-structured data [41, 42, 43]. By analyzing this semi-structured data, and the workload of a target application, efficient schema can be constructed. Thus, performance will be little concerned with the matter of how semistructured data is stored in RDBMS. Second, rather than extracting a schema, different techniques are studied for storing XML documents in relational databases. Examination of how XML data can be mapped into tables or relations can be found in [44, 35, 45, 46, 47, 48]. Besides the pure relational case [45], an object-oriented approach is also proposed. Furthermore, all of these use XML-QL [49] from XML documents to extract data, while simply ignoring the restructured element, (i.e., the result of SQL could be a XML document).

5.2 RELATED WORK

Various XML retrieval approaches were used by the participating groups in INEX 2003. These approaches were generally classified as model-oriented and system-oriented [51]. Their group followed the latter approach by using the initial hybrid XML retrieval system [49]. In an earlier work regarding retrieval from semi-structured documents, Wilkinson [43] shows that simply extracting components from documents likely considered to be relevant to the information need in a query leads to poor system effectiveness. However, in INEX 2003 approach they have investigated various extraction strategies which exist that produced effective results for CAS topics. The hybrid system with their CRE module (which they developed since INEX 2003) furthermore increases the retrieval effectiveness of the initial hybrid XML retrieval system.

The CSIRO group participating in INEX 2002 proposed a similar XML retrieval approach where PADRE, the core of CSIRO's Panoptic Enterprise Search Engine5 is used to rank full articles and elements within articles [32]. Unlike many full-text information retrieval systems, PADRE combines full-text and metadata indexing and retrieval and is also capable of indexing and retrieving more specific elements within articles. A post processing module is then used to extract and re-rank the full articles and elements within articles returned by PADRE. However, unlike our CRE retrieval module, the above approach ignores the structural elements within articles that contain the indexed element. Less specific and more general elements are therefore not likely to appear in the final answer list. For the purpose of ranking the resulting answers of XML retrieval topics, Wolffet al [35] extend the probabilistic ranking model by incorporating the notion of structural roles, which can be determined manually from the document schema. However, the term frequencies are measured only for the structural elements belonging to a particular role, without taking into account the entire context where all these elements belong in the document hierarchy. XRank [34] and XSearch [35] furthermore aim at producing effective ranked results for

XML queries. XRank generally focuses on hyperlinked XML documents, while XSearch retrieves answers comprising semantically related nodes. However, since the structure of IEEE XML documents in the INEX document collection does not typically meet the above requirements, neither of them (without some modifications) could be used in a straightforward fashion with the CAS retrieval task.

Carmine Cesarano [118] developed a system for semantically searching information on the Internet; the first implementation of the system shows an interesting performance in terms of searching precision. Several problems have to be still addressed. From the implementation point of view, we are testing, for pages found just by WSA, a penalizing factor *Kb*, thus assuming as *SyG*(*x*):

$$SyG(x) = \frac{1}{Kb \, . \, AP(x)} \qquad [118]$$

A radically different approach consists in ignoring *SyG*(*x*), i.e. they can decide to use standard search engines just as starting point for their search and completely ignoring the position in which each link is returned. Another problem is that the semantic network representing the ontology for each context has to be automatically built; machine learning strategies such as neural networks[118]

Urvi Shah [119] presented an approach to information retrieval over the Semantic Web utilizing a set of ontology's and inference engine.DAML+OIL is a schema system that provides key improvements over RDFS, including a built-in data typing system, support for enumerations, specializations on properties, and classification and typing by inference. Is used as the knowledge representation language and as an interface for the inference engine, thus fostering edibility and interoperability. The powerful support for rules formulation, constraints and question answering over schema information surpasses what is available in existing database technology.

Inference service can be used to answer queries about explicit and implicit knowledge spiced by the ontology thus providing a query answering facility that performs deductive retrieval from knowledge represented in DAML+OIL. Indeed, the retrieval of precise information is better supported by languages designed to represent semantic content and support logical inference. [119]

J. u [120] successfully implemented a systematic approach from ontology, agent, RDF and database systems. Given this lack of maturity in established implementation in semantic-web, this technology is still considered significant because the web infrastructure has become increasingly difficult to manage. The standard client/server approach to application design has led to the inception of a paradigm where representation and format code interacts with a server data store. This approach and the development techniques associated with it are confined to handling rigid and highly-controlled database environments. The limitation could have when the extension to three-tier and *n*-tier systems, which may affect effectiveness of access/retrieval. As applications migrate onto the Web, their inherent rigidity hampers further development and maintainability. However, Web applications range from portals to e-commerce sites. Thus, they must assemble data from diverse sources and services. Furthermore, the basic requirements for such applications tend to more adaptable in 'Internet time'. This is the sort of environment in which the extensibility of both XML and RDF proves to be of great significance. XML flexibly facilitates the adaptation of data formats, and RDF provides great benefits for the adaptation of data-processing rules.

Yuichi lizuka an integrated information retrieval method for the world wide web (WWW) [121]. This method bases the user interface on a universal relation. Given the user's query, the method returns a preliminary list of candidate sources from which the user selects target sources. The information desired is extracted from the target sources. Besides adopting a universal relation, the other features of this method are as follows. Template mechanism allows HTML pages to be treated as if they were relational database forms. Information resource management resolves heterogeneity of information sources. New application programming interface (API) allows users to construct a query by specifying items, which designate the retrieval items and the retrieval conditions. The proposed method resolves heterogeneity among sources and generates and presents retrieval candidates based on the user's query. Though there are many candidates, the users can choose the sources desired as retrieval targets. So, this method can treats independently controlled sources covering various subjects such as cars, PCs, restaurants, and so on. This method returns the lists of item values within an uniform user domain as the retrieval results. The method strength, the user needs not to analyze HTML documents to obtain the desired information and retrieval can be more easily. The method limitation it need in the future research to follows.- Create an automatic template generation method.- Expand the range of the proposed method to cover information sources such as text databases. XML data, and multimedia databases.

Mapping XML documents to RDBMS has been studied for the last few years to leverage the power, reliability, concurrency control, integrity, crash recovery, triggers, and multi-user access of RDBMS [21], which are not available in XML technology until now. These studies try to narrow down the technology gap between XML hierarchical ordered structure and RDBMS tabular unordered structure. Existing Mapping techniques from XML-to-relational can generally be classified into two tracks. The first one is the structured-centric technique, which depends on the XML document structured to guide the mapping process. The second track is the schema–centric, which makes the use of schema information as DTD or XML schema to derive an efficient relational storage for XML documents.

None of the previous mapping XML-to-Relational technique gave an ideal solution to overcome all issues of the mapping process, such as, size restriction, database vendor dependency, XML document types and information loss in the stored original XML documents due to the shredding process. Also, the size of the resulting relational database, and query performance are another issues of the mapping process.

5.3 Mapping XML documents to relational Database Researches:

There have been number of researches on mapping XML-to-relational database. One of the early studies in this area was proposed by:

Shanmugasundaram et al., 1999, in University of Wisconsin-Madison [98]. In their study, three techniques, i.e. Basic, Shared, and Hybrid Inlining mapping techniques, are proposed for DTDs to relational schemas. 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. To map from DTD to relational schema, they used the following transformation rules to get simplified DTD first:

if an element a is defined as <!ELEMENT a((b|c|e)?,(e?|(f?,(b,b)*))*)>, where b, c ,e and f are other elements. Then, the following rules in figure 5.1, 5.2 and 5.3 could be used to do simplification process:

$$\boxed{\begin{array}{r} (e1, e2)^* \rightarrow e1^*, e2^* \\ (e1, e2)? \rightarrow e1?, e2? \\ (e1|e2) \rightarrow e1?, e2? \end{array}}$$

Figure 5.1: Converting of a nested definition into flat representation [98]

Where "*" mean zero or more of a given element, "?" means zero or one element, "|" means a choice between two elements, and "+" means one or more of a given element.

e1** → e1*
e1*? → e1*
e1?* → e1*
e1?? → e1?

Figure 5.2: Reducing of unary operators to a single operator [98]

…, a* , …, a*, … → a*, …
…, a* , …, a?, … → a*, …
…, a? , …, a*, … → a*, …
…, a? , …, a?, … → a*, …
…, a , …, a, … → a*, …

Figure 5.3: Grouping of sub-elements having the same name [98]

5.3.1 The Basic Inlining Technique:

The Basic Inlining Technique, solves the fragmentation problem by inlining as many descendants of an element as possible into a single relation. However, *Basic* creates relations for every element because an XML document can be rooted at any element in a DTD [123]**.** Basic technique steps:

1) Each XML element is mapped to relation in RDB because an XML document can be rooted at any element in a DTD. So, Basic creates relations for every element in a DTD. For example, the author element in List 5.1 would be mapped to a relation with attributes firstname,

lastname and address.

2) Solves the fragmentation problem by inlining as many descendants of element as possible into a single relation.

3) To address set-valued attributes and recursion: they followed the standard technique for storing sets in RDBMS and created a relation these sets and link them using a foreign key. In List 1, when creating relation for article, they cannot inline the set of authors because the traditional relational model does not support set-valued attributes. Instead, they created a relation for author and link authors to articles using foreign key. And they expressed the recursive relationship using the notion of relational keys and use relational recursive processing to retrieve the relationship. To do so, they introduced the notion of a DTD graph as in Figure 5.3, which represents the structure of a DTD. Its nodes are elements, attributes and operators in the DTD. In the DTD graph, each element appears exactly once in the graph, each attributes and operators appear as many times as they appear in the DTD, and cycles in the DTD graph indicate the presence of recursion. The schema created for a DTD is the union of the sets of relations created for each element. In order to determine the set of relations to be created for a particular element, they created a graph structure called the element graph. The element graph is constructed as follows [98]:

a) Do a depth first traversal of the DTD graph, starting at the element node for which the relations are being constructed.

b) Each node is marked as "visited" the first time it is reached and is unmarked it once all its children have been traversed.

c) If an unmarked node in the DTD graph is reached during first traversal, a new node bearing the same name is created in the element graph.

d) A regular edge is created from the most recently created node in the element graph with the same as the DFS parent of the current DTD node to the newly created node. Figure 5.4 shows the element graph of editor element.

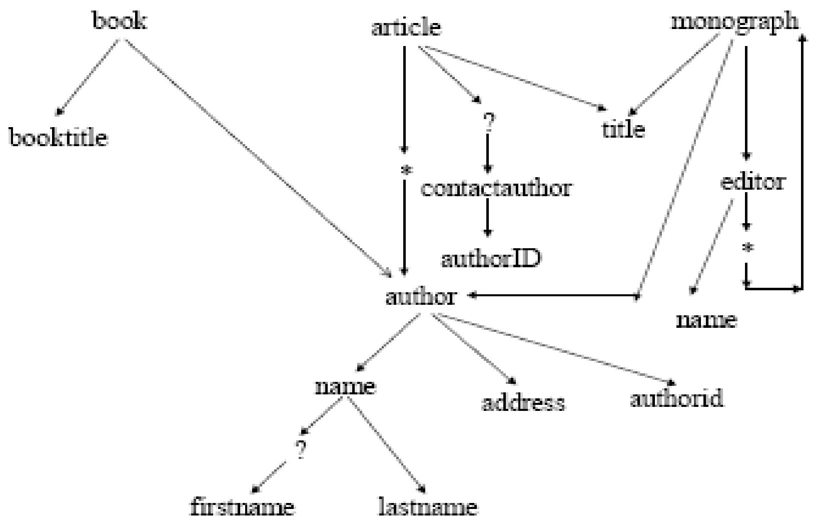


Figure 5.4 : DTD graph [98]

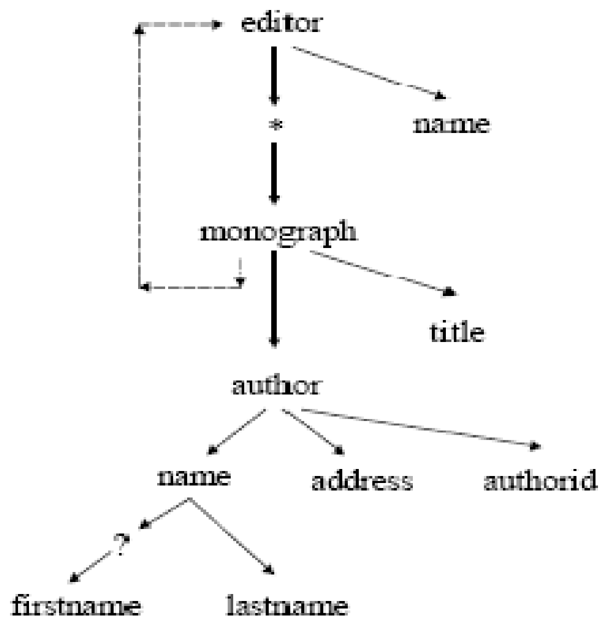


Figure 5.5: Element graph for the editor element [98]

4) Given an element graph, relations are created as follows. A relation is created for the root element of the graph. All the element's descendents are inlined into that relation with the following two exceptions:

a) Children directly below a "*" node are made into separate relations – this corresponds to creating a new relation for a set-valued child.

b) Each node having a backpointer edge pointing to it is made into a separate relation – this corresponds to creating a new relation to handle recursion. List 5.2 shows the relational schema that would be generated for the DTD in 5.1.

```
<!ELEMENT book (booktitle, author)
<!ELEMENT article (title, author*, contactauthor)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, address)>
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address ANY>
```

List 5.1: Document Type Definition DTD [98]

book (bookID: integer, book.booktitle : string, book.author. name.firstname: string, book.author.name.lastname: string, book.author.address: string, author.authorid: string)

booktitle (booktitleID: integer, booktitle: string)

article (articleID: integer, article.contactauthor.authorid: string, article.title: string)

article.author (article.authorID: integer, article.author. parentID: integer, article.author.name.firstname: string, article.author.name.lastname: string, article.author.address: string, article.author.authorid: string)

contactauthor (contactauthorID: integer, contactauthor. authorid: string)

title (titleID: integer, title: string)

monograph (monographID: integer, monograph.parentID: integer, monograph.title: string, monograph.editor.name: string, monograph.author.name.firstname: string, monograph.author.name.lastname: string, monograph.author.address: string, onograph.author.authorid: string)

editor (editorID: integer, editor.parentID: integer, editor . name: string)

editor.monograph (editor.monographID: integer, editor . monograph.parentID: integer, editor.monograph.title: string, editor.monograph.author.name.firstname: string, editor.

monograph.author.name.lastname: string,editor. monograph.

author.address: string, editor. monograph.author.authorid: string)

author (authorID: integer, author.name.firstname: string,

author.name.lastname: string, author.address: string,

author. authorid: string)

name (nameID: integer, name.firstname: string, name. lastname:

string)

firstname (firstnameID: integer, firstname: string)

lastname (lastnameID: integer, lastname: string)

address (addressID: integer, address: string)

List 5.2: Relational schema resulted by Basic technique [98]

5.3.2 Shared Inlining Technique:

The Shared Inlining Technique, is attempt to avoid the drawbacks of *Basic* by ensuring that an element node is represented in exactly one relation. The principal idea behind *Shared* is to identify the element nodes that are represented in multiple relations in *Basic* and to share them by creating separate relations for these elements.[123]

1) Its principle idea is to identify the element nodes that are represented in multiple relations in Basic (such as firstname, lastname, address) and

to share them by creating separate relations for these elements.

2) Relations are created for all elements in DTD graph whose nodes have an in-degree greater than one.

3) Nodes with an in-degree of one are inlined.

4) Nodes having an in-degree of zero are also made separate relations, because they are not reachable from any other node.

5) Elements below a "8" node are made into separate relations.

6) The mutually recursive elements all having in-degree one (such as monograph and editor) one of them make a separate relation. To find mutually recursive elements, look for strongly connected components in the DTD graph.

7) After deciding which element nodes are to be made into separate relations, it is easy to construct the relational schema as follows:

a) Each element node X that is a separate relation inlines all the nodes Y that are reachable from it such that the path from X to Y does not contain a node (other than X) that is to be made a separate relation.

b) Inlining an element X into a relation corresponding to another element Y creates problems when an XML document is rooted at the element X. To facilitate queries on such elements we make use of is Root fields. List 5.3 shows the relational schema derived from the DTD graph of list 5.1.

book (bookID: integer, book.booktitle.isroot: boolean, book.booktitle : string)

article (articleID: integer, article.contactauthor.isroot: boolean, article.contactauthor.authorid: string)

monograph (monographID: integer,monograph.parentID: integer, monograph.parentCODE: integer, monograph.editor.isroot: boolean, monograph.editor.name: string)

title (titleID: integer, title.parentID: integer, title.parentCODE: integer, title: string)

author (authorID: integer, author.parentID: integer, author.parentCODE: integer, author.name.isroot: boolean, author.name.firstname.isroot: :boolean, author.name.firstname: string, author.name.lastname.isroot: boolean, author.name.lastname: string, author.address.isroot: boolean, author.address: string, author.authorid: string)

List 5.3: Relational schema resulted by Shared technique [98]

5.3.3 Hybrid Inlining Technique:

The Hybrid Inlining Technique, or *Hybrid,* is the same as *Shared* except that it inlines some elements that are not inlined in *Shared*. In particular, *Hybrid* additionally inlines elements with in-degree greater than one that are not recursive or reached through a “*” node. Set subelements and recursive elements are treated as in *Shared*.[123]

1) It combines the join reduction properties of Basic with the sharing features of Shared.

2) It is the same as Shared except that it inlines some elements that are not inlined in Shared.

3) It additionally inlines elements with in-degree greater than one not recursive or reached through a “*” node.

4) Set sub-elements and recursive elements are treated as in Shared.

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

Redundancy reducing XML storage in relations (RRXS) within XML Functional Dependency (XFD) is proposed by Yi Chen et al., 2003, at University of Pennsylvania and Universidade Federal do Parana, Brazil [99], as a constraint definition to capture structural constraints as well as semantic information. XFD are used to describe the property that the values of some attributes of a tuple uniquely determine the values of other attributes of the tuple, which is different from relational database in that they must be defined using path expressions. The path language used in XFDs and for XML tree navigation allows traversal along the child (/) and descendant (//) axis. So they defined their path language, $XP^{\{/,//\}}$ by the following grammar [99]:

$PL_1 \rightarrow \ell \backslash PL_1/PL_1 \backslash PL_1//PL_1$

$PL_2 \rightarrow \in \backslash PL_1 \backslash /PL_1$

$PL \rightarrow PL_2 \backslash PL_2/value()$

$SSXP^{\{/,//\}} \rightarrow PL \backslash \$x/PL$

Where ℓ dSenotes an XML node label (element tag or attribute name), denotes the empty path, value() retrieves the value of the context node (only applicable for a leaf node), and $x is a variable bound to a path expression in PL.

A reduced set of the input XFDs is used to guide the design of the target relational schema by translating XFDs to relational functional dependencies and creating a third normal form (3NF) decomposition. In this way, XFDs are mapped to relational keys and relational primary key technology used to validate semantic constraints. In addition, redundant information in the XML document as expressed in XFDs is reduced in the relational design, and the use of node ids is reduced wherever value-based keys exist.

In order to do this, they defined three algorithms, algorithm 1 (5.4): a polynomial time algorithm (function infer), which given an XFD Ø: X > Y and a set of XFDs F, determines whether or not Ø can be inferred from F using L (L represents XFDs defined over XML data). Given an initial set of XFDs, this algorithm is then be used to detect which XFDs can be eliminated and which ones can be simplified by eliminating P-attributes on their left hand sides, thereby deriving a reduce set G of XFDs F.

Algorithm 2 (List 5): RRXS function, takes an XML schema (a set of XFDs, F) and optional DTD (D), and generates a normalised relational schema (R) with a set of keys (K) as well the instant transformation program (M). The transformation M will map an XML tree T which conforms to D and satisfies F to relations M(T) which conform to schema R. Every node is assigned a unique node id or it is identified by a semantic key value to guarantee that the parent-child connections between nodes are preserved. Removing of redundant node id is based on the following observation: if X > Y and Y = X then X and Y are functionally equivalent. Equivalence will recognize equivalent

XFDs and equivalent elements (an element is a set of P-attributes which appear on the left or right side of an XFD), then group those elements into equivalence classes and output G. In the second step, the reduced set H of G is computed to remove redundant XFDs. Then for each equivalence class, shrink removes unnecessary elements, producing the set of XFDs I. During the fourth step, every non-equivalent P-attribute p in I is mapped into a relational attribute p_a to record the ids or values of the nodes reachable by p, and I is mapped into a set of functional dependencies I_R. Finally, a third normal form (3NF) target relational schema R is generated based on I_R. The optional XML schema information D can be used to automatically generate structural XFDs, and also used in the path containment test in the reduced cover algorithm.

```
Function infer
Input: $(F, \varphi : X \to Y)$
Output: True, if $F \perp L\varphi; False, otherwise$
Let I = max{lengths( $\$v/Q \mid \$v/Q$ is a P-attribute of some XFD in
F r $\varphi$}
If $\forall \$v/Q \in X$ satisfies the singleton condition then
```

```
S=X
If $\exists\$v/P \in X$ and P does not end with value( ) then
$S = S \cup \{\$v\}$
 End if
 Else
 return false
  end if
  if $Y \in S$ then
  return true
 end if
 repeat
 $B^{'} = Null$
 For each XFD $\phi : A \rightarrow B \in F$ do
  If $\forall\$v/R, \in A, \exists\$v^{'}/R, \in S$ not marked by $(\phi,\$v)$,
   and expand $(\$v^{'}) \leq$ expand ($v) then
   $B^{'}$ = replace each $v with $\$v^{'}$ In B
  Mark $\$v^{'}$/ R, by $(\phi,\$v)$ if the variable of B is $v,
   or $v is dependent variable.
   Else if $\forall\$v/R, \in A, \exists\$v^{'}/R^{'}, \in S$ not mark by
```

```
        $(\phi,\$v)$, expand $(\$v'/R',) \leq$ expand ($v/R), and
        the path between $R'$, and R , satisfies the singleton
        condition then
        $B'$ = replace each $v with $(\$v')$ and adjust the path under
    $(\$v')$.
        Mark $\$v'/R', by(\phi,\$v)$ if the variable of B is $v,
        or $v is the dependent variable.
        End if
        If $B' \neq Null \quad and |B'| \leq l then$
        $S = S \cup \{B'\}$
        End if
        If Y = $B'$ then
        Return true
        End if
        End for
        Until S does not enlarge
        Return false
```

List 5.4: Algorithm 1, infer function [99]

```
Function RRXS
Input: F, optional DTD D
Output: R with K defined ,M
G==Equivalence(F,D)
H=Reduce(G,D)
I=Shrink(H)
Map each distinct P-attribute p in I to an attribute $p_a$
M=$\bigcup_{\forall p \in I} (p_a \leftarrow p)$
Let A be the set of attributes obtained
Map I to functional dependences $I_R$ over A.
Generate a 3NF relational schema R over the attribute
set
A  according to $I_R$
Return R,M
```

List 5.5: Algorithm 2, RRXS function [99]

Algorithm3 (list 5.6): Equivalence function is used to find relationship between variables and P-attributes to recognise redundant node ids. It consists of two steps to achieve its purpose [99]:

(1) If two XFDs $Ø_1$ and $Ø_2$ satisfy $Ø_1 \supset Ø_2$ and $Ø_2 \rightarrow Ø_1$, then the one

will be chosen is that minimizes the number of variables used for a given set of XFDs;

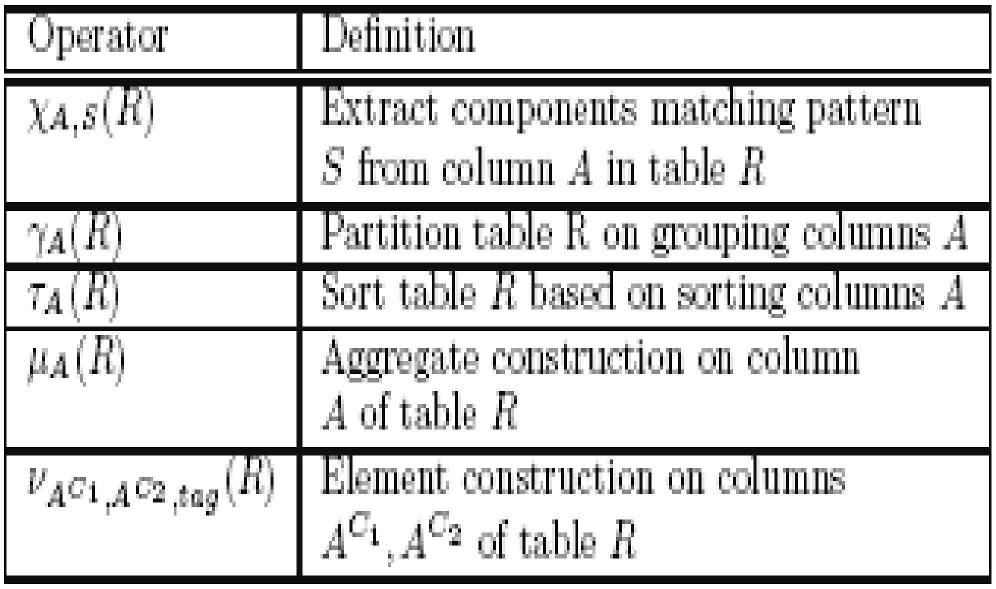
(2) If there are two XFDs Ø$_3$: X $\cup$ Y and Ø$_3$: Y $\supset$ X then elements X and Y are grouped into an equivalence class.

```
**Function Equivalence**
Input: F,D
Output: G
Construct C using the unique child and unique parent
XFDs in F
$F' = F - the \quad XFDsused \quad to \quad construct \quad C$
$n = |F'|$
For i= 1 to n do
Let $\phi_i, be \quad X \quad \rightarrow Y$
If $\forall X_k, \in X, \exists(X'_k \in C \quad or \quad X'_k, \in \phi; i \prec j \leq n)$ such that: expand
(Node Path ( $X'_k$)) $\equiv$ expand (Node Path($X_k$) )then
If $X \rightarrow Y \Leftrightarrow X' \rightarrow Y'$ then
Replace $\phi_i$ ,with $X' \rightarrow Y'$
End if
End if
End for
For i=1 to n do
Let $\phi_i , be \quad X \to Y$
If ((expand(X)$\in$PL$_2$ and $|X| = 1$)or expand(Y)
$\in$PL$_2$) and infer ($F, Y \to X$) then
Put X and Y into the same equivalence class C,
Remove $\phi_i$
End if
End for
$G = (F^{'}, C)$
Return G
```

List 5.6: Algorithm 3, Equivalence function [99]

However, using XFDs and nodes ids together leads to some information loss. In order to overcome this information loss, documents must be completely covered by XFDs. Unfortunately; the suggested rewrite rules are not complete. So, this algorithm will not be sure to reduce the redundancy.

is proposed by Hui Zhang and Frank Wm Tompa, 2004, in University of Manitoba and University of Waterloo[100]. In this approach, user's interference is needed to typically first "shred" their documents by isolating what they predict to be meaningful fragments, then store the individual fragments according to some relational schema, and later translate each XML query (expressed in XQuery) to SQL queries expressed against the shredded documents. So, they defined an extraction operator, $X_{A,S}$(R), adapted from function extract_subtexts() designed for a text-relational abstract data type (this operator takes a table R as input and two parameters, A and S, where A is a column of table R of type text and S is a tree pattern to match against each text entry in the given column A), in order to enable dynamic shredding of XML data. The pattern matching language is a variant of XPath that describes tree patterns instead of path patterns, using hash marks or some similar flags to indicate which nodes are to be returned. Therefore, it differs from an XPath expression by identifying several nodes in a tree that correspond to a single match rather than extracting only the last node in some path. Their algebra is an extended relational algebra based on SQL tables rather than relations with support for text function in order to convert data to string. They used traditional relational operators, selection, join conditions, and projection list which may include text functions as well as non-standard operator which shown in Table 5.2.



Operator	Definition
$\chi_{A,S}(R)$	Extract components matching pattern S from column A in table R
$\gamma_A(R)$	Partition table R on grouping columns A
$\tau_A(R)$	Sort table R based on sorting columns A
$\mu_A(R)$	Aggregate construction on column A of table R
$\nu_{A^{C_1},A^{C_2},tag}(R)$	Element construction on columns A^{C_1}, A^{C_2} of table R

Table 5.1: List of operators [101]

They described these operators as follows [101]

1. Extraction: Let *S*(*COLS, PRE COLS,OPS,* Φ*,F COLS*) be a tree matching pattern, where *COLS*=<col_1, col_2, ... , col_n> is the vector of items in XPath terminology (renamed if needed for uniqueness); *PRE COLS*=<p_1, p_2, ... , p_n> is the vector of the closest ancestor of each item in *COLS*; *OPS*=<op_1, op_2, ... , op_n> is the vector of child or descendent operations on items, i.e., a '/' or a '//' operation; Φ=<$\square_1$, $\square_2$, ... , $\square_n$> is the vector of text matching constraints on each item (if there is no condition applied on a particular item, an always-true condition is assumed); *F COLS* is the vector of items with associated flag #. Their

extraction operator(χ) with a pattern S on column A of table R can be formally defined as:

$$\chi_{A,S(COLS,PRE\_COLS,OPS,\Phi,F\_COLS)}(R)$$

$$= R \blacktriangleright\blacktriangleleft_{A} (\pi_{F\_COLS'F\_ COLS'\_ \square\{A\}}(\square_1(ex(A, col_1, op_1))$$

$$\blacktriangleright\blacktriangleleft_{pre(col2,S)} (\square_2(ex(pre(col_2, S), col_2, op_2))$$

$$\blacktriangleright\blacktriangleleft_{pre(coln,S)} (\square n(ex(pre(coln, S), coln, opn))))$$

Where F *COLS' is* the vector of hidden columns corresponding to *F COLS*.

2. Sorting: A sorting operator τ is used to sort a table according to some sorting criteria. $\tau A(R)$ takes a table R as input and a list of sorting columns A as a parameter. The result of this operation is the table R, but with the rows sorted in the order indicated by A.

3. Groupby: For a grouping column *col* of simple type, the partition operator (γ) partitions a table such that each row in a partition has the same value for *col*; if the grouping column is of type text, it partitions the table based on the value of the (hidden) node identifier of *col*. Thus the groupby operator can be used to partition a table based on simple values or instead on node identity when a text column is specified.

4. Construction operator: to support the conversion of parts or whole relational tables into documents, two construction operators are

included:

a) Aggregate constructor (*μA*) is used mainly for representing the contents of a set-valued column for several tuples as a single tree. Assuming that a groupby operation is performed first, instead of computing an aggregated scalar value for each group, aggregate construction forms a tree from the values over column *A* in each group, appropriately handling *null* values. Their catenate (∘) operator is defined as follows: If $T_1 = T(t_1, a_1, < e_1, e_2, \dots , e_m >)$ and $T_2 = T(t_2, a_2, <f_1, f_2, \dots , f_n>)$ are two text values and [101]

$$E = \begin{cases} <e_1, e_2, \cdots, e_m, f_1, f_2, \cdots, f_n> \\ \qquad \text{if } t_1 = vector \text{ and } t_2 = vector \\ <e_1, e_2, \cdots, e_m, T_2> \\ \qquad \text{if } t_1 = vector \text{ and } t_2 \neq vector \\ <T_1, f_1, f_2, \cdots, f_n> \\ \qquad \text{if } t_1 \neq vector \text{ and } t_2 = vector \\ <T_1, T_2> \\ \qquad \text{if } t_1 \neq vector \text{ and } t_2 \neq vector \end{cases}$$

Then $T_1 \circ T_2 = T(vector, null, E)$.

and the aggregate constructor *μA* is defined as follows:

Let $v_1, v_2, \dots , v_n$ be the list of values in a group on column *A*. Applying the aggregate constructor on this group generates the text value $v_1 \circ v_2 \dots \circ v_n$, and *μA* is the result of applying an aggregate constructor to each group in the table. Note that the order of the subelements of the result is defined to match the order of the rows in the table.

b) *Element constructor* (v) is another construction operator. It takes three parameters, A^{C1},A^{C2}, and *tag*, where (1) A^{C1} is a list of columns that are to become the XML attributes of the resulting element being constructed, here denoted *elet*. The order that columns occur in the list A^{C1} does not matter. (2) A^{C2} is a list of columns that are to become the subelements of *elet*. The order that columns occur in the A^{C2} list is also the order that these subelements occur in the resulting element. (3) *tag* is the tag name of *elet*. By default, the column corresponding to *elet* in the resulting table is named *tag*, but if *tag* conflicts with an existing column name, some renaming is necessary.

Element constructor is applied to each tuple and the result is computed as: concatenate the value T_i of each column i appearing in A^{C2} to construct a tree T *with* all T_i as children. Set the tag of the result to *tag* and the (XML) attributes of the result to the named set of values in A^{C1}, where names correspond to the (table) attribute names in A^{C1}. Since the order of each column appearing in A^{C2} is important, we apply the *catenate* operator in a way such that each child is in the same order as it is in A^{C2}. This can be formally defined as: For a single row in a table, let c_{1j} be the value of column C_{1j} for each C_{1j} in A^{C1}; and let $c_{21}, c_{22}, \dots, c_{2m}$ be the values of columns in A^{C2}. Applying the element constructor to this row produces the text value $T(tag, \{C_{1j} = c_{1j} \mid C_{1j} \square A^{C1}\}, c_{21} \circ c_{22} \circ \ldots \circ c_{2m})$. Finally, $v(tag, A^{C1}, A^{C2})$ is the result of applying such element constructor to each row of the table.

To translate XQuery to relational algebra the following procedure is followed [101]:

1) Given an XQuery Q, XML query processing framework first canonicalizes it to a query Q'.

2) Translate from Q' to an extended relational algebra with support to text, using the translation algorithm described below.

3) After obtaining an initial query plan from translation, query rewritings (see [102]) to optimize it to get better plan, which is then sent to the underlying extended database management system to be executed.

5.4 Translation algorithm:

Given XQuery Q in the canonical form (Q'), represents it as a query tree. A query tree has four kinds of nodes: every internal node is called a CF node and represents a FLWOR expression or a compound return value; every leaf node represents a simple return value and is either a V node to denote a for-variable, U node for a let-variable, or aggU node for an aggregate function applied to a let-variable. The subtrees of each CF node represent either nested FLWOR expressions that are bound to let-variables or the components of the return clause, in the order of their appearance in the XQuery expression. In the former case, the incoming edge is labelled with 'let' and this CF node is referred to as a let-CF node. The operation of the translator of XQuery in canonical form (Q') to relational algebra expression will be as follows:

Let *Q* be a query in basic XQuery canonical form, and *QT* be the query tree for *Q*. Note that *QT* has only one CF node (i.e., root) with some leaf children as returned values. Trans0 proceeds as follows by considering query information and the returned values associated with the CF node (Trans0: Basic CF translation) [101]:

1. Translate for clause: extract binding values for each for clause \$vi := FEi using the extraction operator χ. FE_i is a path expression beginning with either document() or a reference to another variable, which determines the source for the extraction. The rest of the path expression is translated to a pattern matching string, and supplied as the second parameter to χ.

1.1. If FE_i starts with the document() function Trans0 selects the corresponding row and column of the initial table R^0 to form a new one-row, one-column table R^1 containing the document text *doc*, then extracts $\$v_i$ from R^1 using the tree pattern corresponding to FE_i. When executed, this produces a new column together with a hidden mark column on R^1.

1.2. If FE_i starts with another variable, Trans0 must have previously extracted a corresponding column in some table. In this case, extraction starts from that column and forms a new column together with a hidden mark column for that existing table. Whenever distinct is present, duplicate elimination is performed based on value or node identity as desired, depending on the specification of distinct. After this step, each variable corresponds to a column from which the variable takes its binding values, and a hidden mark column indicating where these binding values originate. Since these mark columns are used only by the underlying DBMS, we use the term 'column' to mean 'visible column' unless specified explicitly.

2. Translate let clause: similar to step 1 except that after each extraction, perform a sorting operation on all the remaining columns (except let-columns) based on document order, then perform a partition on all columns except the newly extracted one, followed by an aggregate constructor on the newly extracted column. Thus each let-variable's value is a text tree representing a collection as a vector. To ensure compliance with XQuery semantics, the order of elements in the grouping column list and the sorting column list is the same as in the query.

3. Form a single table: compute the cross product of multiple tables, if any, obtained from the previous steps. Let the resulting table be $R(a_1, a_2, ..., a_n)$.

4. Translate where clause:

4.1. Rewrite the where condition such that variable appearances are replaced by their corresponding column names in *R*. Denote the rewritten condition as *WC*.

4.2. Include a selection operator σ with condition *WC*.

5. Translate return clause:

5.1. Project on columns corresponding to for-variables, let-variables appearing in the return clause, plus those aggregate functions applied on let-columns. If a returned variable has a tag around it, then the projection list includes an element constructor applied to that column.

5.2. Sort the table according to document order or as the query requires. In the case of sorting on document order, the sorting column list is the for-variable list with each for-variable in the order of its appearance in the for–clause, and the sorting is performed on the hidden mark column associated with each for-variable. (This corresponds to W3C's specification for ordering.)

5.3. Apply an element constructor using the columns of the previous step. Its parameters are supplied as indicated by the return clause, with the columns in the second parameter (i.e., subelements list) in the order of their corresponding variable appearances in the return clause and the third parameter as *return-tag* if present. Let the resulting column be named *a*.

6. Generate the result: Project on column *at* and apply an aggregate construction operator on *at* treating all rows as a single group, followed by an element constructor adding the tag *result-tag* or *vector* on the aggregated value of *at*. Hence, the result of this step is a one-row, one-column table containing a constructed text tree *T*.

To extend the translation of basic XQuery expressions to expression with or without nesting in return and let clauses, Let the table *R* in the translation of a basic query be denoted as the *working table*. In the extended translation for general queries, the working table is global to all nested FLWOR expressions.

Trans [101]: XQuery of translation: Let *Q* be an XQuery in canonical form, *QT* be the query tree for *Q*. For each CF node in the query tree *QT*, visited in depth-first order, Trans applies an extension of the basic translation Trans with the following modifications:

Step 1. Save the current working table *R* as *S*.

Step 2’. Translate let clause. If the let clause is a simple variable binding without nested CF, its translation is the same as that in basic translation. Otherwise, call Trans on the nested CF node to create a column containing the sets of values to which the let variable is bound.

Step 3’. Denote the extracted for- and let-columns for this current CF node as CC, all previously extracted for- and let-columns for nested subqueries as RC, and all previously aggregated columns (including both scalar aggregated columns and aggregated construction columns) for nested subqueries as AC.

a. If there are nested CF nodes in the return clause, call Trans on the nested CF nodes.

b. If processing a nested CF node, replace *R* by the left outer join of *S* with *R*. T his step is to ensure that empty substructures will be generated, if needed, precisely where they are required by XQuery semantics.

c. Similar to Step1 but with the projection list enlarged to include all columns in RC $\equiv$AC along with those required in the construction of this CF node.

d. Sort the table as specified by the query, or (if no sorting is specified) sort the table with the sorting column list including all columns in RC *to* CC (except all let-columns) with each for-variable in the order of its appearance in *Q*. As before, sorting is performed on the hidden mark column associated with each for-variable to ensure correct document order.

e. Same as step 3

Step 4. Partition on all columns in RC *and* AC, construct the result of this CF node in the same way as Step 4, and put the result into a new column in table *R*.

It just converts subtexts to relational fields as needed dynamically in response to user queries, and keeps the original XML text untouched. This approach solves the problem of the document size limitation, and the loss of information from the original XML data since it keeps the original document untouched. But not saving the XML document in the relational database will make it impossible to connect with the data already existing in the relational database. Also there is a need for query translation for every XQuery.

SPIDER (Schema based Path IDentifiER): a node labelling scheme is used by Kei Fujimoto et al., 2005, in Nagoya University and Nara Institute of Science and Technology [103] to preserve XML tree structure. SPIDER is a scheme that uniquely numbers all paths from root node appearing in documents under a DTD by referring to information distilled from the DTD. It assigns a unique integer to a sequence of elements and/or attributes names from the root node to a node in an XML tree. SPIDER only identifies paths from the root node to a node in an XML tree; it could not distinguish between multiple nodes appearing in the same path. It is calculated as follows: (1) A table, called StruDTD, is created to enumerate all parent-child relations in DTD. StruDTD, Table 5.2, has three columns: Parent, Child and cOrder. Parent contains the names of parent nodes, and Child contains the names of child nodes of the parent nodes. cOrder is an integer assigned so that a pair of any two columns identifies the remaining column. (2) SPIDER is calculated by referring to cOrder in StruDTD. Let *p.sid* denote SPIDER of parent node, let *c.sid* denote SPIDER of child node, let *f* denote maximal value of cOrder and let childOrd(*p.tag, c.tag*) denote cOrder, where the value of Parent is *p.tag* and the value of Child is *c.tag* in the same row. SPIDER is calculated recursively by the following expression. (The SPIDER of the root node is 1.)

$$sid.c = (\mathrm{sid.p} - 1) \times \mathrm{f} + 1 + childOrd(\mathrm{tag.c}, \mathrm{tag.p})$$

Figure (5.6) shows an example for labelling nodes in XML documents using SPIDER depending on StruDTD table.

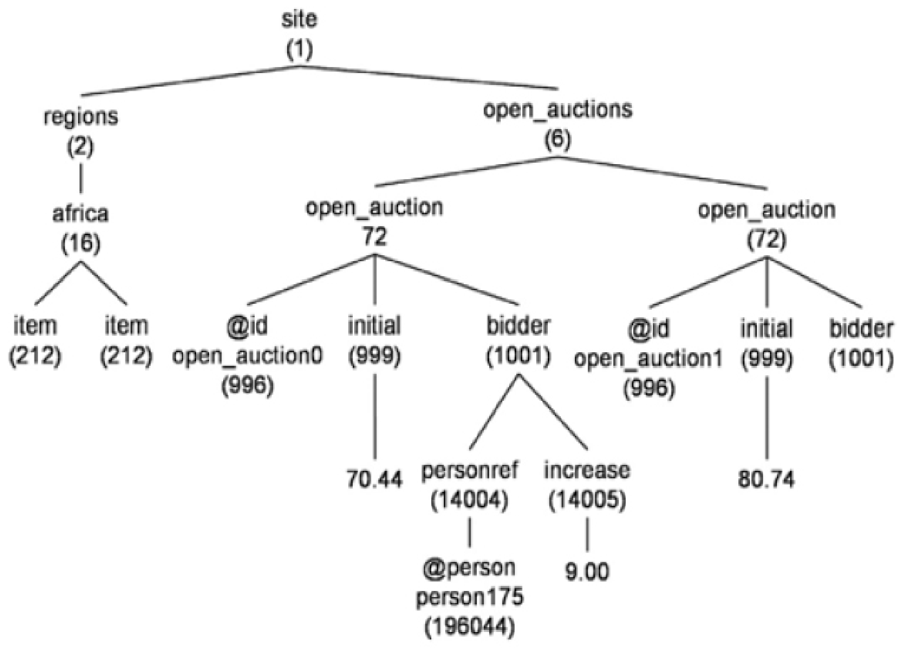


Figure 5.6: Node labelling using SPIDER [103]

To solve this issue they introduced Sibling Dewey Order to identify such nodes. When a new node is to be inserted into an XML document, there is a need to re-label some other nodes to preserve the order of nodes. In this method, only Sibling Dewey Order is relabelled where SPIDER is not affected. Sibling Dewey Order is calculated as follows. First, get the order of the nodes that have the same path from the root node and the same parent node. Then, concatenate the order from the root node to the target node. The ancestor-descendant relationship between any two nodes is checked efficiently by the concatenation. Figure (3.6) shows node labelling using SPIDER and Sibling Dewey Order method.

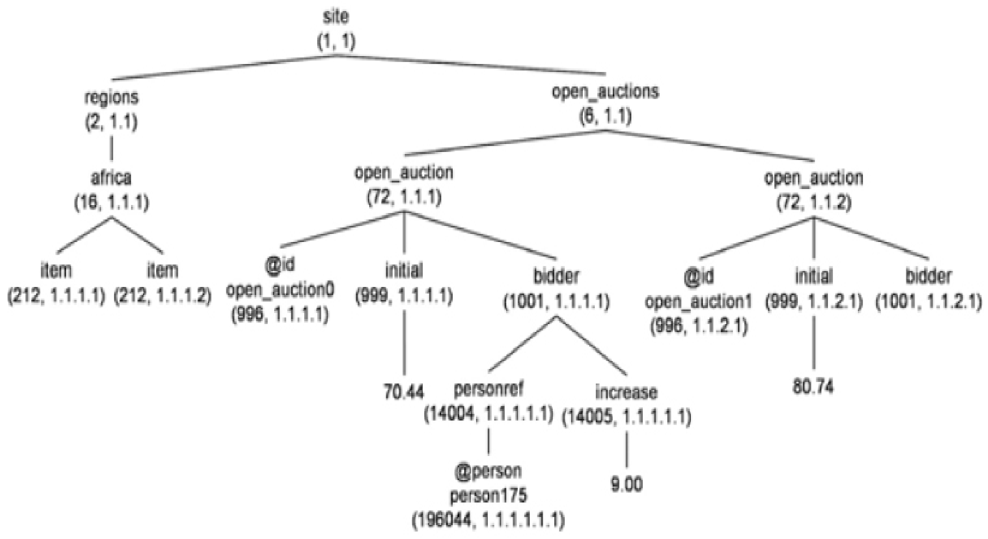


Figure 5.7: Node labelling using SPIDER and Sibling Dewey Order. [103]

Parent	Child	cOrder
Site	Regions	1
Site	Categories	2
Site	Catgraph	3
Site	People	4
Site	open-auctions	5
Regions	Africa	1
Africa	Item	1
open-auction	Open-auction	1

Table 5.3: A part of StruDTD [103]

In order to store XML documents enumerated with SPIDER and Sibling Dewey Order, the following schema are defined [103]:

Element (docID, nodeID, spider, sibling, parentID)

key is (docID, nodeID)

Attribute (docID, nodeID, spider, sibling, parentID, value)

key is (docID, nodeID)

Text (docID, nodeID, spider, sibling, parentID, value)

key is (docID, nodeID)

Path (spider, path, pathexp)

key is (spider)

"Element" is a relation to store element nodes, "Attribute" is a relation to store attribute nodes, "Text" is a relation to store text nodes, and "Path" is a relation to store all paths from the root node to a node in XML documents.

And the following describes the attributes of the above schemas [103]:

docID Identifier to identify XML documents.

nodeID Serial number to identify each node. The order among this serial number has no sense. This number is not updated.

spider SPIDER assigned to nodes. In the Text relation, this attribute stores SPIDER assigned to parent nodes.

sibling Sibling Dewey Order assigned to nodes. In the Text relation, this attribute stores Sibling Dewey Order assigned to parent nodes.

parentID nodeID of parent node.

value Attribute value in Attribute relation, and content of element node in Text relation.

pathexp Path expressions appearing in all XML documents.

Table 5.3 shows the relations storing XML documents in Figure 5.7.

In this mapping scheme, string matching is used to handle the path that contains "//". This matching requires a join between the "path" and "element" relations in such a scheme. This causes the performance to get worse. They cannot also preserve node order exactly by using a pair of SPIDER and Sibling Dewey Order if DTD declaration contains multiple components having the same name but appearing in different places. On the other hand, using node indexing will result in a large extra space compared with the real data, it is impossible to index a document with a large number of nodes. This also results in time consumed to reconstruct the original XML document.

Element

docID	nodeID	Spider	Sibling	parented
1	1	1	1	1
1	2	2	1.1	2
1	3	16	1.1.1	3
1	4	212	1.1.1.1	3
1	5	212	1.1.1.2	1
1	6	6	1.1	6
1	7	72	1.1.1	7
1	9	999	1.1.1.1	7
1	11	1001	1.1.1.1	11
1	12	14004	1.1.1.1.1	11
1	16	14005	1.1.1.1.1	6
1	18	999	1.1.2.1	18
1	20	1001	1.1.2.1	18
1	22	1002	1.1.2.1	11

Attribute

DocID	nodeID	Spider	Sibling	ParentID	value
1	8	996	1.1.1.1	7	Open_auction0
1	13	196044	1.1.1.1.1.1	12	Peron175
1	18	996	1.1.2.1	16	Open_auction1

Text

docID	nodeID	Spider	Sibling	ParentID	value
1	10	999	1.1.1.1	9	70.11
1	15	14005	1.1.1.1.1	14	9.00
1	19	900	1.1.2.1	18	87.55
1	10	453	1.1.1.1	9	56.21
1	15	14005	1.1.1.1.1	14	9.00
1	10	999	1.1.1.1	9	70.11
1	15	1001	1.1.1.1.1	14	7.00

Path

Spider	Pathexp
1	#/site
2	#/site#/regions
16	#/site#/regions#/Africa
212	#/site#/regions#/Africa#/item
6	#/site#/open_auctions
72	#/site#/open_auctions#/open_aution
999	#/site#/open_auctions#/open_aution#/initial
1001	#/site#/open_auctions#/open_aution#/bidder
14004	#/site#/open_auctions#/open_aution#/bidder#/personref
196044	#/site#/open_auctions#/open_aution#/bidder#/personref#/@preson
14005	#/site#/open_auctions#/open_aution#/bidder#/increase

Table 5.4 :An example of storing XML document in relations [103]

Indexing a group of XML nodes methods are proposed by Guangming Xing et al., 2005, in Western Kentucky University, University of Michigan-Dearborn and Syracuse University [104]. These methods include: using path information to refine the storage, indexing a group of XML nodes instead of each individual node, and query evaluation based on the "nodes of interest". They are used to reduce the extra space needed for indexing nodes in which a DTD is mandatory in order to use the index schema. A fewer tables to be created results in reducing the number of path join needed to process the query. This will cause the performance of query to get better. Based on this idea, a set of related nodes may be grouped and stored together in a table instead of storing each type of nodes separately. To do so, the following assumptions are taken into the consideration: (1) Each XML element consists of a collection of attributes and sub-elements. For each attribute, it is required, implied, or optional. For each sub-element, it may either appear exactly one time, optional, or many times (zero or more times, one or more times, etc). This information is stated in DTD or XML schema. DTD or XML schema may still be helpful for generating mapping to tables in relational database. (2) An elements can be stored together with its sub-elements as long as certain constraints are satisfied. (3) Based on the relation with its parent, each node can be classified as fixed node if a node occurs exactly one time, multiple nodes if a node may appear many times, or optional node if this node may appear one time or zero time. (4) A leaf node is defined to be a node if it is either an attribute of an element, or an element that does not have any sub-elements and attributes.

Based on the assumptions above, their mapping algorithm, 5.7, for DTD to tables in a relational database is defined as follows:

```
Algorithm Node Grouping
Input: A DTD
Output: List of Grouped Nodes
Function Main
    Queue q
    q.Enqueue(root)
    while not q.empty()
        p = q.Dequeue()
        list.addLabel()
        list = CalcGroup(p)
        list.Output()
    EndWhile
end
```

```
Function CalcGroup(node)
    list = CreateEmptyList()
    list.SetName(p)
    for each child c of p
        if c is fixed and c is leaf
            list.Add(c)
        else if c is optional and c is leaf
            list.Add(c)
        if c is fixed and c is not leaf
            list.Merge(CalcGroup(c))
        else
            q.Enqueue(c)
        fi
    endfor
    return list
end
```

List 5.7: Node Grouping Algorithm [104]

Query processing using nodes grouping indexing: as the nodes in an XML document are grouped, so there are two possibilities for the nodes appearing in one XPath [104]:

1) They are grouped and stored in one table; or

2) They appear in two different tables, and are related with each other by the label field in the table.

The following example illustrates how a regular path expression can processed under this indexing scheme:

“//catalog[name=”CS”]//author[name=”David”]

As the information about catalog except the books are stored in one table, finding the catalog with name “CS” will be the same as finding a tuple in a relational database, which can be handled by the following SQL query: select * from catalog where name = ’CS’; Similarly, authors with the name of “David” can be found. select * from authors where name = ’David’; the element join of catalog and author could be done similarly as before by determining whether or not the label of an author properly covers the label of a catalog”. In order to improve the efficiency of query processing two techniques are used [104]:

a) Refined Storage with Path index: Instead of storing a path label for each node, the union of all path labels is used. Based on this idea, the nodes have the same path labels are stored in the same unit. Schema tree is used to represent all the path labels. The number of path labels in an XML document is equal to the number of nodes in the schema tree. There are several advantages of using the path index to refine the storage for an XML document:

1. The path joins between the nodes that are impossible to be paired are avoided. For example, as the name for author and the name for catalog are stored in different units, so the path joins between an element from //catalog/book/author/name and //catalog are avoided.

2. The child axis can be handled without introducing level information. As in existing range-based techniques, parent-child relation can't be handled without introducing level information. For example, level information is kept in the storage of each node in XISS. However, when the path index is used, the level of a node is just the length of the path from the root.

3. Another advantage is that path information could be used for query optimization, which will be discussed in the remaining parts of this section by the introduction of "nodes of interest".

b) Nodes of Interest: they used a solution that only does paths joins if it is absolutely necessary by the introduction of "nodes of interest", which is defined as:

1. The nodes in a XPath expression with a predicate as long as the predicate is not trivial (selecting all or selecting nothing are trivial);

2. The nodes in the output statement;

3. Those nodes that are needed to relate the nodes defined by the above two rules.

Using grouping method reduces the space used for labelling and makes the reconstruction of the original documents of XML easier. Introducing nodes of interest reduce the number of path joins needed to process the query.

An algorithm for mapping DTD to relational schema is prposed by Zijing Tan et al., 2005, in Fudan University [105]. This technique preserved not only the content and structure but also the semantics of original XML documents. To tackle the problem of constraint expression, they introduced a way to define functional dependencies and normalization for DTD. In a normalized DTD, every constraint expressed by functional dependencies can be concluded to keys. So they used the key definitions for XML as the foundation for relation generation, and maintain the keys in relations. After investigating the relationship between functional dependencies in XML documents with the corresponding ones in relations; they further proved that if the original DTD is normalized, the generated relations will be in BCNF. The following notations, theories and definitions are given in order present their mapping algorithm, Algorithm 1, of DTD to relations:

Definition 1: “A DTD is defined to be D = (E, A, M, N, r), where:

1) E is a finite set of element types.

2) A is a finite set of attribute types, and there is a special attribute id $\in$ A.

3) M is a mapping from E to string or element type definitions with a regular expression $\alpha ::= \varepsilon | e' | \alpha | \alpha | \alpha, \alpha | \alpha^*$. Here ε is the empty sequence, $e' \in E$, and "|",","," and "*" denote union, concatenation and the Kleene closure.

4) N is a mapping from E to the *powerset of* A, and $\forall e \in E$, id $\in$ N(e).

5) $r \in E$, is called the element type of the root."

Definition 2 "Given a DTD D = (E, A, M, N, r), an XML document conforming to D is modelled as: 1) Here v denotes node, V denotes the finite set of nodes. 2) For each v, name(v) $\in E \cup A$. We further define two subsets of V: $Ve = \{v \mid v \in V, name(v) \in E\}$; $Va = \{v \mid v \in V, name(v) \in A\}$. 3) $\forall v \in Ve$, subelem(v) is a list=$[v_1, v_2, \ldots, v_n]$($vi \in Ve$). If name(v) = e and M(e) = α, name(v_1),name(v_2),. . .,name(v_n) is in the alphabet of α. 4)$\forall v \in Ve$, attr(v) is a set=$\{v_1, v_2, \ldots, v_m\}$ ($v_i \in Va$). If name(v) = e, for each $a \in N(e)$, there is a unique v_i where name(v_i) = a. 5) For each v, value(v) $\in \{S\}$. If name(v_1) = id, value(v_1) is unique across the whole document. 6) There is one and only one special node root, and name(root) = r."

Definition 3 "Given a DTD D, a path expression P over D is defined to be $\rho_1/\rho_2 . . ./\rho_n$. Here $\rho_1 \dots \rho_n \in$ E and $\rho_n \in$ E ∪ A. For i = [2, n−1], $\rho_i \in$ is in the alphabet of M(ρi−1), and $\rho_n \in$ is in the alphabet of $M(\rho_{n-1}) \cup N(\rho_{n-1})$. As a special case, an empty path expression is denoted as ε. Given an XML tree X conforming to D and a node v in X, when ρ_1 is in the alphabet of M(name(v)) ∪ N(name(v)), $\lfloor v\{P\} \rceil$ is defined to be the node set of P. If there is a node sequence $(v_0, v_1, \dots , v_n)$ in X,(v_i is the child node of v_{i-1}, v_0=v), and name(v_i)=ρ_i, $v_n \in \lfloor v\{P\} \rceil$. Specially, when there is only one node in $\lfloor v\{P\} \rceil$, we use v{P}to denote this node. $\lfloor root\{P\} \rceil$ is abbreviated as $\lfloor P \rceil$.

First(P) is defined to be the first element or attribute type of P, and last(P) is defined to be the last element or attribute type of P. If First(P) is in the alphabet of M(r) ∪ N(r), P is called root path expression. Let P = $\rho_1/\dots/\rho_n$, Q=$\rho'_1\dots/\rho'_m$. If ρ'_1 is in the alphabet of $M(\rho_n) \cup N(\rho_n)$, $\rho_1/ . . ./\rho_n/\rho'_1/ . . ./\rho'_m$ is called the concatenation of P and Q, denoted as P/Q. If R = P/Q, we say P is a prefix of R, denoted as $P \subseteq R$. If $R \subseteq P$, and $R \subseteq Q$, R is called a common prefix of P and Q. If for any common prefix R' of P and Q, $R' \subseteq R$ always holds, we say R is the maximum common prefix of P and Q, denoted as MCP(P,Q).

If e, e' ∈ E, e_ is in the alphabet of M(e) and there is no self join of e' in

M(e), e’ is called a singleton for e. if ρ'_i is a singleton for ρ'_{i-1} ($i \in [2,m]$), and ρ'_1 is also a singleton for ρ_n, we say P determines Q.”

Definition 4 “(Value Equality and Node Equality) For nodes v and v’, if name(v) = name(v’) and value(v) = value(v’), we say v and v_ are value equality, denoted as $v \equiv v'$. If v and v’ are the same node of an XML tree, we say v and v’ are node equality, denoted as $v = v'$.”

Definitions 5 through 7 are related to functional Dependencies for XML:

Definition 5 “Given D, a functional dependency(FD) σ over D is an expression of the form $(R_1,R_2,Q_1,\dots, Q_n \rightarrow P_1, \dots, P_k)$. Here R_1 is a root path expression, or $R_1 = \varepsilon$. Let $S = MCP(Q_1,\dots,Q_n, P_1, \dots, P_k)$, $Q_i = S/Q'_i$ and $P_j = S/P'_j$, S determines Q'_i and P'_j. If $S = \varepsilon$, R_2 determines Q_i and P_j ($i \in [1, n], j \in [1, k]$). XML tree X conforming to D satisfies σ, denoted as $X \models (R_1,R_2,Q_1, \dots, Q_n \rightarrow P_1, \dots, P_k)$: iff $\forall v \in \lfloor R_1 \rceil$, $\forall v_1, v_2 \in \lfloor v\{R_2\} \rceil$, $\forall u_1 \in \lfloor v_1\{S\} \rceil$, $\forall u_2 \in \lfloor v_2\{S\} \rceil$, when $u_1\{Q'i\} \equiv u_2\{Q'i\}$ holds for $\forall_i \in [1, n]$, $u_1\{P'_j\} \equiv u_2\{P'_j\}$ also holds for $\forall j \in [1, k]$. In σ, R_1 is called the context path, R_2 is called the target path, $Q_1, \dots, Q_n$ are called the head paths, and $P_1, \dots, P_k$ are called the body paths. If $R1 = \varepsilon$, the FD is absolute and holds inside the whole document; otherwise the FD is relative and holds in the tree rooted by $\lfloor R_1 \rceil$.”

Definition 6 “(Logical Implication and closure) Given D and a set of functional dependencies $\sum$ over D, given any XML document X conforming to D, if X satisfies$\sum$, must also satisfy FD σ, we say $\sum$ logical implies σ, denoted as $\sum \Rightarrow \sigma$. The set composed of all functional dependencies logical implied by $\sum$ is called the closure of $\sum$, denoted as $\sum^+$. $\sum^+ = \{\sigma \mid \sum \Rightarrow \sigma\}$."

Definition 7 "(Key for XML) Given D and a set of functional dependencies $\sum$ hold over D, if $\sigma = (R_1,R_2,Q_1,\ldots,Q_n \rightarrow \text{id}) \in \sum^+$, we say $(R_1,R_2,Q_1,\ldots,Q_n)$ is a key of D."

The following conclusions within Theorem 1 are used by them for developing inference rules to derive new dependencies for given ones:

Theorem 1 "The following conclusions can be proved by definition 5:

1. S = MCP(Q_1, …,Q_n, P_1, …, P_k), If R_1 determines R_2/S, (R_1,R_2,Q_1, …,Q_n →P_1, … , P_k). Specially, when S=ε, if R_1 determines R_2, (R_1,R_2,Q_1,…,Q_n → P_1, …, P_k).

2. (R_1,R_2,Q_1, … , Q_n → Q_i), i = [1, n].

3. When (R_1,R_2,Q_1,… , Q_n → P_1,… , P_k), (R_1,R_2, Q_1, … , Qn, Q' → P1, … , P_k,Q').

4. When (R_1,R_2,Q_1, … , Q_n → P_1, … , P_L), and (R_1,R_2, P_1, … , P_L → t_1, ,

. Create a DTD graph to represent the structure of given DTD, including elements, attributes and operators. Also add the *virtual root*, and split the shared nodes if necessary.

2. Create key relations for the chosen set of keys from original DTD (described in the previous subsection). If $K = (R_1, R_2, Q_1, ..., Q_n)$, mark the node for last(R_2) in DTD graph.

3. Define $(KR_1ID, KQ_1, ..., KQ_n)$ and (KR_2ID) as keys for key relation $KR(KR_1ID, KR_2ID, KQ1, ..., KQ_n)$.

4. Identify *top* nodes that need a separate relation. The nodes are either marked in step 2, or satisfy any of the following conditions: 1) not reachable from any nodes, 2) direct child of ”*” operator node, 3) either node between tow mutually recursive nodes(if one node is child node of ”*” operator node, choose it).

5. Starting from top node *T,* inline all the element and attribute nodes that are reachable form T unless they are other top nodes.

6. Add a *XID* field as key for all the generated relations other than key relations.

7. Add a *parent ID* field for relations to record the key value of parent element if necessary, and if the parent element *X* is inlined into another element *Y,* record the key value for *Y* instead.

List 5.8: Mapping DTD to Relations [105]

So their method keeps the good properties of normalized DTD, and can fully leverage the relational technology.

A hierarchical algorithm (S-GRACE) is proposed by Wang Lian et al., in 2004 [99] for clustering a collection of XML documents based on structural information in the data to alleviate the fragmentation problem of storing them into relational tables. To do so, they developed the notion of structure graph (s-graph), supporting a computationally efficient distance metric defined between documents and sets of documents. To achieve their goal they proposed some definition as follows:

1) A new notation to measure the similarity between XML documents:

"Definition 1: Given a set of XML documents C, the structure graph (or s-graph) of C, sg(C) = (N, E), is a directed graph such that N is the set of all the elements and attributes in the documents in C and (a, b) □ E if and only if a is a parent element of element b or b is an attribute of element a in some document in C."

2. "Theorem 1: Given a set of XML documents C, if a path expression q has answer in some document in C, then q is a subgraph of sg(C). Also, sg(C) is the minimal graph that has this property."

3. "Corollary 1: Given two sets of XML documents C1 and C2, if a path

expression q has an answer in a document of C1 and a document of C2, then q is a subgraph of both sg(C1)and sg(C2)."

4. "Definition 2: For two XML documents C1 and C2, the distance between them is defined by dist(C1, C2) = 1- $\frac{|sg(C1) \cap sg(C2)|}{\max\{|sg(C1)|,|sg(C2)|\}}$,

Where $|sg(C_i)|$ is the number of edges in $sg(C_i)$; i = 1, 2 and $sg(C_1) \cap sg(C_2)$ is the set of common edges of sg(C1) and sg(C2)."

The matrix in definition 2 above gives an evidence to identify which XML documents to be separated and which documents can be clustered.

The S-GRACE algorithm is shown in 5.8:

"In S-GRACE algorithm, the input D a set of XML documents, the s-graphs of the documents are computed and stored in the array SG. The procedure pre_clustering (line 1) creates SG from D using hashing. Two s-graphs in SG are neighbours if their distance is smaller than an input threshold. Compute_distance computes the distance between all pairs of s-graphs in SG and stores them in the array DIST. Given two s-graphs x and y in SG, link(x, y) is the number of common neighbours of x and y, where an s-graph z is a neighbour of x, if dist(x, z) Y, (y is a given distance threshold). In S-GRACE, the number of neighbours of an s-graph is weighted by the number of documents it represents. For a pair of clusters C_i, C_j, link[C_i, C_j] is the number of cross links between elements in C_i and C_j, (i.e., link[C_i, C_j] = y $_{Pq,Ci,Pr>Cj}$ link(p_q, p_r). Also, a goodness measure g(C_i, C_j)between a pair of clusters C_i, C_j is defined by[108].

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(r)} = n_i^{1+2f(r)} - n_j^{1+2f(r)}}$$

```
/* input D: a set of XML document */
/* input $B$: a similarity threshold */
/* input $\alpha$: an integer */
/* input $\beta$: a control parameter for labeling outliers */
/* input k: a control parameter for the number of clusters */
/* output Q: a set of cluster: O: outliers set */
    SG = pre_clustering (D);
    DIST = compute distance (SG)
    LINK = compute_link(DIST, SG,0);
    O = remove_outlier(LINK, SG, $\beta$);
    for each s $\in$ SG do
      G[s] = build_local_heap(LINK,s);
    Q= build _global_heap(SG, q);
    While  size (Q)>$\alpha \times \kappa$      $do\{$
    u= extract_max(Q);
    v = max( q[u]);
    delete(Q,v);
    w_merge(u, v);
     for each  $\chi \in q[u] \cup q[v] do\{$
        Link [x,w] = LINK [x, u] +LINK [x, v];
        Delete(q[x], u ); delete (q[x], v);
        Insert(q[x], w, g(x, w)): insert(q[w], x, g(x, w ));
        Update(Q, x, q[w]);
 }
 Insert(Q,w, q[w]);
 Deallocate(q[u]):deallocate(q[v]);
 }
 O=  O $\cup$ remove_outlier(LINK , Q, q, $\beta$ );
 Second_cluster(Link, Q, q,  $\kappa$ );
```

List 5.9: S-GRACE Algorithm [99]

where n_i and n_j are the number of documents in C_i and C_j , respectively, and $f(\Upsilon)$ is an index on the estimation of number of neighbours for C_i and C_j. In fact, the denominator is the expected number of cross links between the two clusters. Compute link (line 3) computes the link value between all pairs of s-graphs in SG and stores them in the array LINK. Remove outlier then removes the clusters that have no neighbours. Initially, each entry in SG is a separate cluster. For each cluster i, we build a local heap q[i] and maintain the heap during the execution of the algorithm. Q[i] contains all clusters j such that link[i, j] is nonzero. The clusters in q[i] are sorted in decreasing order by the goodness measures with respect to i. In addition, the algorithm maintains a global heap Q that contains all the clusters. The clusters i in Q are sorted in the decreasing order by their best goodness measures, g(i, max(q[i])), where max(q[i]) is the element in q[i] which has the maximum goodness measure." [99]

"The 'while loop' (lines 8-21) iterates until only $\alpha \times k$ clusters remain in the global heap Q, where α is a small integer controlling the merging process. During each iteration, the algorithm merges the pair of clusters that have the highest goodness measure in Q and updates the heaps and LINK. The s-graph of a cluster obtained by merging two clusters contains the nodes and edges of the two source clusters (refers to Definition 1). Outside the loop, remove outlier removes some more outliers from the remaining clusters which are small groups loosely connected to other nonoutlier groups. Second cluster (line 23) further combines clusters until k clusters remain. It also merges a pair of clusters at a time. The purpose is to allow different control strategies to choose the pair of clusters to be merged in the last stage of S-GRACE.” [99].

They had shown that the s-graph of an XML document can be encoded by a cheap, bit string. Clustering can then be efficiently applied to the set of bit strings for the whole document collection. With the structural information encoded, clustering of XML data becomes efficient and scalable using the proposed S-GRACE algorithm.

A B^+tree technique is introduced by Shankar Pal et al., 2004, in Microsoft Corporation [106] for indexing XML instances stored in a relational database in a decomposed form. The B^+tree called primary XML index that encodes the Infoset items of XML nodes.Using of secondary XML indexes improve the performance of common classes of queries: (a) PATH (or PATH_VALUE) index for path-based queries, (b) PROPERTY index for property bag scenarios (c) VALUE index for value-based queries, and (d) work break index for content indexingwith structural information.. They had avoided the approach of decomposition of XML instances based on their schema since their goal is a uniform data representation and query processing with or without XML schemas

Greedy Search algorithm is built by Surajit Chaudhuri et al., 2005, in Microsoft Research, University of Maryland, Seoul National University, and Indiana University [107] taking in to consideration the interplay between the physical design and logical design, in order to overcome the following issue. (1) Solving logical and physical designs independently leads to suboptimal query performance. (2) Taking physical design into account in fact influences the definition of the appropriate search space for logical designs as well as how this space can be effectively searched. Greedy, List 5.9, is an extended version of old one [108], which was proposed for logical design. They extended it by invoking the physical tools instead of the query optimizer to estimate the cost of each mapping.

```
Input: An XML schema $T$, an XML query workload $W$, a space limit $S$.
Output: The mapping $M_{min}$ and physical design $F_{min}$ with the minimal cost.
(01) $C_0$ = GenCandidate($T, W$);
     $C_1$ = MergeTypeCandidate($C_0$); $C_2$ = SplitTypeCandidate($C_0$)
(02) $M_0 = M_{min}$ =GenInitialMapping($T, C_2$)
(03) $C3$ = MergeCandidate($T, W, C_0$); $C = C1 \cup C3$
(04) $W_{SQL}$ = TranslateToSQL($T, M_0, W$)
(05) ($cost_{min}$, $F_{min}$) = PhysDesignTool($W_{SQL}, M_0, S$)
(06) while $C \neq \emptyset$ do
(07)     updated = false
(08)     for each $c \in C$ do
(09)         $M$ = ApplyTransformation($M_0, c$)
(10)         $W_{SQL}$ = TranslateToSQL($T, M, W$)
(11)         ($cost$, $F$) = PhysDesignTool($W_{SQL}, M, S$)
(12)         if $cost < cost_{min}$ then
(13)             ($cost_{min}$, $F_{min}$) = ($cost$, $F$)
(14)             $M_{min} = M$; $c_{min} = c$; updated = true
(15)         endif
(16)     endfor
(17)     if updated is false then break
(18)     else $M_0 = M_{min}$; $C = C - \{c_{min}\}$ endif
(19) endwhile
(20) return $M_{min}$, $F_{min}$
```

List 5.10: Greedy Search algorithm [107]

"The algorithm first selects a set of candidate transformations at line 1. The merge type candidates are stored in C_1 and split type candidates are stored in C_2. It then generates an initial fully split mapping M_0 at line 2 by applying all split type candidates. At line 3, candidates are merged. These newly generated candidates are added to C along with previously generated merge type candidates. At line 5, the algorithm calls the physical design tool to select physical design structures on M_0 using the SQL workload W_{SQL} translated from the XML query workload W at line 4. Lines 6 to 19 repeatedly select the minimal-cost mapping M_{min} that is transformed from the current mapping M_0 with a transformation in C. In each round, the minimal cost mapping M_{min} is initialized as M_0. For each transformation $c \in C$, lines 8 to 16 enumerate a mapping M transformed from M_0 using c (line 9), and call the physical design tool to return the cost and physical configuration of M (lines 10 and 11). Lines 12 to 15 replace M_{min} with M if the cost of M is lower than the cost of M_{min}. At the end of the round, line 17 returns if no better mapping is found. Otherwise, line 18 replaces M0 with M_{min}, deletes from C the transformation c_{min} that generates M_{min} and proceeds to the next round."

They defined, the mapping M from XML schema to relational schema R, where R is a set of relations for given a XSD tree T(V, E, A), as follows [107]:

1. Each node v with annotation in A is mapped to a relation with the annotation as table name. It has two default columns, an ID column as primary key that stores a unique node ID and a PID column as a foreign key that refers to the ID column in the table mapped from its parent.

2. Each leaf node ℓ as descendants of v is mapped to a column in the table for v if there is no annotated node along the path between ℓ and v.

3. If two nodes have the same annotation, they are mapped to the same table and the data instances for these two nodes are mapped to separate rows in the table.

However, they did not take into their account the recursive part of XSD.

5.5 Pattern Matching:

Bruno et al., [101] in their PathStack and TwigStack algorithms, tried to solve the limitation of decomposing of the twig pattern into binary structural ancestor-descendant relationships, which may generate large and possibly unnecessary intermediate results because the join results of individual binary relationships may not appear in the final results. However the approach is found to be suboptimal if there are parent-child relationships in twig patterns. But, the method may still generate redundant intermediate results in the presence of P-C relationships in twig patterns [109].

Haifeng Jiang et al., [110] developed TSGeneric+ twig join processing algorithm, on indexing XML documents, which makes use of a set of stacks to cache elements and a cursor interface that provides standard methods to return elements with possible matches in order to speed up the twig pattern match. Also, they proposed three edge-picking heuristics, top-down, bottom-up and statistics-based to select the first edge to start the processing. Their solution is to extend the existing cursor interface to reflect new abilities to access elements through indices. In addition to the existing *advance()* method, they defined two new methods: (1) Cq □ fwdBeyond(Cp) forwards Cq to the first element e, such that e.start > Cp □ start. (2) Cq □ fwdToAncestorOf(Cp) forwards the cursor to the first ancestor of Cp and returns TRUE. If no such ancestor exists, it stops at the first element e, such that e.start > Cp □start, and returns FALSE. However, it still does not solve the problem of redundant intermediate results in the presence of P-C relationships [109].

Ting Chen et al., [109] proposed holistic twig Join algorithm, iTwigJoin, which works correctly on any XML streaming scheme.

Applied on Tag+Level scheme the algorithm can process Ancestor-Descendant (A-D) or Parent-Child (P-C) only twig patterns optimally, applied on *Prefix-Path Stream* (PPS) scheme the algorithm can process A-D only or P-C only or 1-Branchnode only twig patterns optimally.

Praveen Rao and Bongki Moon [111] developed a system called *PRIX* (PRufer sequences for Inde*x*ing XML) for indexing XML documents and processing twig queries. Their work is different from previous works, in that they tried to get further optimization for twig query processing without breaking a twig into root-to-leaf paths and merging the results.

Tian Yu et al. [112] proposed, TwigStackList, algorithm to process NOT-twig query. Also they developed a new concept Negation Children Extension to determine whether an element is in the results of a NOT-twig query.

Qun Chen et al., [113] proposed an indexing framework, the layer index, and evaluation algorithms for performing the structural join operation on graph-structured XML data. This approach constructed multiple nested layers of tree-structured indexes by recursively decomposing a graph into constituent trees. Their study is different from Shurug Alkhalifa et al [114] which adopted the representation, (*DocID,* *LeftPos:RightPos*) to index XML elements of a tree–structured model.

TJFast, holistic twig join algorithm is proposed by Jiaheng Lu et al., 2005, in National University of Singapore [115] based on their extending of labelling Dewey ID. Extended Dewey gives a powerful labelling scheme, since from the label of an element alone, all the elements names along the path from the root to the element can be derived.

Algorithm TJFast is no longer guaranteed to be optimal in the case where the query contains parent-child relations between branching nodes and their children.

Table 5.5 shows a comparison between some algorithms of pattern matching for XML document.

	Algorithm	***Aims***	***Used Technique***	***Parameters***	***Advantages***	***Disadvantages***	***Reference***
1.	A novel holistic twig join algorithms for matching an XML query twig pattern **1- PathStack** **2- TwigStack**	To solve the limitation of decomposing the twig pattern.	Uses a chain of linked stacks.		Solve the problem of the large size of the intermediate result.	**For PathStack**: many intermediate results may not be part of any final answer.	***[Bruno et al. 2002]***

2.	Holistic Twig Joins on Indexed XML Documents, **1- TSGeneric** **2- TSGeneric+**	Algorithms can be developed to process twig joins based on available access methods.	- Used indexes to speed up the twig pattern matching. -Used two data type structures: - Stacks to catch elements.		Achieve some linear performance for twig pattern queries.	It is still does not solve the problem of redundant intermediate results in the presence of P-C relationship. [Ting Chen et al. 2005]	***[Haifeng Jiang et al. 2003]***
3.	Holistic Twig Join	To avoid unnecessary	**iTwigJoin:** **-** applied on		1) Reduce the amount of input I/O		***[Ting Chen et***

	Algorithm, which works correctly on any XML streaming scheme. **iTwigJoin**	scanning of irrelevant portion of XML documents, and to avoid generating redundant intermediate results.	Tag+Level scheme the algorithm can process A-D or P-C only twig patterns optimally		cost; 2) Reduce the sizes of redundant intermediate results.		***al. 2005***]
4.	**TreeMatch**	To propose a fast tree pattern matching algorithm that can directly find all	The basic idea is as follows: given a tree pattern,				*[J. T. Yao and M. Zhang, 2004]*

		matching of a tree pattern in on step.	*TreeMatch* .				
5.	*TwigStackList : A holistic Join Algorithm for Twig Query with Not-predicates on XML Data:* ***TwigStackList***	*To address the problem of XML NOT-twig matching*					*[Tian Yu et al. 2006]*

5.6 Summary

In schemaless centric techniques reviewed above, they do not require an XML DTD or XML Schema and depend on the XML document's structure to guide the mapping process. In these approaches, XML document is stored as a whole, large solid object (CLOBs, BLOBs) which is a data type provided by most relational database vendors (e.g., Oracle interMedia Text, DB2 Text Extender). Another way is to map the tree or graph structure of XML documents generically into 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 text in columns of tables. The advantages of these approaches are (1) they might be said to provide textual fidelity because 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 all these methods are that, (1) they fail to take advantage of the structural information that is available in the XML Markup, (2) they don’t take into account the query workload while constructing the relational schema, (3) none of the structure of the XML document is preserved, and (4) it is difficult to deal with huge XML documents.

While Schema centric techniques make use of schema information such as DTD or XML Schema to derive the relational storage schema, they need to create a relational schema to store the XML schema in and after that shred the XML documents to capture the data from them and store those data in the created relational schema. The advantages of these techniques are they: (1) restrict XML structure for the use and placement of Markup elements and attributes according to the defined schema, (2) enforce referential constraints, primary and foreign key relationships, (3) simplify the mapping process, since they do not require users to master a new specialized mapping language. But, the techniques reviewed above are (1) all heuristic; (2) don't consider the space of several possible relational mappings to choose the optimal one; (3) in addition, except (Atay, Chebotko et al. 2007), fixed shredding of XML documents will lead for a loss of information from the original one, (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 for database schema is needed as any change in the XML schema, which makes it very expensive in this case. 6) Sometimes, a large number of relations are needed to be created depending on the XML schema, which means a lot of joins are needed to retrieve XML document information.

CHAPTER 6: A Proposed New Algorithm

Introduction

In this chapter a full description is given for the model for mapping XML document into relational database. This includes the main mathematical concepts that are used in this model. A description of the leballing method that is used to label the XML document and identifying its content. Mapping XML to relational database algorithm, reconstructing XML document from relational database algorithms using DOM parser.

6.2 Theory Involved

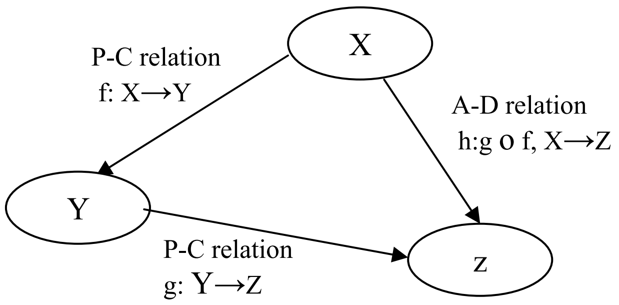
Storing XML documents into relational database means storing ordered structure hierarchical information in unordered structure tables. The aim of storing XML documents into relational database is not just as astore for backup, but it goes more than that for utilizing the strength of relational database for solving large number of data problems in retrieving information, updating data contets, concurrency control and multi-user acess. In order to maintain XML document structure and it contents relationships a alabelling technique is used to label the document elements and element attributes. A global lablling (Tatarinov, Vigla et al.2002) method is used in this research with some updates on to make the update XML document easy and with out need to relabelling the document again. The method makes use of the document structure information to guide the mapping procees. That's means no need for DTD or XML schema information in this method, which sometimes optimize the content of relational database resulted from the mapping document.

6.3 Theory Background

The hierarchical structure nature of XML document gives the ability to represent it as structure tree. A tree representation of XML document can facilitate easiy document contents relationships between nodes. Definitions 1 identify composite and associative relations XML elements as parent–child and ancestor-descendant relations. These relations help in retrieving XML document contents.

6.4 Definition 1: composite relation:

If f is a parent-child relation between X and Y as $f : X \rightarrow Y$ and g is a parent- child relation between Yand Z as $g : X \rightarrow Y$. Then we can say that h :g of is ancestor-descendent relation between X and Z as $h: X \rightarrow Z$. Figure 6.1 illustrates this composite relation.



Where P:: Parent, C::child , A :: ancestor, D:: Descendent

Figure 6.1 : Composite parent –child realtion

An XML document is a tree of nested elements, each element can have none 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 brachets, with content in between:

<element> …content …</element>

The content can contain other element, or can consist entirely of other elements, or might be empty. A ttributes 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 characteristic of XML document is a well formed. A well-formed XML document is one that conforms to some rules, such as:

▪ Having only one root element.

▪ All start tags have matching end tags.

▪ Elements must properly nested.

▪ Attribute values must always be quoted.

▪ Tags are case sensitive.

These restriction on XML document structure makes shredding process and storing of XML document in relational database easier. Definition 2 represent a complete description for XML document as a tree structure.

6.5 Definition 2:

XML tree is composed of many sub-trees of different levels; it can be defined as the following(Atay, Chebotko et al 2007)

$$T = \sum_{i=1}^{n} (E_i, A_i, X_i, r_{i-1})$$

i=1, 2 … n, represent the levels of XML tree, 0 represents the document element or tree root.

Where:

Ei is a finite set of elements in the level *i*.

Ai is a finite set of attributes in the level *i*.

Xi is a finite set of texts in the level *i*.

ri-1 is the root of the sub-tree of level *i*.

The way of processing and handling XML contents is very important in optimizing data retrieval and updating its content since this way reducing the search space of data you are dealing with instead of working with the entire document. Definition 3 and 4 give aproper definition for a way of dealing with an XML document as adynamic partition size.

6.6 Definition 3:

A dynamic fragment (shred) *df(i)* is defined to be the attributes and texts (leaf children) of the sub-tree *i* of the XML tree plus its root ri-1, as follows:

$$df(i) = (Ai, Xi, ri-1),$$

Where

Ai is a finite set of attributes in the level *i*

Xi is a finite set of texts in the level *i*.

ri-1 is the root of the sub-tree of level *i*.

6.6.1 Definition 4:

The root of the fragment (shred) is the node which has an out- degree more than one.

6.7 Mapping framework:

Mapping framework includes mapping XML documents into relational database algorithm as it is the main purpose of the research, reconstructing XML document from relational database algorithm, updating stored XML document within relational database and retrieving these data from relational database. The approach is based on the data of XML document which takes a valid XML document and shreds and composes it into relational database tables.

It dose not consider the XML schema for the following reasons:

▪ Many applications deal with highly flexible XML documents from different resources, which make it difficult to define their structure by a fixed schema or a DTD. Therefore, it is necessary for schema-less approach to deal with such XML documents variation.

▪ It is not practical to design many candidate relationanl schemas for all potential XML data, which may have different XML schema.

6.8 Labelling Method

Classical standerd labelling method is used to maintain the XML document contents. Which it uses a global label approach to give a label to the XML elements and attributes. The label is a unique for each element and attribute. But, no need to be in sequence as in (Tatarinov, Viglas et al. 2002; Soltan and Rahgozar 2006). An initial pre-order traversing for the XML document is performed. No re-labelling for XML document contents(elements and attributes) is need if new element or subtree is added to the XML document.

Figure 6.2 show an example labelling technique.

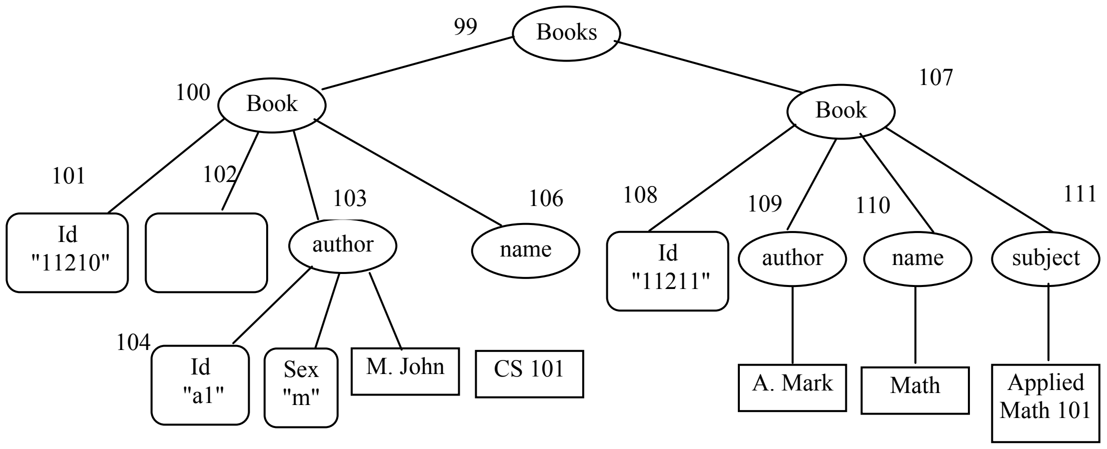


Figure 6.2: A tree representation for XML documents with labeling

6.9 Relational schema

The main issue of this method is it work with all type of XML document data centric (DTD or Schema document) and document centric (schemaless document). So that it can't build an entity model because it need XML schema or DTD to build it but, the algorithm work with schemaless document at the same time with schema document. For that a fixed relational schema consist of two tables is used to store XML documents contents and save their structures since this schema is not depend on the DTD or XML schema. The first table which is called "documents table" preserves the required information of the XML documents, the second table which is called "tochen table" preserves the detailed contents of the XML documents.

A description of a relational schema is given bellow:

1. A master table for documents is needed. It is called "documents". This table will keep information about documents themselves, at minimum it will has the following structure:

documents(doc_id, doc_structure,running time)

additional fields may be added to keep all information about the document itself such as *dates, statistics, types*… etc.

a. **The *doc_id*:** is a unique id generated per document to identify documents.

b. **The *doc_structure*:** is a big text field containing a coded string describing each document structure, any changes on the document structure should be reflected in this field, such as adding a new tag or property, deleting an existing tag or property, or relocating a given tag or property to a different location in the same document (details below).

2. A second table to store the actual contents for all documents is also estanlished. Documents will be shredded into pieces of data that will be called tokens, each document element, tag, or property will be considered a token, the tokens table will have at the minimum this structure, *tokens(doc_id, token_id, token_name, token_value)*.

a. **The *token_id*:** is the primary generated id for each token.

b. **The *doc_id*:** *is* the foreign key linking the tokens table to the documents table.

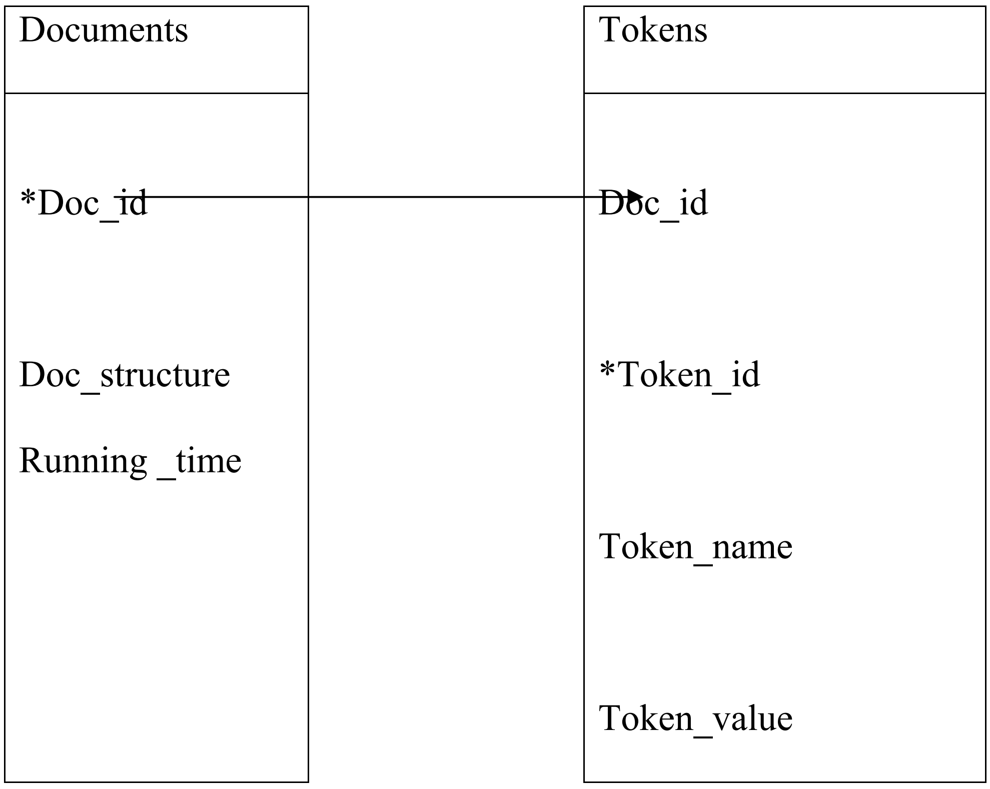
c. ***token_name*:** is the tag name or the property name as found in the original XML document.

d. ***token_value***: is the text value of the XML tag property.

Documents(***doc_id**, doc_structure)

Tokens(**doc_id, *token_id**, token_name, token_value)

Figure 6.3: Relational schema



The relation between the relational database tables

The rules for constructing doc_structure field are as follows:

Rule 1: The *doc_structure* field is where the document structure is maintained. It consists of long series of related keys.

Rule 2: Each element should use T as a start alphabet character and followed with the indexing number as a key of element e.g. T120 is a key referring to a token (element) in the tokens table whose *token_id* = 120.

Rule 3: Each child should use T as a start alphabet character and followed with the indexing number as a key of child. e.g. T12 is a key referring to a token (child) in the tokens table whose *token_id* = 12.

Rule 4: Each attribute should use A as a start alphabet character and followed with the indexing number as a key of attribute. e.g. A17 is a key referring to a token (attribute) in the tokens table whose *token_id* = 17.

This is necessary to delimit keys in the sequence..

Rule 5 : If the token has some properties defined in the original XML document then the key representing this token in the *doc_structure* will be followed with a set of keys defining these properties.

As an example, T120A12A17A2 is a valid key string which can be read as token number 120 has three properties defined by tokens number 12, 17, and 2, and these properties appear in the original document in this order.

Rule 6: If the token has some children tags (sub-tree) in the original XML document, then these children will be represented as a key-string surrounded by angle brackets. As an example, T120 <T12T7 <T2T1>

T77> is a valid string that can be read, token 120 has three sub tags in this order token 12, followed by token 7, then token 77, and token 7 itself has also two sub tags numbered 2, and number 1 in the given order.

6.10 Mapping XML to RDB algorithm:

The data model used for the mapping algorithm uses the W3C's Document Object Model (DOM) which "is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page" [2] to represent XML documents in memory before mapping them, it also uses a stack to traverse the XML document by pushing the children of each node onto stack in reverse order in order to preserve thier order in the *doc_structure* field. List 6.1 shows MapXMLtoRDB algorithm with DOM Document containing the XML document to be mapped and DocID as input, and RDB tables as output. Line 5 pushes the root element of the document to the stack. The do loop is used to construct the *doc_structure* field and to insert the XML tokens (elements and attributes) into token's table (lines 6-28).

In line 7, the top of stack is popped, if the popped element is ">", that means all the children of the parent element were added to the database, and the ">" symbol is appended to the "struc" string (lines 8-10). If not (i.e. the popped element is a node), the element's name and value are inserted into the database, and its id is appended to "struc" string. If this element has an attributes, all its attributes are inserted to the database and there ids are appended to the "struc" string. Lines (21-25) check if the element has children. If so, an "<" is appended to "struc" string, and ">" is pushed to the stack, and all its children are pushed to the stack but in reverse order. Line 26 checks the status of the stack, if it is empty, the do loop is terminated. After that, the "struc" string is inserted to the database (documents table). All element's children are enclosed by angle brackets. The nested brackets differentiate between document's levels, while using the letter 'T' and 'A' to differentiates between element's children and attribute.

```
1 XMR Algorithm
2 Input: DOM Document containing the XML document to be mapped, DocID.
3 Output: XML tables  inserted in Relational Database tables.
```

```
4  Begin
5  Initialize stack with document Element
6  Do loop
7     Pop top of stack → Element
8     If Element = ">"
9       Append to struc string
10    Else
11      Write token to database, element name, element value
12      Get token id for the added token
13      Append Id to struc string
14      If element has attributes
15        For each attribute in attributes collection do
16          Add to database as token, att. name & att. value
17          Get token id
18          Append token id to struc string
19        End for
20      End if
21      If element has child nodes
22          append "<" to struc string
23          Push ">" to stack
```

```
24          Push all childs to stack in reverse order
25      End if
26      If stack is empty exist loop
27    End if
28 End loop
29 Write struc string to database
30 End algorithm
```

List 6.1: XML Mapping to Relational database algorithm

The reconstruction algorithm for building XML document from relational database is omitted due to space issue. The algorithm mapping the document directly without any updating in the orginal data.

6.11 Reconstructing XML document from RDB Algorithm:

In this section, we propose an efficient XML reconstruction algorithm (RRX), which reconstructs the XML root element to reconstruct the original XML document from relational database.

The reconstruction process of XML document from relational database is need for the following reasons:

1. To make sure that the mapping method used in the research is in a level of maintaining the entire XMl document without loss of information and in reasonable time.

2. Document contents could be updated after mapping it into relational database and this update happened on the original XML document file. So, the old XML file is not reflecting the real state of the database table contents.

For the previous reasons, a reconstruction algorithm is used to reconstruct the entire XML document that can be exported by the user somewhere. This algorithm used the W3C's Document Object Model(DOM) to represent XML documents in memory; it also uses stack data structure to preserve document structure. List 6.2 shows "Reconst XML fromRDB" Algorithm with DocID and relational database tables as input, and XML document as output. line 5-6 get the document structure from the database and store it in sStruc as string. The do loop is used to construct the XML document from the database according to its structure in the doc_struc field. The construction process takes into consideration the document's structure from elements and attributes and their ordering. This appears of using the select state to differentiate between children(letter T)and attribute of the element (letter

A). Also, using of angle brackets (<,>) to reserved parent-child relationship. Line 31 returns the XML document.

```
1 RRX Algorithm
2 Input: DocID,RDB tables.
3 Output: XML document
4  Begin
5  Get the doc structure from the DB according to docID
6  sStruct = document!Structure
7     'parse the structure
8   IIndex=1
9   DO While IIndex<=Len(sStruct)
10  select Case Mid(sStruct,IIndex,1)
11     Case "T":
12           Reading an element id
13           Locate token
14          Create Node
15           Add oNew as a sub-node to oCurrent
16           Add oCurrent to oPage
17           IIndex= IIndex + length(element id)
```

```
18        Case "A":
19            Reading an Attribute id
20            Locate token
21            Create attribute
22            Add attribute value to oPage
23            IIndex = IIndex + length(Attribute id)
24      Case"<":  Start of sub-tree, push parent to stack, and change
parent
25          IIndex= IIndex+1
27     Case">":    End of sub-tree, pop stack and change parent
28        IIndex = IIndex+1
29           End Select
30    Loop
31   Return oPage.xml
         32  End Algorithm
```

List 6.2: RDB reconstruct to XML algorithm

The RDB reconstruct algorithm convert the data store in relational database in XML document without loss and but it in XML structure.

6.12 Differences between the presented algorithms and proposed algorithm:

In schemaless centric present algorithm reviewed above, they do not require an XML DTD or XML Schema and depend on the XML document's structure to guide the mapping process. In these approaches, XML document is stored as a whole, large solid object (CLOBs, BLOBs) which is a data type provided by most relational database vendors (e.g., Oracle interMedia Text, DB2 Text Extender). Another way is to map the tree or graph structure of XML documents generically into 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 text in columns of tables. The advantages of these approaches are (1) they might be said to provide textual fidelity because 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 all these methods are that, (1) they fail to take advantage of the structural information that is available in the XML Markup, (2) they don't take into account the query workload while constructing the relational schema, (3) none of the structure of the XML document is preserved, and (4) it is difficult to deal with huge XML documents.

This thesis proposes and develops an efficient mapping algorithm, called XMR, for storing XML documents using relational databases. XMR requires the XML data to be shredded and composed into relational tuples. The Reconstruction algorithm, RRX, reconstructs an XML subtree rooted at a node from the relational database. These algorithms solve the problem of XML type, since it works with all type of XML document, Document Type Descriptor (DTD), schema data, schema less data without need to format it. It take advantage of the structural information that is available in the XML Markup, the structure of the XML document is preserved, and it deal with huge XML documents. The algorithm performance is linear with respect to the document size which is an important issue in the processing time.

CHAPTER 7: Experimental results and Discussion

7.1 Theory implementation:

In this chapter, we will give an example to illustrate the application of the mapping method described in Subsection 6.5.3.1 Consider the XML document in Figure 7.2 as an example. Any XML document can be represented as a rooted, labeled Tree. Figure 7.3 presents an XML tree for the XML document in Figure 7.2. In our method, each node in the tree is given a generated label in pre-order traversal. This label is a unique since it identifies each token in the document.

```
<books>
  <book id="11210" category="fiction">
    <author id="a1" sex="m">M. John</author>
    <name>Computer Science 101</name>    </book>
  <book id="11211">
    <author>A. Mark</author>
    <name>Applied Math 101</name>
    <subject>Math</subject >
  </book>  </books>
```

Figure 7.1: XML document

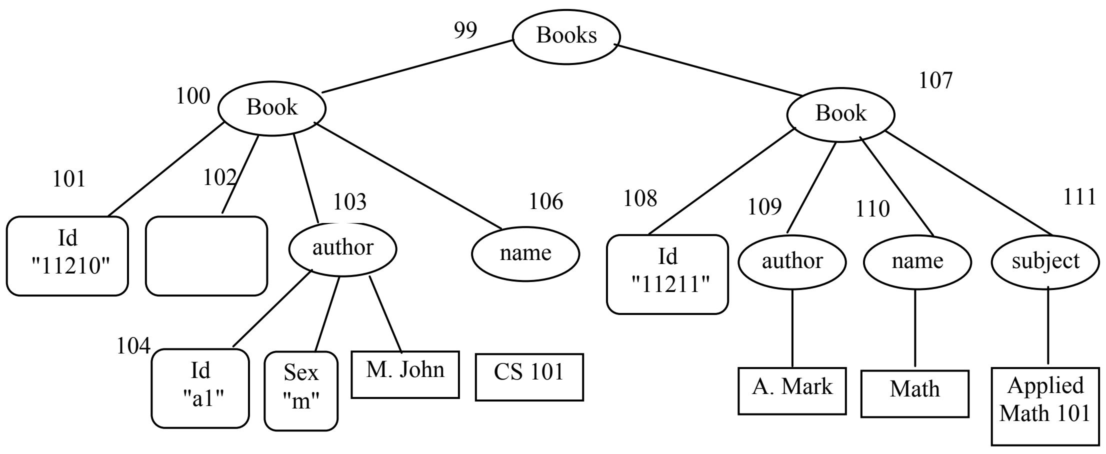


Figure 7.2: A tree representation for XML document in figure 6.1

After transformation, this document will be represented by a single record in the documents table with *doc_id* for example = 10, as in table 6.1. And the tokens table will be containing the records for the document contents as shown in table 6.2. The *doc_structure* field for this document will be.

Doc_id	**Doc_strcuture**
10	T99<T100A101A102<T103A104**A105**T106>T107 A108<T109T110T111>>

Table 7.1: Documents table

doc_id	token _id	token_name	token_value
10	99	Books	Null
10	100	Book	Null
10	101	Id	11210
10	102	Category	Fiction
10	103	Author	M. John
10	104	Id	a1
10	105	Sex	M
10	106	Name	Computer Science 101
10	107	Book	Null
10	108	Id	11211
10	109	Author	A. Mark
10	110	Name	Applied Math 101
10	111	Subject	Math

Table 7.2: Tokens table

7.2 Technologies Used:

The technologies used in the project can be classified into:

1. XML technology as a source and relational database technology as a target.
2. Visual basic 6.0 programming language is used as a tool to create the GUI and to Implement the system component. It is used for some reasons:

- The V.basic structure is very simple, particularly as the executable code.
- It is particularly easy to develop graphical user interfaces and to connect them to handler functions providede by the application>
- V.basic is a component intrgration language which is attuned to Microsoft's Component Object Model(COM).
- COM componenets and be written in different languages and then integrated using VB.
- COM components can be embedded in /linked to the application's user interface and also in/to stored documents (Object Linking and Embedding "OLE", Compound Documents").
- You can separate designing the user interface from writing the code for a form or page.

3. Microsoft Office Access is used as a relational database mananement system (RDBMS) from Microsoft which it combines the relational Microsoft Jet Database Engine with a graphical user interface and software development tool. It can easy connect with Visual basic programming language. It is used as a database development platform for the following reasons:

- It is significantly cheaper to implement and maintain compared with large database system Oracle or SQL Server.
- Company consulting rats are typically lower for Access database consultants compared with Oracle or SQL Server.
- Other software manufactures are more likely to provide interfaces to MS Access than any other desktop database system.
- When it designed correctly, access database can be ported to SQL Server or Oracle.
- An Access database can be placed on a website for access the remote users. Simple forms cand be developed within access, Data Access pages.

7.3 System Design Consideration

The system is designed in a way to achieve the method and project requirements. It consist of the two main components each of which represents one of the project requirements. These componenets are:

1. Mapping XML document from relational database.
2. Reconstructing XML document from relational database.

7.4 Experimental Envirioumentss

An Intel Core 2 Duo computer with 2 GHz CPU, 1 GB RAM, 256 MB shared Cache and running Windows Vista is used for the experimental test. Visual Basic 6 is used as software development kit with Microsoft Access 2003 as relational database target.

7.5 Experimental Data

The data is taken randomly from the XML data repository that is available at the web site of the School of Computer Science and Engineering, University of Washington [117].

7.6 Experimental procedure:

Five XML documents with different sizes are used in the experiment. The performance metric is the time spent for mapping XML documents to relational database and the time spent for reconstructing these documents from relational database. The experiment is repeated five times and the mean value of those times is reported to obtain a realistic and accurate results.

7.7 Experimental results

Document size	4 KB	28 KB	64 KB	602KB	1MB
Mapping time (secs)	0.01988238	0.14977736	.3551445	3.574335	5.85278136

Table 7.3: The time spent for mapping XML documents

The results in table 7.1 shows that the time for mapping XML document to RDB is acceptable and the relation is linear between the document size and the mapping time.

Fi

gu

re

7.3

:G

raf

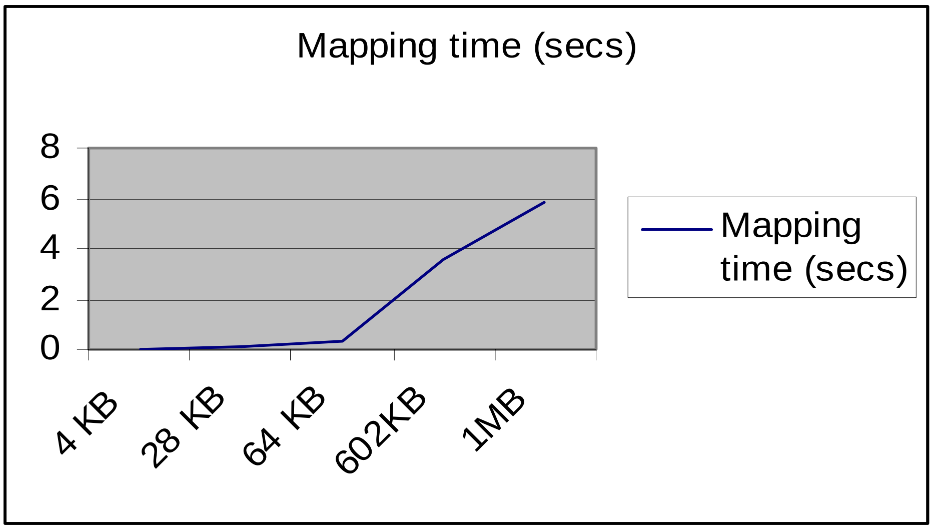


Figure 7.3: Graph represent mapping time

7.8 Reconstructing time:

Table 7.4: The time spent reconstructing them

Document size	**4 KB**	**28 KB**	**64 KB**	**602KB**	**1MB**
Reconstructing time (secs)	0.018990234	0.44980958	1.926836	18.305544	32.06255104

The results in table 7.3 shows that the time for reconstructing XML from RDB is acceptable and the relation is linear between the document size and reconstructing time.

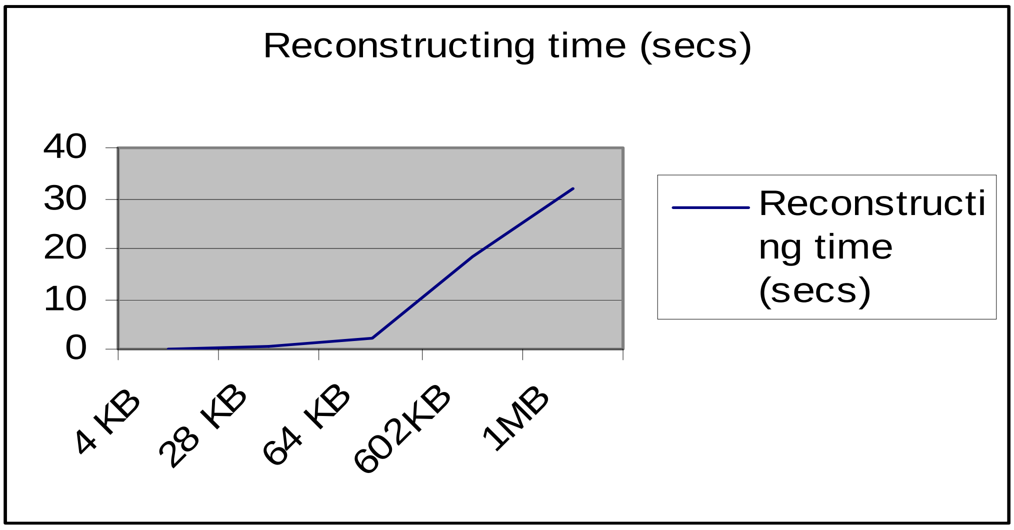


Figure 7.4 Graph represent reconstructing time

Table 7.5: Compare between Mapping time and Reconstructing time:

Document size	**4 KB**	**28 KB**	**64 KB**	**602KB**	**1MB**
Mapping time (secs)	0.01988238	0.14977736	0.3551445	3.574335	5.85278136
Reconstructing time (secs)	0.018990234	0.44980958	1.926836	18.305544	32.06255104

The results in table 7.3 shows that the time for mapping XML document to RDB spent a small time than the reconstructing operation in the same size of data.

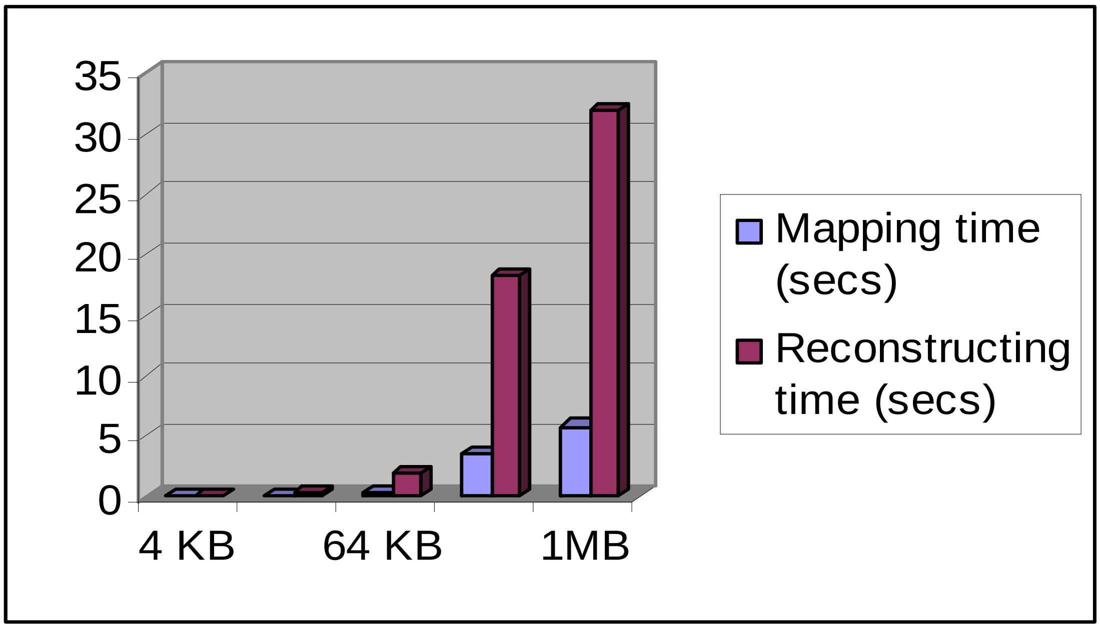


Figure 7.5: Graph represent Compare between mapping time and reconstructing time

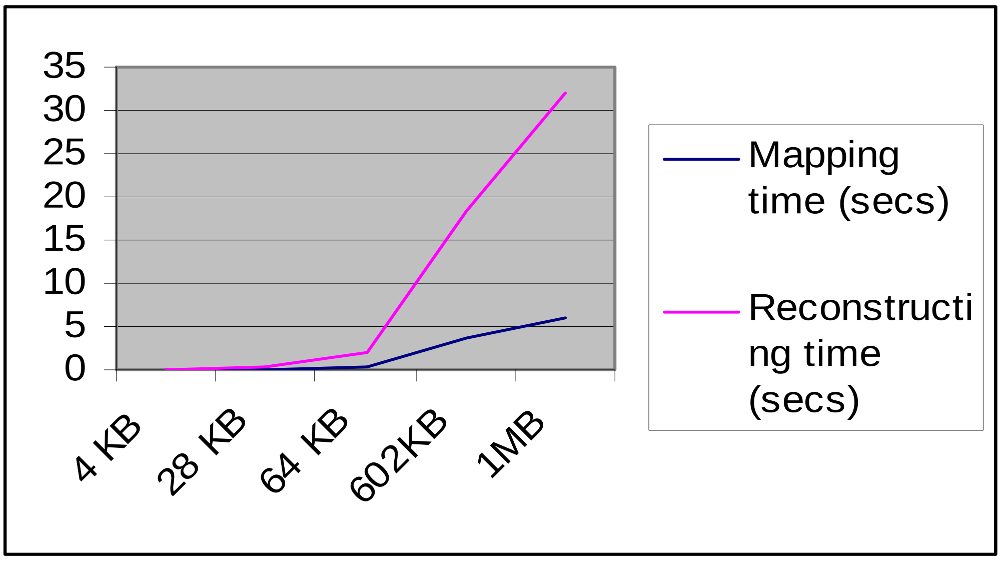


Figure7.6: Graph mapping time reconstructing time

7.9 Mapping Experiments Compare with other algorithms:

In order to study the performance of both the DOM-based data mapping algorithm (our XMR Algorithm and the OXInsert algorithm) We used An Intel Core 2 Duo computer with 2 GHz CPU, 1 GB RAM, 256 MB shared Cache and running Windows Vista is used for the experimental test. Visual Basic 6 is used as software development kit with Microsoft Access 2003 as relational database target. An experiments data is the same data in section 7.5 (4KB,28KB, 64KB,602KB, and 1024KB).

The performance shown in table 7.4 The table shows that OXInsert is efficient with the small document size and the time is well when the XML document the main memory (<= 1 MB). OXInsert performs the best on schema DTD. OXInsert performs the worst on schemaless document. The performance shown in table 7.4

Table 7.4: The OXInsert algorithm mapping time

Doc size	4 KB	28 KB	64 KB	602KB	1MB
Mapping time (secs)	0.02065214	0.23257331	0.3951687	3.886561	6.73245127

The table 7.4 shows that OXInsert is efficient with the small document size and the time is well when the XML document the main memory (<= 1 MB). OXInsert performs the best on schema DTD. OXInsert performs the worst on schemaless document.

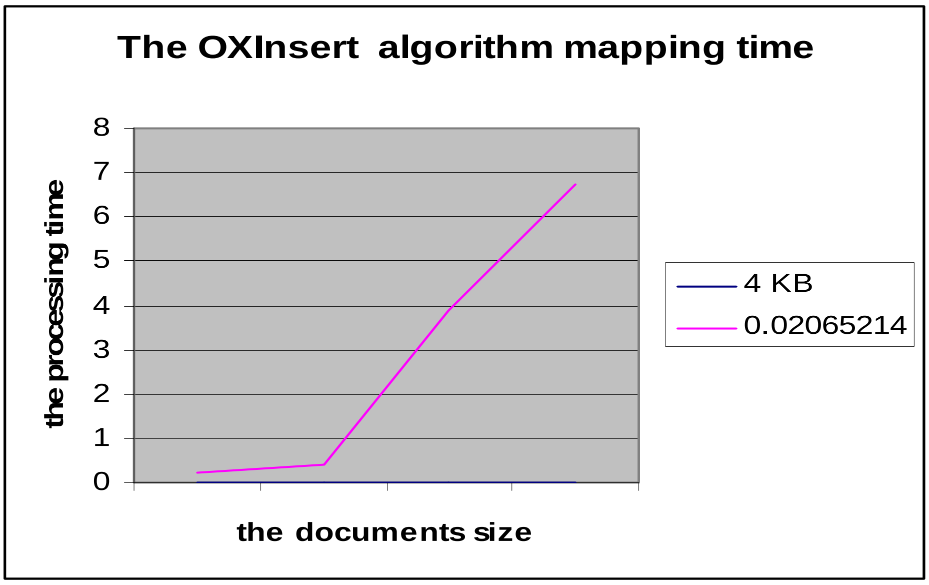


Figure7.7: Graph mapping time OXinsert

The figure 7.7 shows that OXInsert is efficient and scales well when the XML trees of the documents in the main memory (<= 1MB).

Compare the XMR algorithm and OXinsert algorithm:

Table 7.8: The OXInsert algorithm mapping time

	4 KB	28 KB	64 KB	602KB	1MB
XMR algorithm	0.019882 38	0.1497773 6	.35514 45	3.57433 5	5.8527 8136
OXisert algorithm	0.020652 14	0.2325733 1	0.3951 687	3.88656 1	6.7324 5127

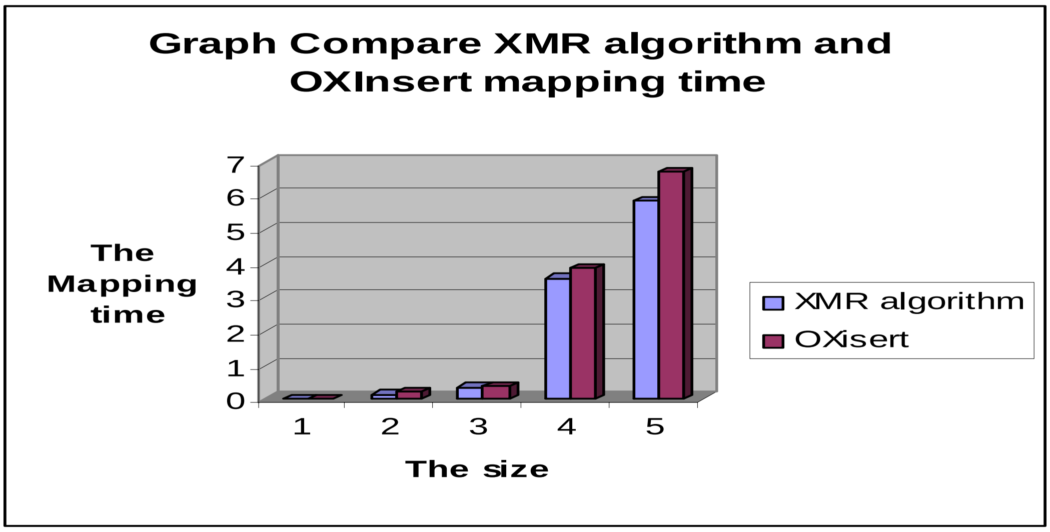


Figure7.8: Graph For Compare XMR mapping time and OXinsert mapping time

Figure 7.8 shows the difference between our algoritm and OXisert algorithm. Although this time our performance advantage is best than OXinsert but not obvious compare to OXinsert.

7.10 Compare the XMR algorithm and ODTDMap algorithm[99]

Table 7.6: The ODTDMap algorithm mapping time

Document size	4 KB	28 KB	64 KB	602KB	1MB
Mapping time (secs)	0.04697201	0.32882471	0.75155212	7.06938753	12.02483461

Table 7.7: The XMR mapping time Compare with ODTDMap mapping time

	4 KB	28 KB	64 KB	602KB	1MB
XMR algorithm	0.01988238	0.14977736	.3551445	3.574335	5.85278136
ODTDmap Algorithm	0.04697201	0.32882471	0.75155212	7.06938753	12.02483461

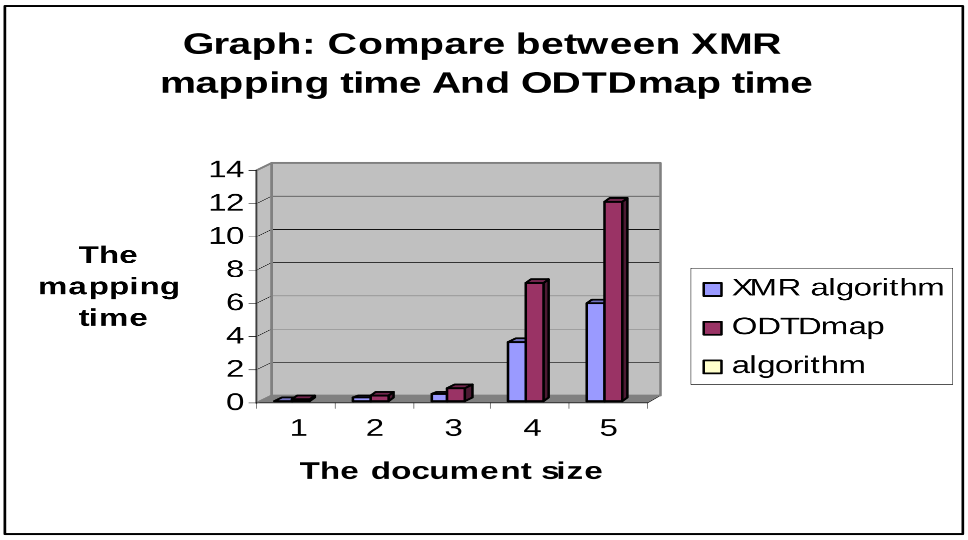


Figure7.9: Graph For Compare XMR mapping time and ODTDmap mapping time

Figure 7.9 shows the substantial performance advantage of our algorithm XMR over the OXinsert algorithm.

7.11 Reconstruction Experiments Compare with other Algorithms:

There are a few reconstruction algorithms defined in XML-publishing space[Carey et al 2000, Fernandez et al., 2002a, Shanmugasundaram et al., 2000] where existing relational data is published as an XML document. For our experimental study, we implemented algorithm XRR and two versions of algorithm Return Descendants.

The above algorithms were coded using Java 1.4.1 software development kit, Personal Microsoft Access 2003 Database was used as an XML storage. Experiments were run on the computer with CPU Pentium IV 2.4 GHz and RAM 512 MB operated by Windows XP Professional. For each experiment, we performed the reconstruction of the whole XML document for 6 times and computed the average of last 5 runs ignoring the frst run. In all experiments, we reconstructed XML data in memory and did not output it into a file or on a screen to avoid unnecessary I/O operations. which performs actual XML document reconstruction. Using the data in the mapping algorithms.

To compare scalability and performance of our algorithms with algorithm ReturnDescendants we reconstructed XML documents of size 4, 28, 64,602, and 1024. Experimental results shows in table

Table 7.8: The RRXreconctructing time Compare with returndesent time

	4 KB	**28 KB**	**64 KB**	**602KB**	**1MB**
RRX algorithm	0.018990234	0.44980958	1.926836	18.305544	32.06255104
Returndesent algortim	0.07631357	0.47885743	2.16345	21.006731	39.14256218

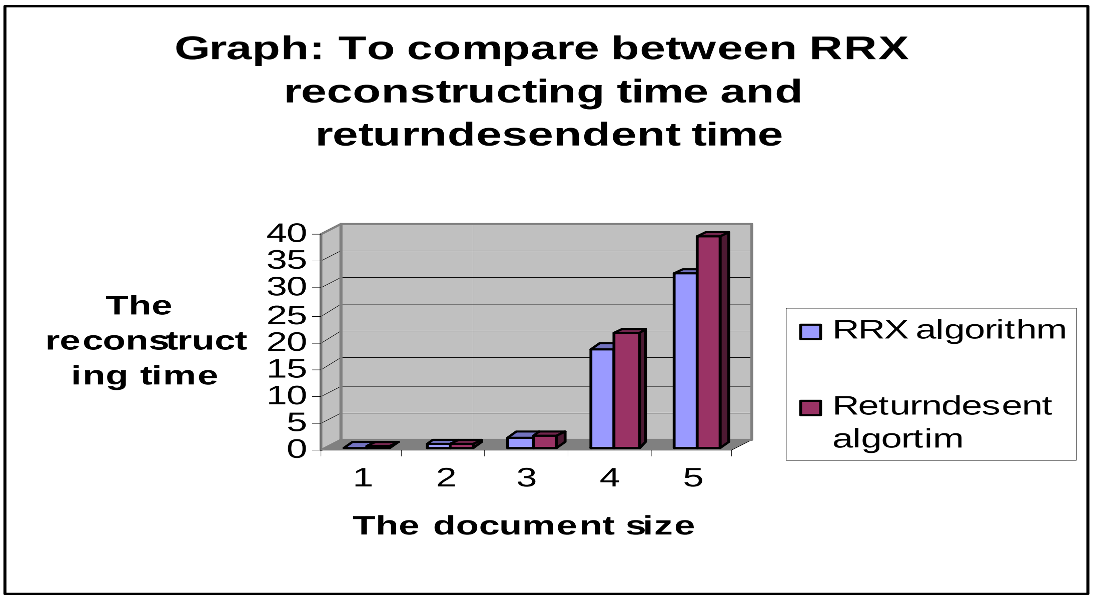


Figure7.10: Graph:To compare between RRXreconstructing time and Returndesendent time

Figure 7.10 shows the comparesion result between the two algorithm in the reconstructing time .

7.12 Confidence Interval of results:

1- calculated the sample mean of the mapping time XMR algorithm and standard deviation :

Mean= 1.99038412

Standard deviation=0.615900027

The critical value for a 95% confidence interval is 1.96, where (1-0.95)/2 = 0.025. A 95% confidence interval for the unknown mean μ is ((1.99038412 - (1.96*0.615900027)), (1.99038412 +

(1.96*0.615900027))) = (1.99038412 -1.207164053, 1.99038412 +1.207164053) = (0.783220067, 3.197548173).

2- calculated the sample mean of the mapping time ODTDMap algorithm :

Mean =4.044314196

standard deviation = 0.926013132

The critical value for a 95% confidence interval is 1.96, where (1-0.95)/2 = 0.025. A 95% confidence interval for the unknown mean μ is ((4.044314196 - (1.96* 1.326013132)), (4.044314196 + (1.96* 1.326013132))) = (4.044314196 - 1.862752875, 4.044314196 + 1.862752875) = (2.181561321, 5.907067071).

Table 7.8: Table show Confidence Interval of results: The XMR mapping time Compare with ODTDMap mapping time

Algorithm	Low	Upper
XMR	0.783220067	3.197548173
ODTDMap	2.181561321	5.907067071

Table 7.8 shows the confidence interval of the ODTDMap algorithm and XMR algorithm it's the same result that XMR has a better mapping and reconstructing time when the document size is a smale but, it is not efficient with a big document size.

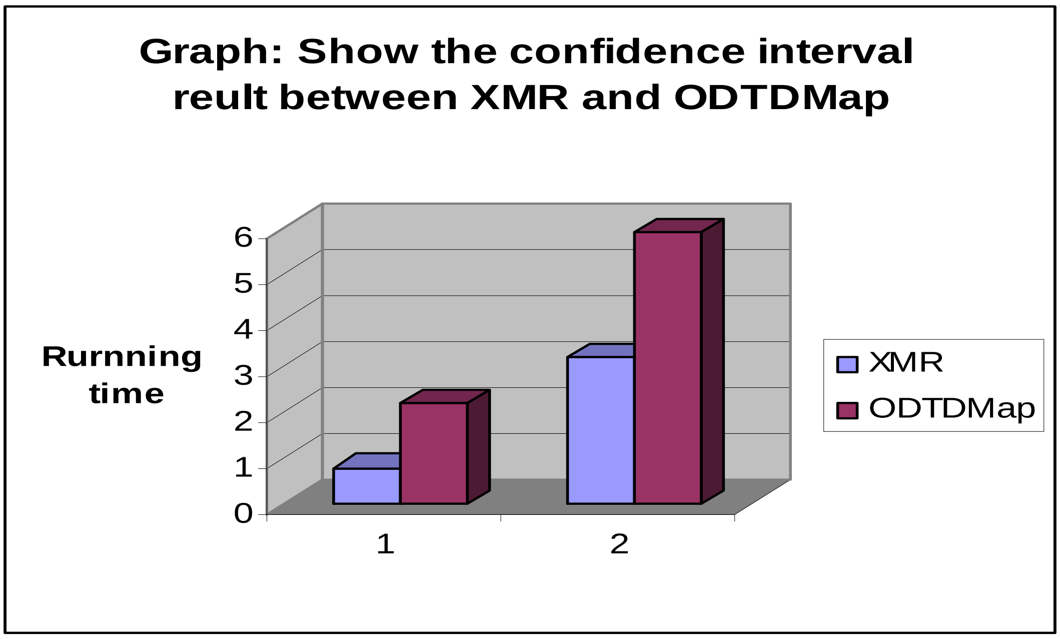


Figure7.11: Graph:To compare between XMR confidence interval and ODTDMap confidence interval

7.13 Algorithm Operations

First inialize stack with document elements. Then an outer loop will run n time n is the number of element on the stacks. The algorithm do four comparison and two inner loop first one when the element has attribute and the second one when element has child nodes. Comparison one has one assignment when if clause is true or five assignment if it false. Comparison two have five assignment if it true. Comparison three have three assignment if true. Comparison four there is one assignment if it true. If we interested in the average case, we would assume that about the half the time the if clause true. And we get one assignment each time we complete the outer loop.

The gives us the following average number of operations carried out. Using Mathmatica software to calculate the number of operation.

$$
\begin{aligned}
f(n) &= 2+\sum_{i=1}^{n} 1+(\frac{1+5}{2})+1+(\frac{0+5}{2})+1+(\frac{0+3}{2})+1 \\
&= 2+\sum_{i=1}^{n}4+\sum_{i=1}^{n}3+\sum_{i=1}^{n}\frac{5}{2}+\sum_{i=1}^{n}\frac{3}{2} \\
&= 2+4n+3n+\frac{5}{2}n+\frac{3}{2}n \\
&= 2+7n+\frac{5+3}{2}n \\
&= 2+7n+4n \\
&= 2+11n
\end{aligned}
$$

$2+11n$ n is the number of the element in the document.

Therefore this algorithm has complexity O(11n+2), which is acceptable taking into consideration it deals with all types of XML documents. And we have seen that there are better mapping algorithm but, work on one part of XML data centric or non centric data but this algorithm work with all type of XML document. In general the algorithm complexity it is good as start work .

Chapter 8: Conclusion and Future Works

8.1 Conclusion

XML is widely accepted as a standard medium for representing data exchanged between businesses on Internet since 1998. However, it was not designed for efficient storage and retrieval. As a result, seeking an efficient storage and query medium of XML documents is an attractive area of research in the database community.

For that, Mapping XML documents to RDBMS has been studied for the last few years to leverage the powerful, reliability, concurrency control, integrity, crash recovery and multi-user access of RDBMS, which are not available in XML technology until now. These studies are trying to bridge the technology gap between XML hierarchical ordered structured and RDBMS tabular unordered structure. Existing Mapping techniques from XML-to-relational can be generally classified into two tracks: the first one is the structured-centric technique, which depends on the XML document structure to guide the mapping process, and the second track is the schema–centric, which makes the use of schema information such as DTD or XML schema to derive an efficient relational storage for XML documents.

None of the above mapping XML-to-Relational technique gave an ideal solution to all the types of XML documents, which are data-centric, document-centric, and mixed documents of the previous two.

This research, identified the challenging issues for the data mapping problem which is the database vendor dependency and XML document types and information loss stored in the original XML documents due to the shredding process.

In this research, we proposed an automatic mapping technique of XML documents to RDBMS with XML-API for a database. This technique will leverage the advantages of mature relational database features and the strength of XML in data representation and exchange on the Internet. To accomplish this goal the research will propose a new Dynamic shredding mapping technique for the mapping XML-to-relational to overcome the issues of the XML documents size, loss of information stored in the original documents, and mixed XML document types. The new Technique is carried out by two linear data mapping algorithm. XML mapping to relational database (XMR) and reconstructing algorithm (RRX) to reconstruct data from relational data base to XML, base on well known parser Dom to address the problem of

XML mapping int relational data base and reconstructing XML from relational database.

We started off with a labeling and indexing technique which we use a globel indexing technique that any element or attribute take a unique lebal by this method it is very easy to maintain document structure at a low cost price and easily, building the original document is straight forward, performing first level semantic search is also achievable either on a single document or on all documents.

We introduced the mapping algorithm theoretical definition and step of mapping element and attribute in the relational database and described the algorithm suqdio code.

We trried to implement the algorithm to evaluate the algorithm performance the algorithm coded using V.Basic 6 software , Personal Microsoft Access 2003 Database was used as an XML storage. Experiments were run on the computer with CPU Pentium IV 2.4 GHz and RAM 512 MB operated by Windows XP Professional. For each experiment, we performed the reconstruction of the whole XML document for 6 times and computed the average of last 5 runs ignoring the frst run. In all experiments, we reconstructed XML data in memory and did not output it into a file or on a screen to avoid unnecessary I/O operations. which performs actual XML document reconstruction. Using the data in the mapping algorithms.

Experimental studies showed that these algorithms are efficient and well scalable with respect to size of input document, which is an important, issue in the data process and compare the expermentail with other thre algorithms (OXinsert, S-Greace,and Bacis inling) to evaluate the result which it show that our algorithm is better in processing time and no lossless of orginal document content but, the efficiency is low with large document size.

Finally it levarg the gap between the two technologies. And it deal with all types of XML documents (data-centric, document centric and mixed documents). Also, the algorthim avoid the relabeling problem in other algorithms. The algorithm overcome the limitation on the other present algorithms as we aim. This algorithm work well as the proposal guest but it need to working in the query subject to be perfect in mapping XML to relational data base.

8.2 Future works

There are several directions to extend the work described int this thesis. Work in indexing technique to perfom the lebaling method to efficient mapping data in database and make the query easy and to avoid the big text field document.

Improve this method to achieve complex semantic search, differentiate between XML data type (i.e., strings, dates, integers), in order to apply less than or greater than queries.

The application need to use SAX technology to become acceptable to all document size. Also, we will work to overcome the gap between the XML query and the SQL and how to return SQL to XML query.

8.3 References:

[1]. www.w3c.org/xml2002.

[2]. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In Proceedings of VLDB, pages 302–314, 1999.

[3]. Kei Fujimoto, Masatoshi Yoshikawa, Dao Dinh Kha and Toshiyuki Amagasa: A mapping Scheme of XML Documents into Relational Databases using Schema-based Path Identifiers. Proceedings of the 2005 International Workshop on Challenges in Web Information and Integration (WIRI'05), 2005 IEEE.

[4]. Ventzislav Tzvetkov, Xiong Wang. "DBXML - Connecting XML with Relational Databases," cit, pp. 130-135, The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

[5]. Thomas Kudrass: Management of XML Documents without Schema in Relational Database Systems. 2002 Elsevier Science B.V. Information and Software Technology 44 (2002) 269-275.

[6]. Zijing Tan, JianJun Xu, Wei Wang, Baile Shi. "Storing Normalized XML Documents in Normalized Relations," cit, pp. 123-129, The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

[7]. Hui Zhang and Frank Wm. Tompa: Querying XML Documents by Dynamic Shredding. DocEng'04, October 28–30, 2004, Milwaukee, Wisconsin, USA. Copyright 2004 ACM.

[8]. Andrey Balmin, Yannis Papakonstantinou: Storing and querying XML data using denormalized relational databases. The VLDB Journal (2005) 14: 30–49.

[9]. Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis and Vasili Zolotov: Indexing XML Data Stored in a Relational Database. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[10]. Steffen Ulsø Knudsen, Torben Bach Pedersen, Christian Thomsen, Kristian Torp. "RelaXML: Bidirectional Transfer Between Relational and XML Data," ideas, pp. 151-162, 9th International Database Engineering &#38; Application Symposium (IDEAS'05), 2005.

[11]. Alin Deutsch, Mary Fernandez and Dan Sucui: Storing Semistructured Data with STORED. SIGMOD '99 Philadelphia PA, Copyright ACM 1999.

[12]. Jagadish, H., Al-Khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos S., Patel, J., Srivastava D., Wiwatwattana N., Wu,

Y. and Yu, C.: TIMBER: A Native XML Database, SIGMOD 2003, June 9-12, 2003, San Diego, CA. Copyright 2003 ACM 1-58113-634-X/03/06.

[13]. Maxim Grinev, Andrey Fomichev, and Sergey Kuznetsov: Sedna: A Native XML DBMS Copyright MODIS ISPRAS, 2004.

[14]. Mark Logic's Content Interaction Server: http://xqzone.marklogic.com.

[15]. A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of XML documents. In Proc. of WebDB, 2000.

[16]. Guangming Xing, Jinhua Guo and Ronghua Wang: "Managing XML Documents Using RDBMS," snpd-sawn, pp. 186-191, Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05), 2005.

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

[18] . eXist "Open Source XML Database" (http://exist.sourceforge.net/)

[19]. Yi Chen, Susan Davidson, Carmem Hara, and Yifeng Zheng: RRXS: Redundancy reducing XML storage in relations. Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[20]. www.w3c.org/xml2003.

[21]. Ramez Elmasri, Shamkant B. Navathe: Fundamentals of Database Systems, Third Edition, 2000.

[22]. Kei Fujimoto, Masatoshi Yoshikawa, Dao Dinh Kha and Toshiyuki Amagasa: A mapping Scheme of XML Documents into Relational Databases using Schema-based Path Identifiers. Proceedings of the 2005 International Workshop on Challenges in Web Information and Integration (WIRI'05), 2005 IEEE.

[23]. Zijing Tan, JianJun Xu, Wei Wang, Baile Shi. "Storing Normalized XML Documents in Normalized Relations," cit, pp. 123-129, The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

[24].Hui Zhang and Frank Wm. Tompa: Querying XML Documents by Dynamic Shredding. DocEng’04, October 28–30, , Milwaukee, Wisconsin, USA. Copyright 2004 ACM.

[25].Andrey Balmin, Yannis Papakonstantinou: Storing and querying XML data using denormalized relational databases. The VLDB Journal (2005) 14: 30–49.

[26]. Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis and Vasili Zolotov: Indexing XML Data Stored in a Relational Database. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[27]. Colleen Graham: No Clear Winner in Overall RDBMS Market Share Race. 2005 Gartner, Publication Date: 23 May 2005 ID Number: G00127787.

[28]. V. Fresno-Fernández, S. Montalvo-Herranz, J. Pérez-Iglesias and J. Á. Velázquez-Iturbide: eXitor: A Tool For The Assisted Edition Of Xml Documents.

[29] Extensible Markup Language (XML). http//www.w3.org/XML/.

[30] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas, The Rufus system:

[31] S. Abiteboul, D. Quass, J. MeHugh, J.Widom, J.Wiener. The Lorel Query Language for Semi-structured Data, International Journal on Digital Libraries, 1(1), pp. 68-88, April 1997.

[32] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom, Lore: A Database Management System for Semi-structured Data, SIGMOD Record 26(3): 54-66 (1997).

[33] M. Fernandez, D. F. Florescu, J. Kang, A. Levy and D. Suciu, Catching the Boat with Strudel: Experiences with a Web-Site Management System, Proc. of ACM SIGMOD Conference on Management of Data, WA, 1998.

[34] F. Bancihon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lecluse, P. Pfeffer, P.Richard, F. Velez. The design and implementation of O2, an object-oriented database system, Proc. of the Second International Workshop on Object-oriented Database, 1988, ed. K Dittrich.

[35] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, J. F. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, VLDB 1999: 302—214.

[36]Report of a Workshop held at the Center for Intelligent Information Retrieval,

University of Massachusetts Amherst, September 2002.

[37] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval,

Zurich, Switzerland, August 18-22, 1996, pages 21–29. ACM Press, 1996.

[38] J. Pehcevski, J. Thom, and A.-M. Vercoustre. Hybrid XML Retrieval: Combining Information Retrieval and a Native XML Database. Submitted for publication.

[39] http://www.lotusnotes.com/, 1998.

[40] R.V. Zwol, P. Apers, and A. Wilschut, Modeling and Querying Semi-structured Data With MOA, Workshop on Query Processing for Semi-structured Data and Non-standard Data Formats, 1999.

[41] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu, Adding Structure to Unstructured Data, Proc. of The Internation Conference on Database Theory (ICDT), Delphi, Greece, 1997.

[42] A. Deutsch, M. F. Fernandez, D. Suciu, Storing Semistructured Data with STORED, SIGMOD Conference 1999: 431- 442.

[43]S. Netorov, S. Abiteboul, and R. Motwani, Extracting Schema for Semistructured Data, Proc. of ACM SIGMOD Conference on Management of Data, Seattle, WA, 1998.

[44] D. Florescu, D. Kossman, A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Rapport de Recherche No. 3680 INRIA, Rocquencourt, France, May 1999.

[45] F. Tian, D. DeWitt, J. Chen, and C. Zhang, The Design and Performance Evaluation of Various XML Storage Strategies. Submitted for publication ,Computer Science, University of Wisconsin, Madison.

[46] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas, Efficient Relational Storage and Retrieval of XML Documents, Proceedings of WEBDB 2000.

[47]M. F. Fernandez and J Simeon and P. Wadler, An Algebra for {XML} Query",Foundations of Software Technology and Theoretical Computer Science", pp.11-45, 2000.

[48]M. J. Carey and D. Florescu and Z. G. Ives and Y. Lu and J. Shanmugasundaram, E. J. Shekita and S. N. Subramanian",XPERANTO: Publishing Object-Relational Data as (XML)",WebDB (2000), pp. 105-110.

[49] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, XML-QL: a Query Language for XML, Proc. of the Int. WWW Conf., 1999.

[50] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura, XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases, ACM Transactions on Internet Technology, Vol. 1, No. 1, June 2001.

[31] S. M. N. Fuhr and M. Lalmas. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2003. In Proceedings of the 2nd Workshop of the INitiative for the Evaluation of XML Retrieval (INEX), Dagstuhl, Germany, December 15-17, 2003, pages 1–11, 2004.

[52] W3C Recommendation. XQuery 1.0: An XML Query Language. Available at http://www.w3.org/TR/xquery.

[53] Haixun Wang and Xiaofeng Meng. On the Sequencing of Tree Structures for XML Indexing. Proceedings of the 21st International Conference on Data Engineering (ICDE 2005) 1084-4627/05.

[54] Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

[55] Qun Chen, Andrew Lim, Kian Win Ong and Ji Qing Tang. Indexing graph-structured XML data for efficient structural join operation. Data & Knowledge Engineering 58 (2006) 159–179.

[56] Haifeng Jiang, Hongjun Lu, Wei Wang, and Beng Chin Ooi. XR-Tree: Indexing XML Data for Efficient Structural Joins. Proceedings of

the 19th International Conference on Data Engineering (ICDE'03) 1063-6382/03.

[57]. N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: Optimal XML pattern matching. In SIGMOD, pp310-321, 2002.

[58]. Ting Chen, Jiaheng Lu, and Tok Wang Ling. On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques. SIGMOD 2005 June 1416, 2005, Baltimore, Maryland, USA Copyright 2005 ACM 1595930604/ 05/06.

[59]. H. Jiang, W. Wang, H.J. Lu and J.X. Yu. Holistic Twig Joins on Indexed XML documents. Proceedings of the 29th VLDB Conference, Berlin, Germany, pp273-284, 2003.

[60] Praveen Rao and Bongki Moon. Sequencing XML Data and Query Twigs for Fast Pattern Matching. ACM Transactions on Database Systems, Vol. 31, No. 1, March 2006, Pages 299–345.

[61] Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

[62] Shurug Al-Khalifa, H.V.J agadish, Nick Koudas, Jignesh M.P atel, Divesh Srivastava, and YuqingWu. Structural Joins: A Primitive for

Efficient XML Query PatternMatching. Proceedings of the 18th International Conference on Data Engineering (ICDE.02). IEEE 2002.

[63] D. Lee and W. W. Chu. Comparative analysis of six XML schema languages. SIGMOD Record, 29(3):76{87, 2000.

[64] Document object model (DOM). Technical report, World Wide Web Consortium, http://www.w3.org/DOM/.

[65] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. SIGMOD Record, 26(3):54{66, 1997.

[66] H. Lu, G. Wang, G. Yu, Y. Bao, J. Lv, and Y. Yu. XBase: Making your gigabyte disk files queriable. In Proc. of the ACM SIGMOD Int'l Conference on Management of Data, pages 630{630, Madison, Wisconsin, June 2002.

[67] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu.

[68] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAulifie, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, M. J. SetCarey White, and DeWitt. Shoring up persistent applications. In Proc. of the ACM SIGMOD Int'l Conference on

Management of Data, pages 383{394, Minneapolis, Minnesota, May 1994.

[69] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native XML base management system. VLDB Journal, 11(4):292{314, 2002.

[70] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. In Proc. of the Third Int'l Workshop on the Web and Databases (Selected Papers), pages 53{62, Dallas, Texas, USA, May 2000.

[71] A. Berglund, S. Boag, D. Chamberlin, M. F. Ferneandez, M. Kay, J. Robie, and J. Simeon. XML path language (XPath) 2.0. Technical report, World Wide Web Consortium, 2002.

[72] Su Cheng Haw and G. S. V. Radha Krishna Rao. Query optimization techniques for XML databases. International Journal of Information Technology, 2(1):97–104, 2005. 29.

[73] Paul F. Dietz. Maintaining order in a linked list. In Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82), pages 122–127, San Francisco, CA, United States, May 5-7 1982.

[74] Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In Proceedings of the 27th International

Conference on Very Large Data Bases (VLDB'01), pages 361–370, San Francisco, CA, United States, September 11-14 -2001

[75] Ramez Elmasri and Shamkant B. Navathe. Fundamentals of Database Systems. Addison-Wesley, 3rd edition, 2000.

[76] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, and Vassilis J. Tsotras. Efficient structural joins on indexed xml documents. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), pages 263–274, Hong Kong, China, August 20-23 2002.

[77] Haifeng Jiang, Hongjun Lu, Wei Wang, and Beng Chin Ooi. XR-Tree: Indexing XMLdata for efficient structural joins. In Proceedings of the 19th Internati onal Conference on Data Engineering (ICDE'03), pages 253–263, Bangalore, India, March 5-8 2003.

[78] Hanyu Li, Mong-Li Lee, Wynne Hsu, and Chao Chen. An evaluation of xml indexes for structural join. SIGMOD Record, 33(3):28–33, September 2004.

[79] M. Zhang, J.T. Yao. The XML Algebra for Data Mining. In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01), pages 361–370, San Francisco, CA, United States, September 11-14 -2001

[80] W3C Recommendation. XQuery 1.0: An XML Query Language. Available at http://www.w3.org/TR/xquery.

[81] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. International Journal on Digital Libraries, (1):68{88, 1997.

[82] A. Deutsch, M. F. Ferneandez, and D. Florescu. A query language for XML. In Proc. of the Eighth Int'l World Wide Web Conference, Toronto, Canada, May 1999.

[83] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a graphical language for querying and restructuring XML documents. In Proc. of the Eighth Int'l World Wide Web Conference, Toronto, Canada, May 1999.

[84]http://www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Sacha.Berger/THESIS/HTML-view/annotations.html#ceri99xmlgl .

[85] XQuery 1.0: An XML Query Language, available at http://www.w3.org/ TR/xquery

[86] D. Schach, J. Lapp, J. Robie. Querying and Transforming XML. In Proc. of the Query Languages workshop, Cambridge, Mass., Dec.1998, h ttp://www.w3.org/TandS /QL /QL98/pp/query- transform.html.

[87] Arnaud Sahuguet, Bogdan Alexe, Sub-Document Queries Over XML With XSQuirrel. International World Wide Web Coference Committee.(IW3C2).

[88] W3C XSL Working Group. The Query Language Position Paper of the XSL Working Group. In Proc. of the Query Languages workshop, Cambridge, Mass.,Dec.1998, http: / /www. w3.org/TandS / QL / QL98/pp/xsl-wg-position.html.

[89] D. Schach, J. Lapp, J. Robie. Querying and Transforming XML. In Proc. of the Query Languages workshop, Cambridge, Mass., Dec.1998 ,htp://www.w3.org/ Tand S/QL/QL98/pp/query- transform.html.

[90] J. Robie. XQL FAQ. http://metalab.unc.edu/xql/.

[91] Arnaud Sahuguet. Bogdan Alexe. Sub-Document Queries Over XML with XSQuirrel. International world wide web conference Committee(IW3C2). WWW 2005 May 10-14,2005 , china ,japan.

[92] XQuery 1.0: An XML Query Language, available at http://www.w3.org/ TR/xquery

[93]A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. SIGMOD Record, 29(1):68{79, 2000.

[94] S.W. Ambler, "Mapping Objects to Relational Data", Ambysoft White Paper, 2003,

http://www.ambysoft.com/mappingObjects.html, [last access 2003-08-07-].

[95] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone, “Database Systems – Concepts, Languages and Architectures”, Mc Graw Hill, 1999.

[96] I. Tatarinov, S.D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, “Storing and Querying Ordered XML Using a Relational Database System”, SIGMOD Conference, June 2002.

[97] B. Surjanto, N. Ritter, and H. Loeser, “XML Content Management based on Object-Relational Database Technology”, in Proc. Of the 1st Int. Conf. On Web Information Systems Engineering (WISE), Hongkong, June 2000.

[98]. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In Proceedings of VLDB, pages 302–314, 1999.

[99]. Wang Lian, David Wai-lok Cheung,Nikos Mamoulis, and Siu-Ming Yiu: An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 16, NO. 1, JANUARY 2004.

[100]. Hui Zhang and Frank Wm. Tompa: Querying XML Documents by Dynamic Shredding. DocEng'04, October 28–30, 2004, Milwaukee, Wisconsin, USA. Copyright 2004 ACM.

[101]. N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: Optimal XML pattern matching. In SIGMOD, pp310-321, 2002.

[102]. Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. The Knowledge Engineering Review, Vol. 18:3, 265–280. © 2004, Cambridge University Press DOI: 10.1017/S0269888904000074.

[103]. Kei Fujimoto, Masatoshi Yoshikawa, Dao Dinh Kha and Toshiyuki Amagasa: A mapping Scheme of XML Documents into Relational Databases using Schema-based Path Identifiers. Proceedings of the 2005 International Workshop on Challenges in Web Information and Integration (WIRI'05), 2005 IEEE.

[104]. Guangming Xing, Jinhua Guo and Ronghua Wang: "Managing XML Documents Using RDBMS," snpd-sawn, pp. 186-191, Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05), 2005.

[105]. Zijing Tan, JianJun Xu, Wei Wang, Baile Shi. "Storing

Normalized XML Documents in Normalized Relations," cit, pp. 123-129, The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

[106]. Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis and Vasili Zolotov: Indexing XML Data Stored in a Relational Database. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[107]. Surajit Chaudhuri, Zhiyuan Chen, Kyuseok Shim, and Yuqing Wu. Storing XML (with XSD) in SQL Databases: Interplay of Logical and Physical Designs. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 12, DECEMBER 2005.

[108]. P. Bohannon, J. Freire, P. Roy, and J. Simeon, “From XML Schema to Relations: A Cost-Based Approach to XML Storage,” Proc. Int’l Conf. Data Eng., 2002.

[109]. Ting Chen, Jiaheng Lu, and Tok Wang Ling. On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques. SIGMOD 2005 June 1416, 2005, Baltimore, Maryland, USA Copyright 2005 ACM 1595930604/ 05/06.

[110]. H. Jiang, W. Wang, H.J. Lu and J.X. Yu. Holistic Twig Joins on Indexed XML documents. Proceedings of the 29th VLDB Conference,

Berlin, Germany, pp273-284, 2003.

[111]. Praveen Rao and Bongki Moon. Sequencing XML Data and Query Twigs for Fast Pattern Matching. ACM Transactions on Database Systems, Vol. 31, No. 1, March 2006, Pages 299–345.

[112]. Tian Yu, Tok Wang Ling, and Jiaheng Lu. TwigStackList: A Holistic Twig Join Algorithm for Twig Query with Not-predicates on XML Data. DASFAA 2006: 249-263.

[113]. Qun Chen, Andrew Lim, Kian Win Ong and Ji Qing Tang. Indexing graph-structured XML data for efficient structural join operation. Data & Knowledge Engineering 58 (2006) 159–179.

[114].Shurug Al-Khalifa, H.V.J agadish, Nick Koudas, Jignesh M.P atel, Divesh Srivastava, and YuqingWu. Structural Joins: A Primitive for Efficient XML Query PatternMatching. Proceedings of the 18th International Conference on Data Engineering (ICDE.02). IEEE 2002.

[115]. Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

[116] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, (2002), “Storing and Querying Ordered XML using a Relational Database System”, in Proc. of SIGMOD, pp 204-215.

[117] U. Washington, Computer Science & Engineering Research, (2002), “XMLData Repository”. Retrieved Jun 15, 2007 from http:// www.cs. washington.edu /research /xmldatasets/.

[118] Carmine Cesarano DIS, via Claudio 21 80125 - Napoli, IT 2003.

[119] Urvi Shah TripleHop Technologies, Inc New York, NY 10010 urvi@triplehop.com 2002).

[120] Joan Lu, School of Computing and Engineering University of Huddersfield, UK,HD1 3DH, School of Software, Yunnan Universiry, Kunming, China.

[121] (Yuichi lizuka, NTT Cyber Space Labs. **1-1** Hikarinooka Yokosuka-shi Kanagawa 239-0847 Japan, +81 468 59 2771, iizuka @ dq.isl.ntt.co.jp).

[122]

[123] Jayavel Shanmugasundaram Kristin Tufte Gang He Chun Zhang David DeWitt Jeffrey Naughton Relational Databases for Querying XML Documents:Limitations and Opportunities. ACM Transactions on Database Systems, 12(1),2005).

Appendix:

```
' Class Name    : Alabeling Dom_Base tree Phd Applcation

' Author        : Seif El Duola F. Elhaj

' Description   : "XML Documents" Mapping Program

' Date          : Dec. 11, 2007

' Last modified :

'

Option Explicit


'-------------------------------------------'

' Module Level Variables

'-------------------------------------------'

' Data Buffer

Private m_lngDocumentId As Long

Private m_strDocumentName As String

Private m_datDateCreated As Date

Private m_strStructure As String


' Other Module level declarations

Private m_bytBufferStatus As BufferStates
```

```
'-------------------------------------------'
' Class Specific events
'-------------------------------------------'
Private Sub Class_Initialize()
   Dim sErr As String
   On Error GoTo Err_Routine
   sErr = ""

   ' Initialize data buffer
   Me.Clean
Exit_Point:
   On Error Resume Next
   If sErr <> "" Then
      Class_Terminate
      On Error GoTo 0
      Err.Raise xbAppErr, "bfrDocument::Initialize", sErr
   End If
   Exit Sub
Err_Routine:
   sErr = "Vb: [" & Err.Number & "] " & Err.Description
   GoTo Exit_Point
```

```
End Sub


Private Sub Class_Terminate()
   On Error Resume Next
End Sub


'------------------------------------------'
' Property Get/Let
'------------------------------------------'
Public Property Get DocumentId() As Long
   DocumentId = m_lngDocumentId
End Property
Public Property Let DocumentId(ByVal lId As Long)
   m_lngDocumentId = lId
End Property


Public Property Get DocumentName() As String
   DocumentName = m_strDocumentName
End Property
Public Property Let DocumentName(ByVal sName As String)
   m_strDocumentName = Left(sName, 50)
   If m_strDocumentName = "" Then
      m_strDocumentName = "Document Untitled"
   End If
End Property

Public Property Get DateCreated() As Date
   DateCreated = m_datDateCreated
End Property
Public Property Let DateCreated(ByVal tDate As Date)
   m_datDateCreated = tDate
End Property

Public Property Get Structure() As String
   Structure = m_strStructure
End Property
Public Property Let Structure(ByVal sValue As String)
   m_strStructure = sValue
End Property

Public Property Get BufferStatus() As Byte
   BufferStatus = m_bytBufferStatus
```

```
End Property

Public Property Let BufferStatus(ByVal bStatus As Byte)

   m_bytBufferStatus = bStatus

End Property



'-------------------------------------------'

' Public methods

'-------------------------------------------'

Public Sub Clean()

   m_lngDocumentId = 0

   m_strDocumentName = ""

   m_datDateCreated = 0

   m_strStructure = ""



   m_bytBufferStatus = BufferStates.xbIgnoreContent

End Sub
```

```
' Module Name   : CallBacks
' Author        : Seif Elduola F. El Haj
' Date          : jan. 29, 2008
' Last modified :
' Description   : A mechanism to pass callback functions to classes
'
Option Explicit


'------------------------------------------'
'             General Constants            '
'------------------------------------------'
Global Const MAX_PATH = 260


'------------------------------------------'
'             External declares            '
'------------------------------------------'
Public Declare Function SHGetPathFromIDList Lib "shell32" (ByVal
pidList As Long, ByVal lpBuffer As String) As Long
Private Declare Function SendMessage Lib "user32" Alias
"SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal
wParam As Long, ByVal lParam As String) As Long
```

```
'-------------------------------------------'
'          Pass Mechanisim functions        '
'-------------------------------------------'
Public Function GetTheCallBackAddress(ByVal sProcName As String)
As Long
   Select Case sProcName
      Case "BrowseCallbackProc": GetTheCallBackAddress =
GetAddressofFunction(AddressOf BrowseCallbackProc)
   End Select
End Function

Private Function GetAddressofFunction(add As Long) As Long
  GetAddressofFunction = add
End Function

'-------------------------------------------'
'          The Call back procedures         '
'-------------------------------------------'
Public Function BrowseCallbackProc(ByVal hWnd As Long, ByVal
lMsg As Long, ByVal lp As Long, ByVal pData As Long) As Long
```

```
Dim lpIDList As Long, ret As Long, sBuffer As String
    'MS suggests "On Error Resume Next" to prevent an error from
    'propagating back into the calling process.
    On Error Resume Next

    Const WM_USER = &H400
    Const BFFM_INITIALIZED = 1                ' Indicates the browse
dialog box has finished initializing.
    Const BFFM_SELCHANGED = 2                 ' Indicates the selection
has changed.

    Const BFFM_SETSTATUSTEXT = (WM_USER + 100)  ' Sets the
status text to the null-terminated string specified by the message's lParam
parameter.
    Const BFFM_SETSELECTION = (WM_USER + 102)   ' Selects the
specified folder.
    Select Case lMsg
        Case BFFM_INITIALIZED
            Call SendMessage(hWnd, BFFM_SETSELECTION, 1, App.Path
& vbNullChar)
        Case BFFM_SELCHANGED
```

```
            sBuffer = Space(MAX_PATH)

            ret = SHGetPathFromIDList(lp, sBuffer)

            If ret = 1 Then

               Call SendMessage(hWnd, BFFM_SETSTATUSTEXT, 0,

sBuffer)

            End If

    End Select

    BrowseCallbackProc = 0

End Function
```

```
' Module Name   : Constants

' Author        : Seif Elduola F. El Haj

' Date          : Jan. 30, 2008

' Last modified :

'

' Description   : Application common constants

'

Option Explicit


'-------------------------------------------'

' General Constants

'-------------------------------------------'

' Redefined system constants

Global Const xbAppErr = vbObjectError + 512 + 4000


Global Const xbShiftMask = 1

Global Const xbCtrlMask = 2

Global Const xbAltMask = 4


' Format Strings

Global Const fmtYMD = "yyyy/mm/dd"
Global Const fmtDMY = "dd/mm/yyyy"

Global Const fmtMDY = "mm/dd/yyyy"

Global Const fmtHMS = "hh:mm:ss"

Global Const fmtYMDHMS = "yyyy/mm/dd, hh:mm:ss"


Global Const fmtReal = "#0.00"

Global Const fmtStandard = "#,##0.00"

Global Const fmtFinancial = "#,##0.00;(#,##0.00)"

Global Const fmtInt = "#,##0"

Global Const fmtIntPerc = "0%"


' Important Field Lengths

Global Const SmallPageSize = 25

Global Const MidPageSize = 50

Global Const BigPageSize = 100


' Measures

Global Const msComboSpacing = 5

Global Const TreeSpacing = 5

Global Const ResizeGap = 50
```

```
' App Specific
Global Const SelectTag = "=>"
Global Const MaxRepId = 8

'-------------------------------------------'
' General Error messages (non-systematic)
'-------------------------------------------'
Global Const errSysFailure = "An Upnormal Error At Application Load,
Terminating!!"
Global Const errNoSysInfo = "Sorry!! Help sub-system is not available
right now"
Global Const errNullValue = "The current field can not be null"
Global Const errListLimit = "You should select a value from the list"
Global Const errSave = "The last save operation did not complete
successfully"
Global Const errDel = "Unable to delete the current record, it could be
related to other records"
Global Const errMisc = "The last operation has failed, see the technical
details"

'-------------------------------------------'
```

```
' Application Specific Prompts
'-------------------------------------------'
Global Const prmConfirmDel = "This record is going to be deleted right
now. Are you sure?"
Global Const prmNotSaved = "The document you are about to close has
been modified, do you like to save?"
```

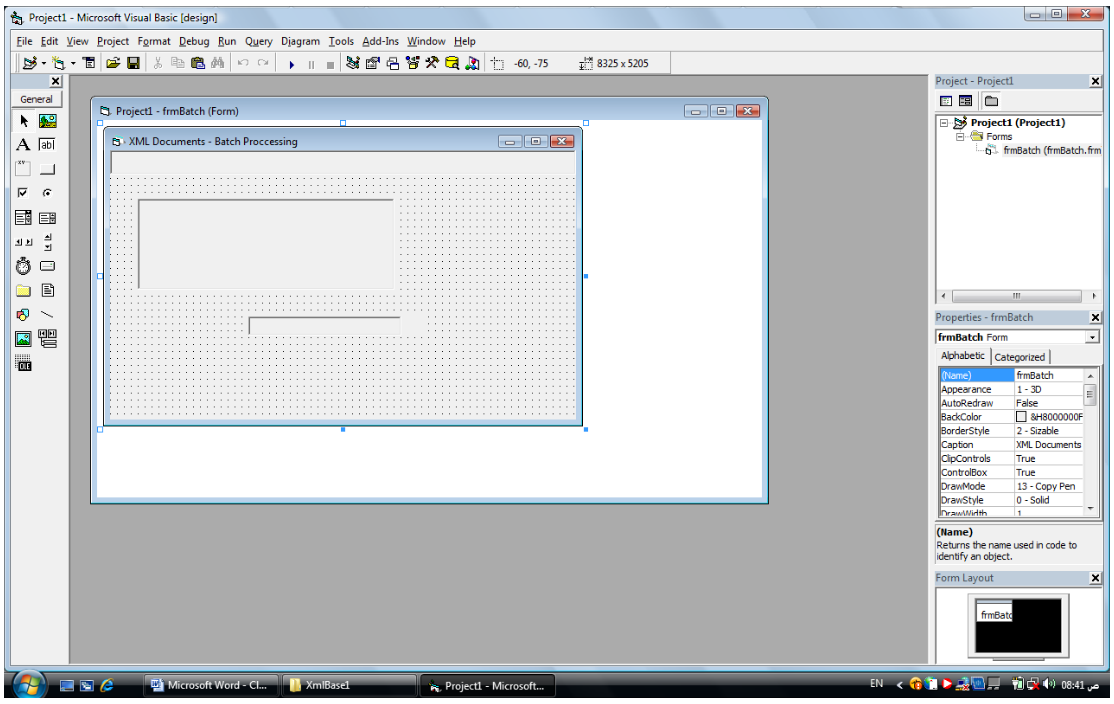
```
' Class Name    : dbxDocuments
' Author        : Seif Elduola F. El Haj
' Description   : "Documents" business object class
' Date          : Feb. 8, 2008
' Last modified :
'
Option Explicit

'-------------------------------------------'
' Module Level Variables
'-------------------------------------------'
' Module level declarations
Private m_clsErrors As sysErrorTrap
Private m_clsDataBuffer As bfrDocument

'-------------------------------------------'
' Class Specific events
'-------------------------------------------'
Private Sub Class_Initialize()
    Dim sErr As String
    On Error GoTo Err_Routine
    sErr = ""

    ' Initialize
    Set m_clsErrors = New sysErrorTrap
    Set m_clsDataBuffer = New bfrDocument
Exit_Point:
    On Error Resume Next
    If sErr <> "" Then
        Class_Terminate
        On Error GoTo 0
        Err.Raise xbAppErr, "dbxDocuments::Initialize", sErr
    End If
    Exit Sub
Err_Routine:
    sErr = "Vb: [" & Err.Number & "] " & Err.Description
    GoTo Exit_Point
End Sub

Private Sub Class_Terminate()
    On Error Resume Next
    Set m_clsErrors = Nothing
    Set m_clsDataBuffer = Nothing
End Sub


'------------------------------------------'
' Property Get/Let
'------------------------------------------'
Public Property Get TheDataBuffer() As bfrDocument
    Set TheDataBuffer = m_clsDataBuffer
End Property
Public Property Set TheDataBuffer(ByVal cBuffer As bfrDocument)
    Set m_clsDataBuffer = cBuffer
End Property


'------------------------------------------'
' Public methods
'------------------------------------------'
Public Sub GetRow(ByVal lKey As Long)
    Dim rstGet As sysDaoSqler


    ' Reset Err
    On Error GoTo Err_Routine
    m_clsErrors.ResetErr

    ' Create and open a data source
    Set rstGet = New sysDaoSqler
    rstGet.NewWhere "(DocumentId = " & lKey & ")"
    rstGet.OpenDataSource p_clsDbase, "SELECT Documents.* FROM
Documents", "", "", dbOpenSnapshot, True

    ' If not found
    If rstGet.IsEmptyRs Then
        m_clsErrors.CollectErr "Specified Document record is not found"
        GoTo Exit_Point
    End If

    ' Load record into buffer
    With m_clsDataBuffer
        .DocumentId = rstGet.TheDataSource!DocumentId
        .DocumentName = rstGet.TheDataSource!DocumentName & ""
        .DateCreated = rstGet.TheDataSource!DateCreated
        .Structure = rstGet.TheDataSource!Structure & ""
    End With
Exit_Point:
    On Error Resume Next
    Set rstGet = Nothing
    If m_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "dbxDocuments::GetRow",
m_clsErrors.TechDetails
    End If
    Exit Sub
Err_Routine:
    m_clsErrors.CollectLastVbErr
    GoTo Exit_Point
End Sub

Public Sub PutRow()
    Dim rstPut As sysDaoSqler

    ' Reset Err
    On Error GoTo Err_Routine
    m_clsErrors.ResetErr
```

```
' Creat and open a data sink
Set rstPut = New sysDaoSqler
rstPut.NewWhere "(DocumentId = " & m_clsDataBuffer.DocumentId
& ")"
rstPut.OpenDataSource p_clsDbase, "SELECT Documents.* FROM
Documents", "", "", dbOpenDynaset, False

' Start a Trans
DBEngine.Workspaces(0).BeginTrans
   On Error GoTo Err_Rollback

   ' Edit or Add a new row
   If m_clsDataBuffer.DocumentId = 0 Then
      rstPut.TheDataSource.AddNew
      rstPut.TheDataSource!DateCreated = Now
   Else
      If rstPut.IsEmptyRs Then
         m_clsErrors.CollectErr "Specified Document record is not
found"
         GoTo Err_Rollback
```

```
            End If
            rstPut.TheDataSource.Edit
        End If

        ' Write buffer to data sink
        rstPut.TheDataSource!DocumentName =
m_clsDataBuffer.DocumentName
        rstPut.TheDataSource!Structure = m_clsDataBuffer.Structure
        rstPut.TheDataSource.Update
        rstPut.TheDataSource.Bookmark =
rstPut.TheDataSource.LastModified
        m_clsDataBuffer.DocumentId = rstPut.TheDataSource!DocumentId
    DBEngine.Workspaces(0).CommitTrans
Exit_Point:
    On Error Resume Next
    Set rstPut = Nothing
    If m_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "dbxDocuments::PutRow",
m_clsErrors.TechDetails
    End If
```

```
   Exit Sub
Err_Routine:
   m_clsErrors.CollectLastVbErr
   GoTo Exit_Point
Err_Rollback:
   DBEngine.Workspaces(0).Rollback
   m_clsErrors.CollectErr "Rolled Back, due to the following error"
   GoTo Err_Routine
End Sub

Public Sub DelRow(ByVal lKey As Long)
   Dim sTemp As String

   ' Reset Err
   On Error GoTo Err_Routine
   m_clsErrors.ResetErr

   ' Start a Trans
   DBEngine.Workspaces(0).BeginTrans
      On Error GoTo Err_Rollback
```

```
        ' Delete all tokens
        sTemp = "UPDATE Tokens SET Tokens.DocumentId = Null " & _
            "WHERE (Tokens.DocumentId = " & lKey & ");"
        p_clsDbase.TheDb.Execute sTemp

        ' Delete the Document
        sTemp = "DELETE Documents.DocumentId FROM Documents "
& _
            "WHERE (Documents.DocumentId = " & lKey & ");"
        p_clsDbase.TheDb.Execute sTemp
        If p_clsDbase.TheDb.RecordsAffected <> 1 Then
          m_clsErrors.CollectErr "Specified Document is not found, or
cann't be deleted"
          GoTo Err_Rollback
        End If
    DBEngine.Workspaces(0).CommitTrans
Exit_Point:
    On Error Resume Next
    If m_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "dbxDocuments::DelRow",
m_clsErrors.TechDetails
    End If
    Exit Sub
Err_Routine:
    m_clsErrors.CollectLastVbErr
    GoTo Exit_Point
Err_Rollback:
    DBEngine.Workspaces(0).Rollback
    m_clsErrors.CollectErr "Rolled Back, due to the following error"
    GoTo Err_Routine
End Sub


'-------------------------------------------'
' Private methods
'-------------------------------------------'
```


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
-60, -75
8325 x 5205
General
Project1 - frmBatch (Form)
XML Documents - Batch Proccessing
Project - Project1
Project1 (Project1)
Forms
frmBatch (frmBatch.frm)
Form Layout
frmBatch
Microsoft Word - Cl...
XmlBase1
Project1 - Microsoft...
EN
08:41

Properties - frmBatch

frmBatch Form

Alphabetic | Categorized

(Name)	frmBatch
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	XML Documents
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1

(Name)
Returns the name used in code to identify an object.

Line 15: Class ComctlLib.Toolbar of control tbrOperations was not a loaded control class.

Line 87: Class MSFlexGridLib.MSFlexGrid of control grdTable was not a loaded control class.

Line 22: The property name _ExtentX in tbrOperations is invalid.

Line 23: The property name _ExtentY in tbrOperations is invalid.

Line 24: The property name ButtonWidth in tbrOperations is invalid.

Line 25: The property name ButtonHeight in tbrOperations is invalid.

Line 26: The property name AllowCustomize in tbrOperations is invalid.

Line 27: The property name Wrappable in tbrOperations is invalid.

Line 29: The property name _Version in tbrOperations is invalid.

Line 66: The property name Buttons in tbrOperations is invalid.

Line 93: The property name _ExtentX in grdTable is invalid.

Line 94: The property name _ExtentY in grdTable is invalid.

Line 95: The property name _Version in grdTable is invalid.

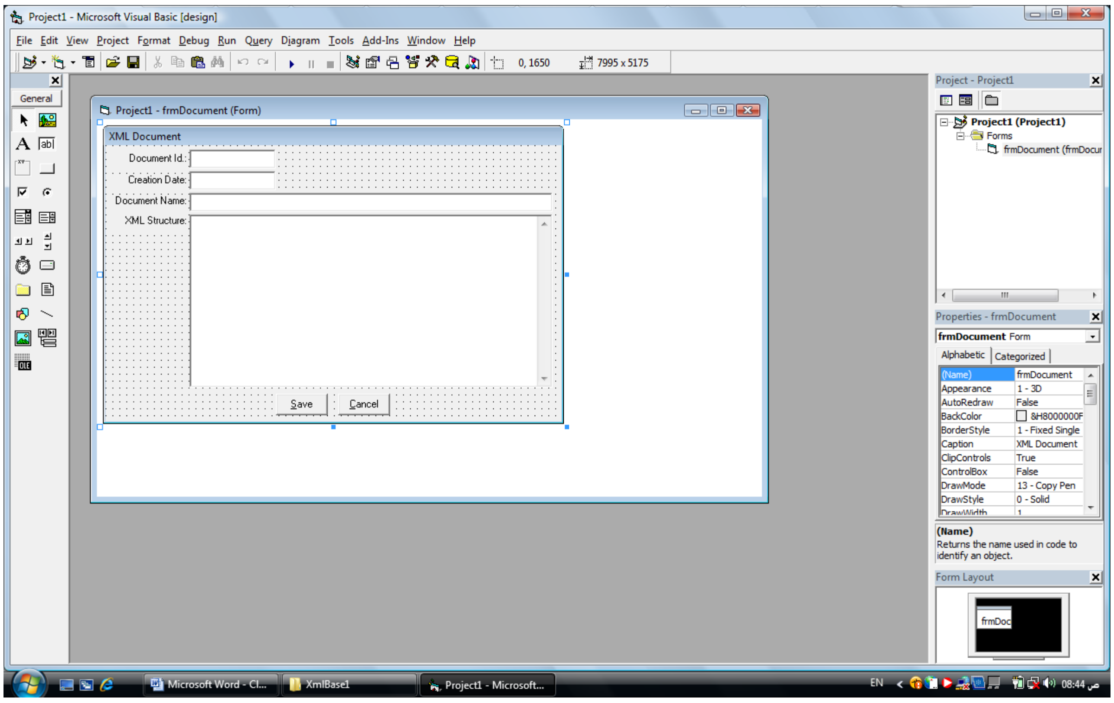
Line 96: The property name FocusRect in grdTable is invalid.

Line 97: The property name HighLight in grdTable is invalid.

Line 98: The property name ScrollBars in grdTable is invalid.

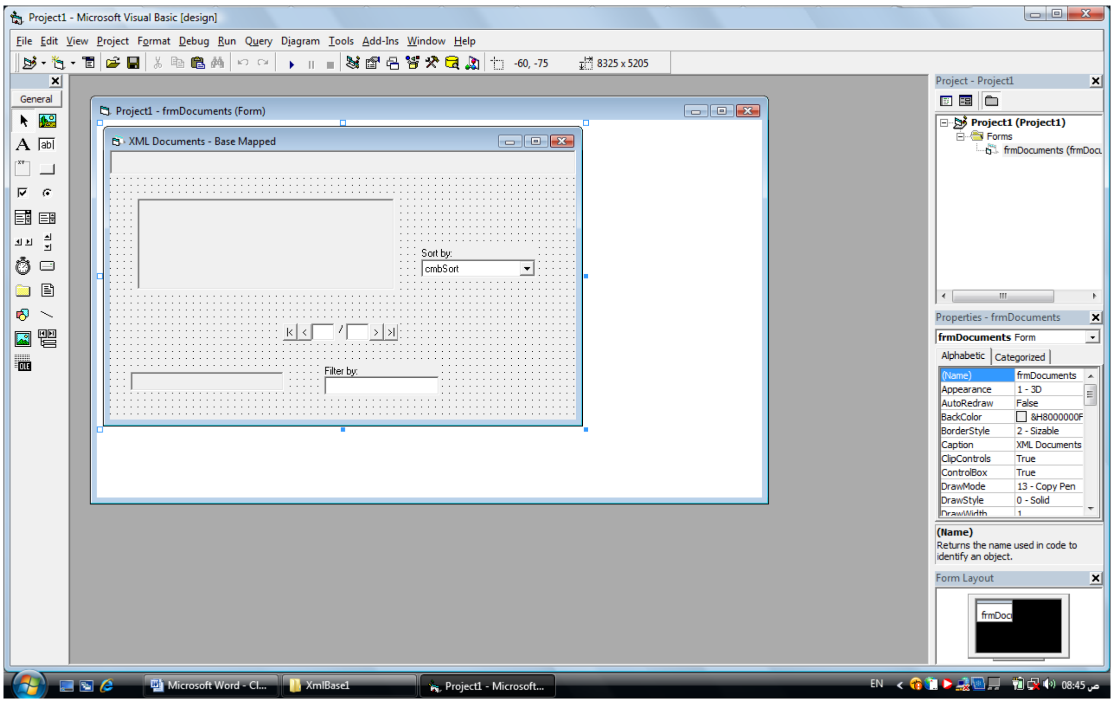
Line 99: The property name SelectionMode in grdTable is invalid.

Line 100: The property name AllowUserResizing in grdTable is invalid.



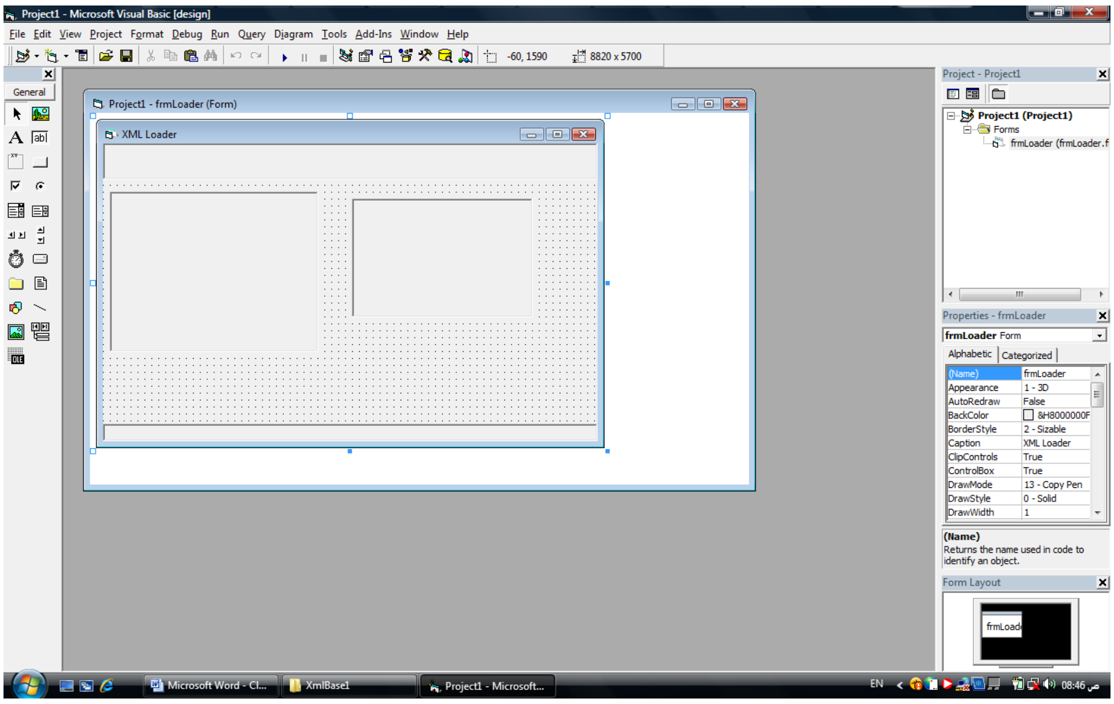
(Name)	frmDocument
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	1 - Fixed Single
Caption	XML Document
ClipControls	True
ControlBox	False
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1

(Name)
Returns the name used in code to identify an object.



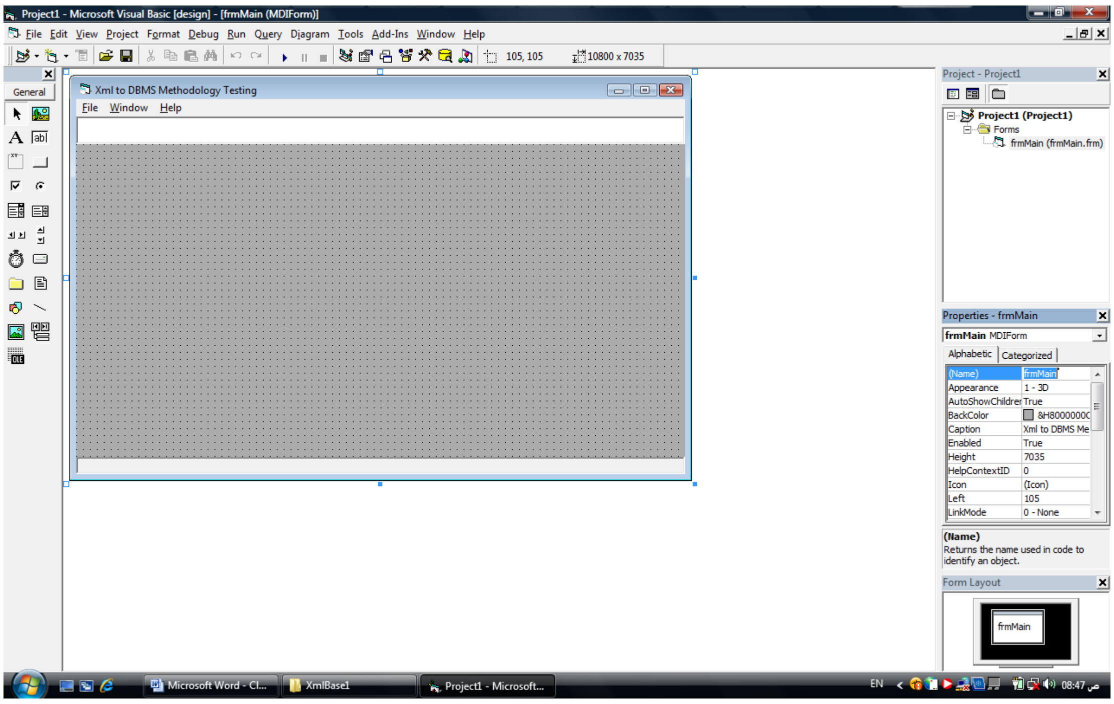
(Name)	frmDocuments
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	XML Documents
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1

(Name)
Returns the name used in code to identify an object.



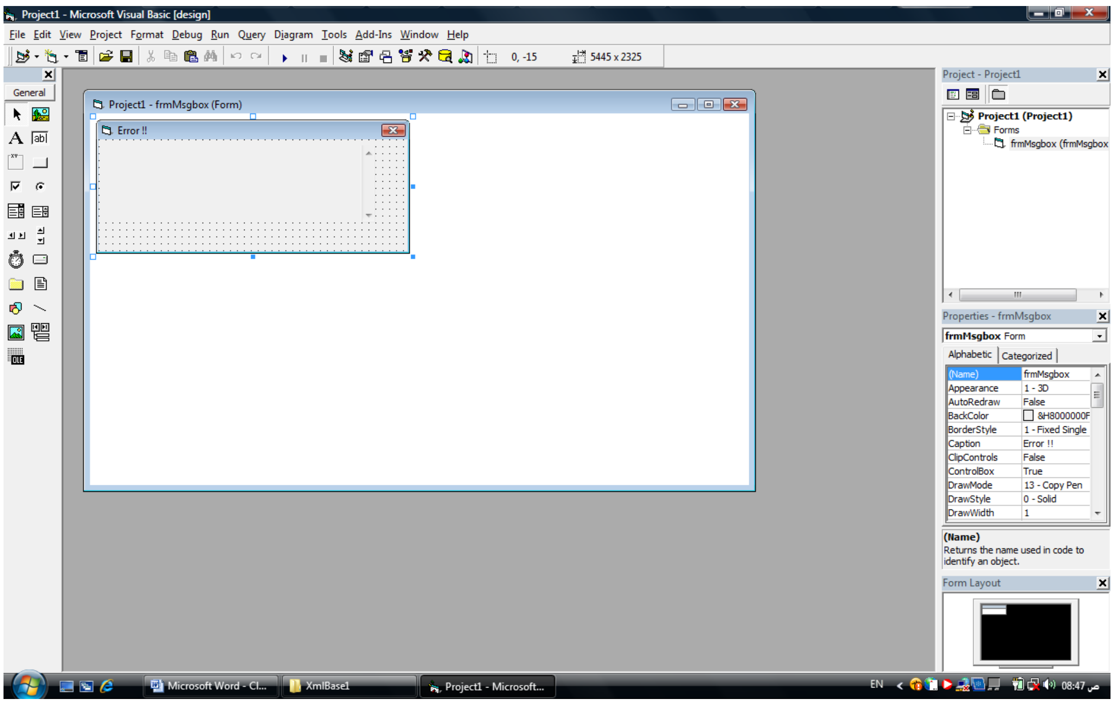
(Name)	frmLoader
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	XML Loader
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1

(Name)
Returns the name used in code to identify an object.



(Name)	frmMain
Appearance	1 - 3D
AutoShowChildren	True
BackColor	&H8000000C
Caption	Xml to DBMS Me
Enabled	True
Height	7035
HelpContextID	0
Icon	(Icon)
Left	105
LinkMode	0 - None

(Name)
Returns the name used in code to identify an object.



(Name)	frmMsgbox
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	1 - Fixed Single
Caption	Error !!
ClipControls	False
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1

(Name)
Returns the name used in code to identify an object.

```
' Module Name   : Generic
' Author        : Seif El Duola F. El Haj
' Date          : Mar. 1, 2008
' Last modified :
' Description   : My Generic tools
'
Option Explicit


'------------------------------------------'
' Data Types Tools
'------------------------------------------'
Public Function EnforceDataType(ByVal sData As String, ByVal sType
As String) As String
Dim bNumber As Byte, iNumber As Integer, lNumber As Long,
rNumber As Currency
   Select Case sType
      Case "SystemDate"
         On Error Resume Next
         If Not IsDate(sData) Then
            On Error GoTo 0
            EnforceDataType = ""
```

```
        Else
            EnforceDataType = Format(sData, fmtYMD)
            If Err.Number <> 0 Then
                On Error GoTo 0
                EnforceDataType = ""
            End If
        End If
    Case "Date"
        EnforceDataType = ParseDate(sData)
    Case "Byte"
        On Error Resume Next
        bNumber = CByte(sData)
        If Err.Number <> 0 Then
            EnforceDataType = "0"
        Else
            EnforceDataType = sData
        End If
    Case "Integer"
        On Error Resume Next
        iNumber = CInt(sData)
        If Err.Number <> 0 Then
            EnforceDataType = "0"
        Else
            EnforceDataType = sData
        End If
    Case "Long"
        On Error Resume Next
        lNumber = CLng(sData)
        If Err.Number <> 0 Then
            EnforceDataType = "0"
        Else
            EnforceDataType = sData
        End If
    Case "Currency"
        On Error Resume Next
        rNumber = CCur(sData)
        If (Err.Number <> 0) Or (rNumber < 0) Then
            EnforceDataType = "0.00"
        Else
            EnforceDataType = Format(sData, fmtReal)
        End If
    Case "Numeric"
```

```
            If IsNumeric(sData) Then
                EnforceDataType = sData
            Else
                EnforceDataType = "0"
            End If
        Case "Filter"
            If InStr(sData, "[") > 0 Then
                EnforceDataType = ""
            Else
                EnforceDataType = sData
            End If
    End Select
End Function

Private Function ParseDate(sText As String) As String
Dim sBuff(1 To 3) As String, i As Integer, iSeg As Integer
Dim dd As Integer, mm As Integer, yy As Integer, d As Date
    On Error GoTo Err_Routine

    i = 1
    iSeg = 1
```

```
sText = Trim(sText)
Do While i <= Len(sText)
   If IsNumeric(Mid(sText, i, 1)) Then
      sBuff(iSeg) = sBuff(iSeg) & Mid(sText, i, 1)
      i = i + 1
   Else
      i = i + 1
      Do While i <= Len(sText)
         If IsNumeric(Mid(sText, i, 1)) Then
            Exit Do
         Else
            i = i + 1
         End If
      Loop
      iSeg = iSeg + 1
      If iSeg > 3 Then
         Exit Do
      End If
   End If
Loop
```

```
   On Error Resume Next ' (here to prevent overflow)
   dd = CInt("0" & sBuff(1))
   mm = CInt("0" & sBuff(2))
   yy = CInt("0" & sBuff(3))
   If (dd < 1) Or (dd > 31) Or (mm < 1) Or (mm > 12) Or (yy < 1000)
Then
      GoTo Err_Routine
   End If
   d = DateSerial(yy, mm, dd)
   If Err.Number <> 0 Then
      GoTo Err_Routine
   Else
      ParseDate = Format(d, fmtDMY)
   End If
Exit_Point:
   Exit Function
Err_Routine:
   ParseDate = ""
   GoTo Exit_Point
End Function
```

```
'--------------------------------------------'
' Grid Specific Tools
'--------------------------------------------'
Public Sub FormatGrid(grdToFormat As MSFlexGrid, ParamArray
sTokens() As Variant)
Dim p As Long, i As Long, j As Long
    ' sTokens is a set of pairs, one pair per column.
    ' a pair is (Caption, width, Align) values
    p = ((UBound(sTokens) + 1) / 3) - 1
    grdToFormat.Cols = p + 1
    grdToFormat.Rows = 1
    grdToFormat.Row = 0
    For i = 0 To p
        j = i * 3
        grdToFormat.Col = i
        grdToFormat.Text = sTokens(j)
        grdToFormat.ColWidth(i) = sTokens(j + 1)
        grdToFormat.ColAlignment(i) = sTokens(j + 2)
        grdToFormat.CellFontBold = True
    Next
End Sub
```

```
Public Function GridFiller(ByRef grdToFill As MSFlexGrid, ByRef
cFrom As sysDaoSqler, ByVal bPaging As Boolean, _
            sIdColumn As String, ParamArray sColumns() As Variant)
As Variant
  Dim arrGrid() As Variant
  Dim arrTots() As Variant
  Dim iIndex As Integer
  Dim sTmp1 As String
  Dim iLoc As Integer
  Dim sTmp2 As String
  Dim lRecCount As Long

  On Error GoTo Err_Routine
  p_clsErrors.ResetErr

  grdToFill.Rows = 1
  grdToFill.Cols = UBound(sColumns) + 2
  ReDim arrTots(0 To UBound(sColumns))      ' Required here for
cases when the rsForm is empty
```

```
    If cFrom.IsEmptyRs Then
        GoTo Exit_Point
    End If

    ReDim arrGrid(0 To UBound(sColumns), 0 To 2)
    lRecCount = 0

    For iIndex = 0 To UBound(sColumns)
        sTmp1 = sColumns(iIndex)
        Do While sTmp1 <> ""
            iLoc = InStr(sTmp1, ">")
            If iLoc = 0 Then
                sTmp1 = ""
            Else
                sTmp2 = Left(sTmp1, iLoc - 1)
                sTmp1 = Mid(sTmp1, iLoc + 1)
            End If
            Select Case Left(sTmp2, 3)
                Case "<N:": arrGrid(iIndex, FillerCols.flrName) =
Mid(sTmp2, 4)
```

```
            Case "<F:": arrGrid(iIndex, FillerCols.flrFormat) =
Mid(sTmp2, 4)
            Case "<S:": arrGrid(iIndex, FillerCols.flrHaveSum) =
Mid(sTmp2, 4)
         End Select
      Loop
      arrTots(iIndex) = 0
   Next

   If Not bPaging Then
      ' Read and fill the entire table
      cFrom.TheDataSource.MoveFirst
   'Else
   '   The current required page must be set by the calling routine
   End If
   Do While Not cFrom.TheDataSource.EOF
      sTmp1 = ""
      For iIndex = 0 To UBound(sColumns)
         If arrGrid(iIndex, FillerCols.flrName) = "" Then
            sTmp1 = sTmp1 & vbTab
         Else
```

```
            If arrGrid(iIndex, FillerCols.flrFormat) = "" Then
               sTmp1 = sTmp1 & vbTab &
cFrom.TheDataSource(arrGrid(iIndex, FillerCols.flrName))
            Else
               If Left(arrGrid(iIndex, FillerCols.flrFormat), 3) = "prp"
Then
                  ' Proper Value:
                  ' All Proper format strings must be prefixed with 'prp'
followed by the enum name
                  sTmp1 = sTmp1 & vbTab &
p_clsPropers.ProperValue(Mid(arrGrid(iIndex, FillerCols.flrFormat), 4),
cFrom.TheDataSource(arrGrid(iIndex, FillerCols.flrName)))
               Else
                  ' Format Value
                  sTmp1 = sTmp1 & vbTab &
Format(cFrom.TheDataSource(arrGrid(iIndex, FillerCols.flrName)),
arrGrid(iIndex, FillerCols.flrFormat))
               End If
            End If
         End If
         If arrGrid(iIndex, FillerCols.flrHaveSum) = "Yes" Then
```

```
            arrTots(iIndex) = arrTots(iIndex) +
cFrom.TheDataSource(arrGrid(iIndex, FillerCols.flrName))
         End If
      Next

      grdToFill.AddItem sTmp1
      If sIdColumn <> "" Then
         grdToFill.RowData(grdToFill.Rows - 1) =
cFrom.TheDataSource(sIdColumn)
      End If

      cFrom.TheDataSource.MoveNext
      lRecCount = lRecCount + 1
      If bPaging Then
         If lRecCount >= cFrom.ThePageSize Then
            Exit Do
         End If
      End If
   Loop
Exit_Point:
   ' Pass back the total in all cases, even though the rsFrom is empty
```

```
    GridFiller = arrTots
    If p_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "GridFiller", p_clsErrors.TechDetails
    End If
    Exit Function
Err_Routine:
    p_clsErrors.CollectLastVbErr
    GoTo Exit_Point
End Function

Public Sub FitGrid(ByRef grdToFit As MSFlexGrid, ByVal iCol As
Integer)
    Dim i As Long
    Dim t As Long

    If iCol >= grdToFit.Cols Then
        ' Grid is not formatted yet
        Exit Sub
    End If
    t = 450
    For i = 0 To grdToFit.Cols - 1
        If i <> iCol Then
            t = t + grdToFit.ColWidth(i)
        End If
    Next
    If grdToFit.Width > t Then
        grdToFit.ColWidth(iCol) = grdToFit.Width - t
    End If
End Sub

Public Sub SelectRow(ByRef grdToSel As MSFlexGrid, Optional
ByVal bWithFocus As Boolean = True)
    grdToSel.Col = 1
    grdToSel.RowSel = grdToSel.Row
    grdToSel.ColSel = grdToSel.Cols - 1
    If Not grdToSel.RowIsVisible(grdToSel.Row) Then
        grdToSel.TopRow = grdToSel.Row
    End If
    If grdToSel.Visible And bWithFocus Then
        grdToSel.SetFocus
```

```
   End If
End Sub


'-------------------------------------------'
' Combo Specific Tools
'-------------------------------------------'
Public Function FillSortCombo(ByRef cmbToFill As ComboBox, _
                    ByVal iDefault As Integer, _
                    ParamArray sTokens() As Variant) As Variant()
   Dim rData() As Variant
   Dim iMax As Integer
   Dim i As Integer
   Dim j As Integer


   On Error GoTo Err_Routine
   p_clsErrors.ResetErr
   cmbToFill.Clear


   iMax = ((UBound(sTokens) + 1) / 2) - 1
   If iMax = -1 Then
      FillSortCombo = rData
```

```
        GoTo Exit_Point
    End If
    ReDim rData(0 To iMax, 0 To 1)


    For i = 0 To iMax
        j = i * 2
        rData(i, 0) = sTokens(j)
        rData(i, 1) = sTokens(j + 1)
        cmbToFill.AddItem sTokens(j)
    Next
    cmbToFill.ListIndex = iDefault
    FillSortCombo = rData
Exit_Point:
    If p_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "FillSortCombo", p_clsErrors.TechDetails
    End If
    Exit Function
Err_Routine:
    p_clsErrors.CollectLastVbErr
    GoTo Exit_Point
```

```
End Function

Public Sub FillXmlCombo(ByRef cmbToFill As ComboBox, ByRef
xmlFrom As MSXML2.IXMLDOMNodeList, ByVal sIdColumn As
String, ByVal lDefault As Long, ParamArray sColumns() As Variant)
Dim lSel As Long, N As MSXML2.IXMLDOMNode, i As Integer,
sData As String
    On Error GoTo Err_Routine
    p_clsErrors.ResetErr
    cmbToFill.Clear

    lSel = -1
    If xmlFrom.length = 0 Then
        GoTo Exit_Point
    End If
    For Each N In xmlFrom
        sData = ""
        For i = 0 To UBound(sColumns)
            sData = sData & Space(msComboSpacing) &
N.selectSingleNode(sColumns(i)).Text
        Next i
        cmbToFill.AddItem Trim(sData)
        cmbToFill.ItemData(cmbToFill.NewIndex) =
CLng(N.selectSingleNode(sIdColumn).Text)
        If lDefault = CLng(N.selectSingleNode(sIdColumn).Text) Then
            lSel = cmbToFill.NewIndex
        End If
    Next
Exit_Point:
    cmbToFill.ListIndex = lSel
    Set N = Nothing
    If p_clsErrors.HaveErrs Then
        On Error GoTo 0
        Err.Raise xbAppErr, "FillXmlCombo", p_clsErrors.TechDetails
    End If
    Exit Sub
Err_Routine:
    p_clsErrors.CollectLastVbErr
    GoTo Exit_Point
End Sub

Public Sub SearchCombo(cmbToSearch As ComboBox, lKey As Long)
```

```
Dim i As Long, lFound As Long
   If cmbToSearch.ListCount = 0 Then
      cmbToSearch.ListIndex = -1
      Exit Sub
   End If
   lFound = -1
   For i = 0 To cmbToSearch.ListCount - 1
      If cmbToSearch.ItemData(i) = lKey Then
         lFound = i
         Exit For
      End If
   Next
   cmbToSearch.ListIndex = lFound
End Sub

Public Sub AddCombo(cmbToAdd As ComboBox, sItem As String,
Optional iIndex As Integer = -1)
   cmbToAdd.AddItem sItem
   If iIndex <> -1 Then
      cmbToAdd.ItemData(cmbToAdd.NewIndex) = iIndex
   End If
End Sub


'------------------------------------------'
' Printer Specific Tools
'------------------------------------------'
Public Sub GetPrinters(ByVal cmbToFill As ComboBox, Optional
ByVal sDefault As String = "")
Dim objPrinter As Printer, lIndex As Long
   lIndex = -1
   For Each objPrinter In Printers
      cmbToFill.AddItem objPrinter.DeviceName
      If objPrinter.DeviceName = sDefault Then
         lIndex = cmbToFill.ListCount - 1
      End If
   Next
   cmbToFill.ListIndex = lIndex
End Sub


'------------------------------------------'
' Toolbar Specific Tools
'------------------------------------------'
```

```
Public Sub ShowButtons(ByRef tbrBar As ComctlLib.Toolbar)
Dim i As Integer
   On Error GoTo Err_Routine
   With tbrBar
      .ImageList = p_frmMain.imgOperations
      For i = 1 To .Buttons.Count
         If .Buttons(i).Style = tbrDefault Then
            .Buttons(i).Image = .Buttons(i).Key
         End If
      Next i
   End With
Exit_Point:
   Exit Sub
Err_Routine:
   MsgBox tbrBar.Buttons(i).Key & Err.Number & " " &
Err.Description
End Sub


'-------------------------------------------'
' Math. Specific Tools
'-------------------------------------------'
```

```
Public Function Ceiling(ByVal sNum As Single) As Long
   Ceiling = Int(sNum) + IIf(Int(sNum) < sNum, 1, 0)
End Function


Public Function StripTag(ByVal sSource As String, ByVal sTag As
String) As String
Dim iLoc As Integer, x As String
   iLoc = InStr(sSource, "<" & sTag & ">") + Len(sTag) + 1
   sSource = Right(sSource, Len(sSource) - iLoc)
   iLoc = InStr(sSource, "</" & sTag & ">")
   StripTag = Left(sSource, iLoc - 1)
End Function


'--------------------------------------------'
' Misc.
'--------------------------------------------'
Public Function RemAmper(ByVal sText) As String
Dim i As Integer
   i = InStr(sText, "&")
   If i = 0 Then
      RemAmper = sText
   Else
      RemAmper = Left(sText, i - 1) & Right(sText, Len(sText) - i)
   End If
End Function


'-------------------------------------------'
' User recent choices
'-------------------------------------------'
Public Sub PutFormInfo(ByRef f As Form, sWhat As String)
Dim sValue As String
   sValue = "I:" & f.Tag & ",S:" & f.WindowState

   If f.WindowState = vbNormal Then
      If InStr(sWhat, "H") <> 0 Then
         sValue = sValue & ",H:" & f.Height
      End If
      If InStr(sWhat, "W") <> 0 Then
         sValue = sValue & ",W:" & f.Width
      End If
      If InStr(sWhat, "L") <> 0 Then
         sValue = sValue & ",L:" & f.Left
```

```
        End If
        If InStr(sWhat, "T") <> 0 Then
            sValue = sValue & ",T:" & f.Top
        End If
    End If
    SaveSetting App.EXEName, "Forms", "F" & f.Tag, sValue
End Sub

Public Function GetFormInfo(ByRef f As Form) As Boolean
Dim sValue As String, iLoc As Integer, sKey As String, iState As
Integer
    GetFormInfo = False
    sValue = GetSetting(App.EXEName, "Forms", "F" & f.Tag, "")
    If sValue = "" Then
        GoTo Exit_Point
    End If
    ' We have a coded string to use (I:#,H:#,W:#,L:#,T:#)
    iState = -1
    Do While sValue <> ""
        iLoc = InStr(sValue, ",")
        If iLoc = 0 Then
```

```
      sKey = sValue
      sValue = ""
    Else
      sKey = Left(sValue, iLoc - 1)
      sValue = Right(sValue, Len(sValue) - iLoc)
    End If
    Select Case Left(sKey, 2)
      Case "I:": ' Do nothing
      Case "S:": iState = Val(Right(sKey, Len(sKey) - 2))
      Case "H:": f.Height = Val(Right(sKey, Len(sKey) - 2))
      Case "W:": f.Width = Val(Right(sKey, Len(sKey) - 2))
      Case "L:": f.Left = Val(Right(sKey, Len(sKey) - 2))
      Case "T:": f.Top = Val(Right(sKey, Len(sKey) - 2))
    End Select
  Loop
  If (iState = vbMinimized) Or (iState = vbMaximized) Then
    f.WindowState = iState
  End If
  GetFormInfo = True
Exit_Point:
End Function
```

```
' Module Name   : Globals

' Author        : Seif ElDuola F. El Haj

' Date          : Jan. 10, 2008

' Last modified :

'

' Description   : Global level definitions

'

Option Explicit


'-------------------------------------------'

' Global variables

'-------------------------------------------'

Public p_clsAccelr As sysAccelerator

Public p_clsErrors As sysErrorTrap

Public p_clsPropers As sysPrpValues

Public p_clsDbase As sysDatabase

Public p_frmMain As frmMain


'-------------------------------------------'

' Enumerations

'-------------------------------------------'
```

```
' General Enumerations

Public Enum EnumBounds

   reMaxDataTypes = 2

End Enum


Public Enum AboutModes

   xbModeNormal = 0

   xbModeSplash = 1

End Enum


Public Enum BufferStates

   xbSaveData = 0

   xbIgnoreContent = 1

End Enum


Public Enum FillerCols

   flrName = 0

   flrFormat = 1

   flrHaveSum = 2

End Enum
```

```
'------------------------------------------'
'          App Specific Enumerations         '
'------------------------------------------'
Public Enum FormIds
   id_frmMain = 0
   id_frmMsgBox = 1
   id_frmDocument = 2
   id_frmDocuments = 3
   id_frmLoader = 4
   id_frmBatch = 5
End Enum
```