EVALUATION OF XML DOCUMENTS QUERIES BASED ON NATIVE XML DATABASE

RAGHAD YASEEN LAZIM

Master of Computer Science

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : Raghad Yaseen Lazim

Date of Birth : 1984-05-15

Title : Evaluate XML Documents Queries Based on Native XML Database

Academic Session :

I declare that this thesis is classified as:

	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)
	RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserve the right as follows:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified By:

(Student's Signature) (Supervisor's Signature)

A11248288 AZHAR KAMALUDIN

New IC/Passport Number Name of Supervisor

Date: Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

SUPERVISOR'S DECLARATION

We hereby declare that I/We* have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Master in Computer Science.

(Supervisor's Signature)

Full Name : DR ADZHAR BIN KAMALUDIN

Position : SENIOR LECTURER

Date :

(Co-supervisor's Signature)

Full Name : PROF MADYA MAZLINA ABDUL MAJID

Position : PROF MADYA

Date :

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

_____

(Student's Signature)

Full Name : RAGHAD YASEEN LAZIM

ID Number : MCC11002

Date : 30 Noverber 2016

To My Parents, for their affection, love,
encouragement and prays of day and night that
make me to be able to get such a success and honour

EVALUATION OF XML DOCUMENTS QUERIES BASED ON NATIVE XML DAABASE

RAGHAD YASEEN LAZIM

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Master of Compuer Science

Faculty of Computer System & Software Engineering
UNIVERSITI MALAYSIA PAHANG

NOVEMBER 2016

ABSTRACT

As the amount of data available on the Internet grows rapidly, more and more of the data becomes semi structured. The *Extensible Markup Language* (XML), as a format for semi structured data, has become a standard for the representation and exchange of data over the Internet. Early in the XML history there were thoughts about whether XML is different from other data formats that require a database of its own. The popularity and wide-spread use of XML among a diverse set of organizations has engendered a rethinking of the storage and retrieval practices for data. Most early XML storage practices relied on mappings and transformations between XML data trees and relational database tables within a Relational Database. Though relational databases can represent nested data structures by using tables with foreign keys, it is still difficult to search these structures for objects at an unknown depth of nesting; by contrary, it is a potential advantage in XML. Also, the nested and repeating elements in XML documents can quite easily result in an unmanageable number of tables. Furthermore, it is usually very difficult after insertion to change the relational schema due to XML schema changes. The limitations of relational approaches are now well known. Moreover, local update to the document should not cause drastic changes to the whole storage system. Therefore, the design of the storage system should trade-off between the query performance and update costs. This study is to evaluate the Native XML database (NXD) performance in a comparison with XML_Enabled Database (XED), and then to enhance Entity Relationship (ER) algorithm of the relational schema for the improvement of Insert, Delete, Update and Search XML document (XML files with a large number of elements) and finally, to validate the algorithm in NXD and compare the performance of XED and NXD, by implementing the same command and control data model. Five different sizes of datasets have been used (65.8, 101, 117, 127, 183 MB). Benchmark techniques is used to measure the performance. XMark and XMark-1 are two main tools of Benchmarks in the research field, and they have used for the dataset. The performance of a system can be measured by using datasets of varying sizes, different documents with different features. The size of XML documents and the number of elements have been determined by the factor of the main driver of generation. The result of this study shown that XED has better performance for the datasets <=117 MB. The performance of XED begins to decline with the increase in the size of XML data (>127 MB), while NXD shown better performance in for the data (=>127 MB). NXD produced better results in the reporting section, which implies that the NXD X-Query has performance gains from query optimization. Most of the figures show that the XED starts better, but becomes worse as data size grows. The difference becomes obvious as the query becomes more complicated.

ACKNOWLEDGEMENTS

First and foremost, all praise and deep thanks are to ALLAH to whom be ascribed all perfection and majesty, who helped and guided me through the challenges of my study. Glory is to ALLAH who has given me strength, patience, and knowledge to continue and finish my master journey

I would like to express my sincerest gratitude to my supervisor, Dr. Adzhar Kamaludin, who has supported me throughout my study and research, for his patience, motivation, enthusiasm, and immense knowledge, whilst allowing me to work in my own way. Thanks for his guidance helped me in all the time of research and writing of this thesis, and make this research possible. One simply could not wish and imagine for a better or friendlier supervisor. Besides my supervisor, I also would like to express my special thanks to my co-supervisor Assoc. Prof. Dr. Mazlina Abdul Majid for her co-operation throughout the study. I also sincerely thanks for the time spent proofreading and correcting my many mistakes. Also, very special thanks to Dr. Qin Hongwu for his advice, support and kindness during difficult periods of my study.

A special thanks to the special person -Liu Meng (Omar)-. Liu Meng is an amazing person who has been supportive in every way. My sincere thanks go to all my lab mates and members of the staff of the Faculty of Computer Systems & Software Engineering, UMP, who helped me in many ways and made my stay at UMP pleasant and unforgettable. Many special thanks go to all my friends for their excellent co-operation, inspirations and supports during this study.

ABSTRAK

Oleh kerana jumlah data yang wujud di Internet meningkat dengan cepat, semakin banyak data menjadi separa berstruktur. The Extensible Markup Language (XML), sebagai suatu format data berstruktur semi, telah menjadi satu standard bagi perwakilan dan pertukaran data melalui Internet. Pada awal sejarah XML terdapat pandangan tentang sama ada XML adalah berbeza daripada format data lain yang memerlukan pangkalan data yang tersendiri. Popularity dan penggunaan meluas XML di pelbagai organisasi telah melahirkan pandangan baru bagi amalam penyimpanan dan capaian data. Kebanyakan Di peringkat pemula, amalam penyimpanan XML adalah bergantung kepada pemetaan dan transformasi di antara pepohon data XML dan jadual pangkalan data hubungan dalam pangkalan data hubungan. Walaupun pangkalan data hubungan boleh mewakili struktur data sesarang dengan menggunakan jadual dengan kunci asing, ia masih sukar untuk carian struktur objek pada kedalaman sesarang yang tidak diketahui; sebaliknya, ia adalah satu kelebihan yang berpotensi dalam XML. Selain itu, elemen sesarang dan berulang dalam dokumen XML boleh dengan mudah menyebabkan sejumlah jadual tidak terurus. Seterusnya, ia biasanya sangat sukar selepas sisipan untuk menukar skema hubungan disebabkan oleh skema pertukaran XML. Had bagi pendekatan hubungan ini, kini diketahui umum. Selanjutnya, kemaskin setempat kepada dokumen itu tidak boleh menyebabkan perubahan drastik kepada sistem penyimpanan keseluruhan. Oleh itu, reka bentuk sistem penyimpanan perlu saling bertukar peranan di antara prestasi pertanyaan dan kos kemaskini. Kajian ini bertujuan untuk menilai prestasi pangkalan data XML natif (NXD) dibandingkan dengan pangkalan data XML_Enabled (XED) dan untuk meningkatkan Hubungan Entiti (ER) algoritma bagi skema hubungan untuk peningkatan sisipan, buang, kemaskini dan carian dokumen XML (XML fail dengan jumlah elemen yang besar) dan akhirnya untuk mengesahkan algoritma dalam NXD dan membandingkan prestasi XED dan NXD melalui pelaksanaan arahan dan kawalan data model yang sama. Lima saiz set data yang berbeza telah digunakan (65.8, 101, 117, 127, 183 MB). Teknik penanda aras digunakan untuk mengukur prestasi. XMark dan XMark-1 adalah dua alat utama penandaarasan dalam bidang penyelidikan ini dan telah digunakan untuk semua set data. Prestasi sesuatu sistem boleh diukur dengan menggunakan set data yang berlainan saiz, dokumen yang berbeza dengan ciri-ciri yang berbeza. Saiz dokumen XML dan bilangan elemen adalah ditentukan oleh faktor pemacu utama penjanaan. Hasil kajian ini menunjukkan bahawa XED mempunyai prestasi yang lebih baik bagi set data <= 117 MB. Prestasi XED mula menurun dengan peningkatan dalam saiz data XML (> 127 MB), manakala NXD menunjukkan prestasi yang lebih baik dalam untuk data (=> 127 MB). NXD menghasilkan keputusan yang lebih baik dalam bahagian laporan, yang membayangkan bahawa NXD X-pertanyaan mempunyai dapatan prestasi dari pengoptimuman pertanyaan. Hampir semua angka-angka ini menunjukkan bahawa XED bermula lebih baik, tetapi menjadi lebih teruk bila saiz data meningkat. Perbezaan menjadi jelas bila pertanyaan menjadi lebih rumit.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DBMS	Database Management System
DTD	Document Type Definition
EER	Enhanced Entity Relationship
NXD	Native XML Database
RDB	Relational Database
SQL	Structured Query Language
XED	XML_Enabled Database
XQuery	XML Query

CHAPTER 1

INTRODUCTION

1.1 Introduction

As the amount of data available on the Internet grows rapidly, more and more of the data becomes semi structured and hierarchical. The data has no absolute schema fixed in advance, and its structure is irregular or incomplete. Semi structured data arises when the source does not impose a rigid structure, and when the data is obtained by combining several heterogeneous data sources during the process of data integration. Semi structured data is often called schemaless or self-describing, because there is no separate description of its type or structure and its data (Ganguly, Sarkar, 2012).

The *Extensible Markup Language* (XML), as a format for semi structured data, has become a standard for the representation and exchange of data over the Internet. Using XML as a data representation standard, it is possible to represent not only the data itself, but also its semantics. XML was designed specifically to describe content, rather than presentation. It is a textual representation of data (Seligman and Rosenthal, 2001).

The popularity and wide-spread use of XML among a diverse set of organizations has engendered a rethinking of the storage and retrieval practices for data. Most early XML storage practices relied on mappings and transformations between XML data trees and relational database tuples within a conventional Relational Database Management Systems (RDBMS).

As a consequence, a single business operation might require numerous translations of the data from XML to relational table formats and vice versa at a significant cost in speed, reliability and efficiency of representation (Nicola and John, 2003; SC Haw, 2007) .

XML documents in relational systems would be simply mapped into existing relational database structures like "Large Objects" (LOBs), with XML being stored intact as plain unparsed text. Another approach used by RDBMs has been to create appropriate relational schemas and "shred" XML documents into many tables. The limitations of these two approaches are now well known. While inserting and extracting full XML documents is relatively fast in the case of LOB storage, this approach can be relatively slow during query processing and fragment extraction due to the need for XML parsing at query execution time (Nicola, and Linden, 2005; Zhang, and et al, 2009). While the "shredded" approach can provide reasonable performance given a good mapping, mapping an XML schema to an equivalent relational schema is usually a complicated job. Also, the nested and repeating elements in XML documents can quite easily result in an unmanageable number of tables. Furthermore, it is usually very difficult after insertion to change the relational schema due to XML schema changes. Thus, the flexibility of XML is essentially lost with this approach (Shao, 2010; Fiebig et al, 2002).

XML data is a challenge for relational databases. Hence the need to develop appropriate storage systems to store XML data and have the ability to speed the access of relevant and accurate information (Oracle 2012) Therefore, Native XML database becomes more and more popular and gained popularity as a flexible storage format. In Native XML database, XML data stored in the internal solid data model which retains the structure of the XML data. When storing XML data in a database, many questions arise:

(i) How to store the data in the databases?
(ii) How to express queries and updates?
(iii) How can XML data be indexed to speed up frequent queries?
(iv) How to detect if a modifying operation affects an established index and must therefore be updated to keep it consistent?
(v) How can indexes that are best for a given application determined automatically?

Thus, the querying process in XML data has become a challenge (Qtaish and Ahmad, 2014), and there has been a strong demand for improved query languages for processing XML documents. XML data is hierarchical structure (tree structure), and document data represented in XML comprise a sequence of possibly nested tags which can be expressed by a tree structure, and querying and transforming XML data from one format into another will be a frequent task.

This thesis focus on the path expressions in native XML databases. The main focus will be enhancing path expressions as expressive for database management systems (Native XML database and XML_Enabled database); path expressions specify special regular expressions on trees. In addition, path expressions as ubiquitous in many XML query languages (e.g., XPath (Berglund and Boag), XQuery (Boag et al) and XSLT (Clark).

1.2 Background

XML stands for extensible Markup Language, which is originated from Standard Generalized Markup Language (SGML, ISO 8879). It is a markup language designed to be relatively human-legible. Hypertext Markup Language (HTML) is another predominant markup language for web page design. It is also originated from SGML prior to the inception of XML. However, XML is not a replacement for HTML, as they are designed for different objectives, XML is designed to describe data and to focus on what data is. HTML is designed to display data and to focus on how data looks. In other words, HTML is about displaying information, while XML is about describing information.

Consider the bibliography information shown in Figure 1.1, and the corresponding HTML and XML representations in Figure 1.2. We can observe some differences between HTML and XML. In HTML, the tags and document structures are fixed. Only the tags which are defined in the HTML standard (e.g., *(hi), (p))* can be used. In XML, however, users can define their own tags (e.g., *(book), (title))* and document structure. As a result, it is more flexible and powerful. XML schema languages, e.g., Document Type Definition (DTD) and XML schema, are proposed to provide a high level abstraction of XML documents in terms of constraints on both structure and content.

Bibliography

Foundation of Database.Abibeboul, Hull, Vianu Addison Wesley, 1995

Data on the Web.Abiteboul, Buneman, Suciu Morgan Kaufman, 1999

Figure 1.1. Example of *Bibliography Data*

HTML DOCUMENY	XML DOCUMNENT
<h1> Bibliography of Database </h1> <p> Foundation of Database Abibeboul, Hull, Vianu Addison Wesley, 1995 <p> Data on the Web Abiteboul, Buneman, Suciu Morgan Kaufman, 1999	<bibliography> <book> <title> Foundation of Database </title> <Auther> Abibeboul </Auther> <Auther> Hull </Auther> <Auther> Vianu </Auther> <Publisher> Addison Wesley </publisher> < year> 1995 </year> </book> ..... </bibliography>

Figure 1.2. *HTML and XML* Representations of Data Example in Figure 1.1

XML is widely used in various scopes of applications: XML plays an important role in data exchange between different information systems. It can reduce the complexity of exchanging incompatible data formats; XML facilitates communications and data sharing between different systems and programming languages; XML is widely used for content management; with extensible Stylesheet Language Transformations (XSLT), XML documents can be transformed into other XML or "human readable" documents such as HTML, WML, PDF, flat file and EDI, etc.; XML enables integration of Web application; and many others.

With the popularity of XML and increasing amount of information stored and exchanged using XML, efficient hosting of XML stores and efficient processing of XML queries becomes important in the database community. To utilize the mature relational database technologies such as persistent storage, transactional consistency, recoverability, security, efficient search and update operations, many approaches are

proposed to store XML data in relational databases. They are typically based on shredding DTD/schema into relations. Typical examples include STORED, Edge (Florescu and Kossmann, 1999), XISS/R (Harding and Moon, 2003), XParent(Jiang et al, 2002), MonetDB(Schmidt et al, 2000), MOA (Van Zwol et al, 1999), XRel(Yoshikawa et al, 2001; Zhou et al, 2001) etc.

Also, conventional database vendors provide relational support for XML data, e.g., DB2 (IBM), Microsoft SQLXML (Microsoft) and Oracle 9i (Oracle). However, there is inherent impedance mismatch between the relational (sets of tuples) and XML (unranked trees) data models. An alternative approach to using relational databases for XML data is to build a specialized XML data manager, i.e., one which can reflect the hierarchical structure of the XML data. This is referred to as a native XML database. In native XML databases, XML data is generally modelled as trees, where tree nodes represent XML elements, attributes and text data, and edges for element/sub-element relationship. The tree representation of the XML document in Figure 1.2 is shown in Figure 1.3. Timber (Jagadish et al, 2002), XBase(Wang, et al, 2002), Lore (McHugh, and et al, 1997) (for *Lightweight Object Repository)* and several industrial products such as Tamino(Scheming., 2001) and XYZFind are native XML database systems. Our research focuses on query processing over native XML databases.

A novel data model requires query languages to take advantage of the storage features of that model. To retrieve tree-shaped XML data, different XML query languages have been proposed such as Lorel (Abiteboul et al, 1997), XPATH (Berglund et al), UnQL(Buneman, et al, 1996), XQUERY (Chamberlin et al. 2001), Quilt (Chamberlin and et al, 2000), XML-QL (Deutsch, et al, 1998) and Strudel (Fernandez, et al, 2000). XPATH and XQUERY are recommended by W3C and are regarded as the state-of-the-art query languages for XML databases.XQUERY on XML tree is analogous to SQL on relational tables. XPATH is a major subset of XQUERY and path expression is the core of XPATH. For example, the following path expression */paper [@conf= "ICDE"J[@year= "2000"j'/author* returns all *author* elements which have paper(s) published in ICDE 2000.

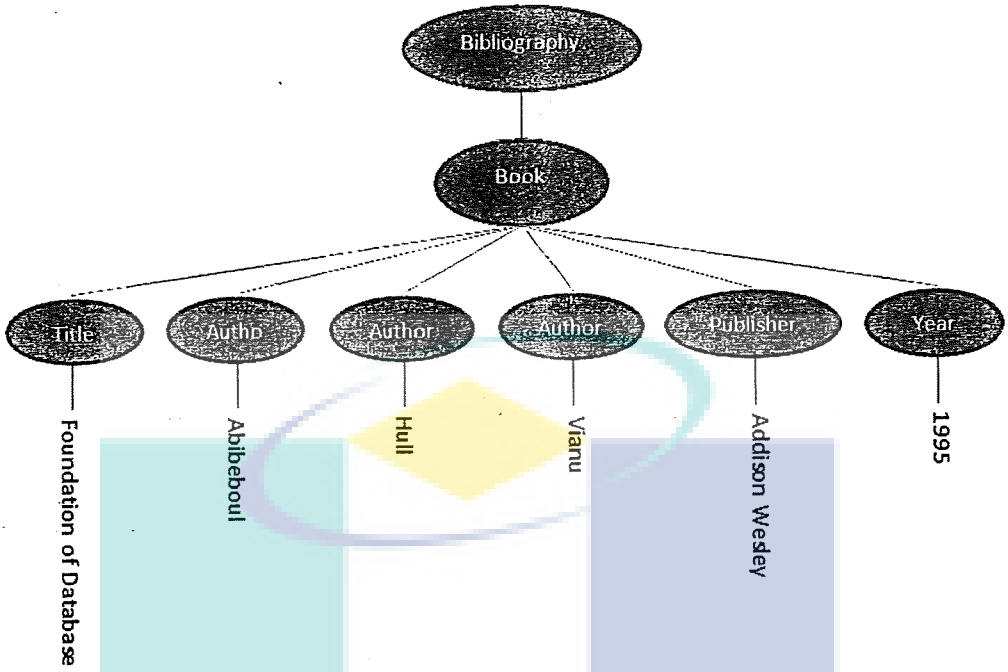


Figure 1.3. Tree Representation of *XML* Data in Figure 1.2

1.3 Problem Statement

There are several desirable features for the XML database storage systems. First, the difficulty of converting the shredding of XML data into relational tables increased with increasing the data size (Zhang and Tompa, 2004). Secondly, the storage system should be robust enough to store any XML documents with arbitrary tree depth or width, with any element-name alphabet, and with or without associated schemata. Moreover, local update to the document should not cause drastic changes to the whole storage system. Therefore, the design of the storage system should trade-off between the query performance and update costs.

There are three important problems related to the processing of path expressions:

1. What is the storage system for XML documents to support efficient evaluation as well as storing and updating XML documents?
2. How to evaluate path expressions efficiently for different types of queries?
3. How to choose which physical operators are the best ones given a path expression and an XML database?

Due to the mismatch between the nature of the structure of the database and the XML data structure. The problem for this study is:

(i) Decline in the performance of the XML_Enabled database

(ii) Degradation of performance of XML_Enabled database, when search of XML documents (Group of data).

1.4 The Objective of the Study

This research has three main objectives:

(i) To evaluate the Native XML database performance in a comparison with XML_Enabled database.

(ii) To enhance entity relationship (ER) algorithm of the relational schema for the improvement of Insert, Delete, Update and Search XML document.

(iii) To validate the algorithm in Native XML databases, and compare the performance of XML_Enabled database and Native XML database, by implementing the same command and control data model

1.5 The scope of the study

(i) XML language which is the standard language for the exchange of information, materials across the web more easily and smoothly than it was before. They also have an important role in building and sharing bibliographic searches, registrations resulting management systems, as well as exchange online.

(ii) XML language which holds great importance in the field of internet and information, it can achieve greater results with the use of new techniques for storing XML data and one of these techniques is a Native XML Database (NXD).

(iii) The use of Native XML database systems not only means evaluating the performance of current products, but also is regarded as an incentive for further development of XML database systems.

1.6 Research Plan

The research has two phases as shown in Figure 1.4:

(i) First phase: Explore the functionality of a Native XML Database in handling XML documents.

(ii) Second phase: Enhance the performance of the data queries based on Native XML Database, in comparison with XML_Enabled database.

The research is implemented as follows:

(i) The database size is measured for both databases to evaluate how efficient the database server to store XML documents. The size of building indexes is measured in megabytes.

(ii) Examine the efficiency and the time for XML document reconstruction from the database (XML_Enabled database and Native XML database) is measured in seconds.

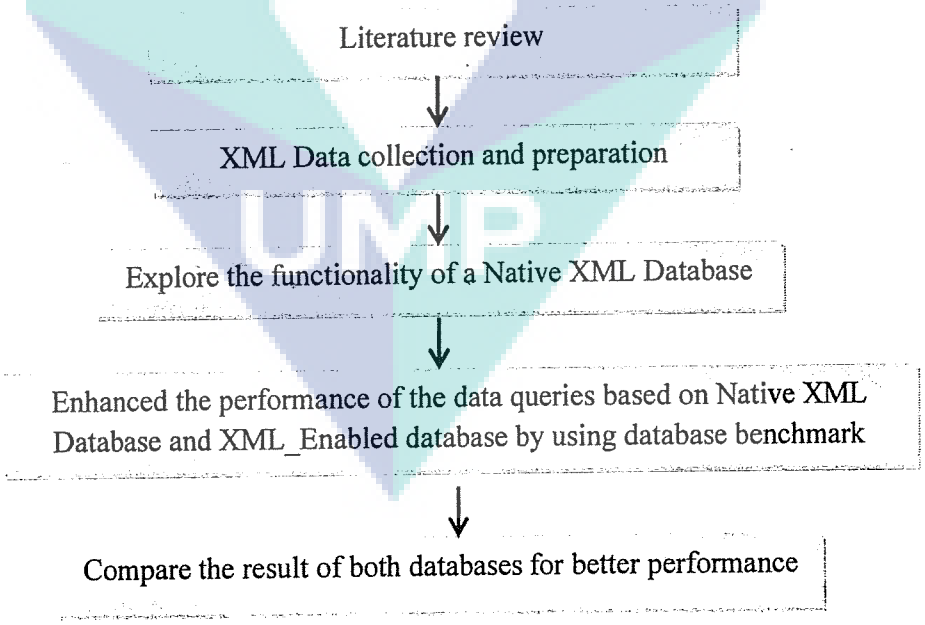


Figure 1.4 *Research plan*

1.7 Thesis Organization

This thesis contains five chapters following this introductory chapter, structured as follows:

In the Chapter 1, the framework of my thesis is presented such as introduction, problem statements, objectives and research scopes.

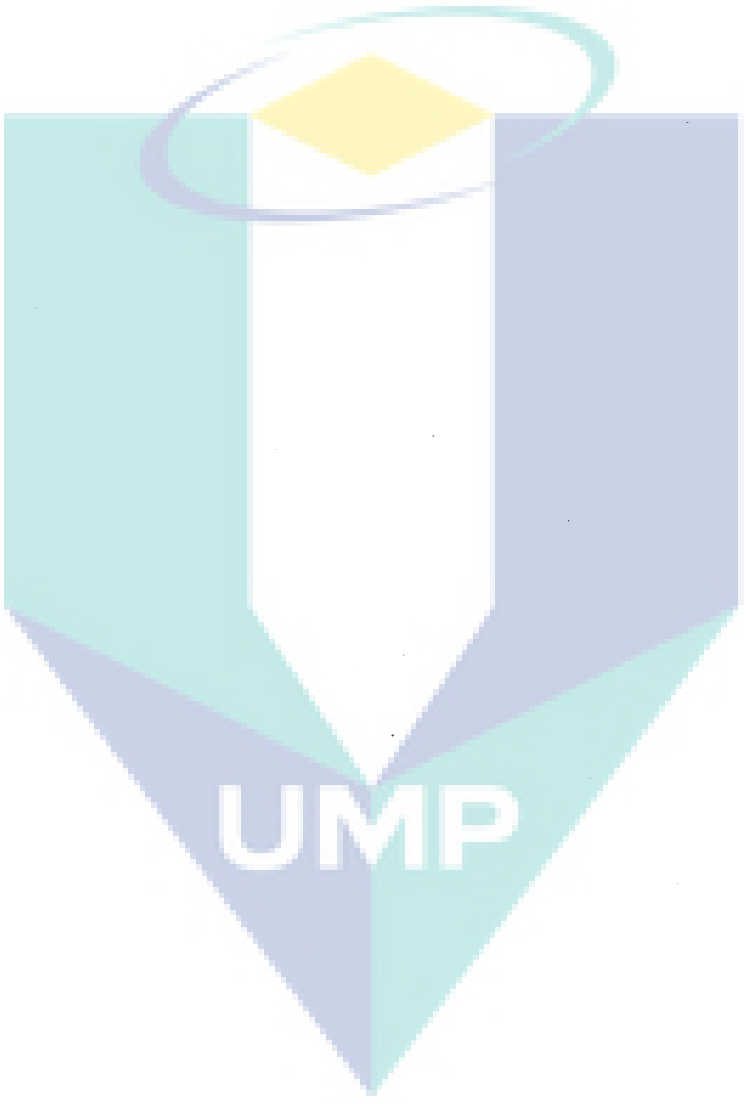
Chapter 2: This chapter discussed the literature review gives a quantity of Extensible Markup Language (XML) and its associated standards. The chapter starts with an exploration of the emergence of XML language and its advantages and disadvantages. The chapter explains the main primary types of XML, databases, and how these databases handle with XML documents. Additionally, conventional approaches discuss the query execution and the problems it face when processing queries on collections of XML documents.

Chapter 3: This chapter, shows the detailed view of the thesis methodology. The details on the concrete implementation of the storage and indexing components by using XML database system benchmark. Finally, explain benchmark queries engine uses the data sets collected.

Chapter 4: This chapter, shows the experimental settings to the detail view of database design and the data sets that used in the thesis experiment. The empirical part contains the testing of the research variables and discussion that involves synthesizing the results obtained and evaluates the effectiveness of both databases (XML_Enabled database and Native XML database).

Chapter 5: In this chapter, the extensive discussion is presented for storing and extracting complete xml documents, inserting process, updating process, deleting process and searching process. Also, validation results is further analyzed.

Chapter 6: This chapter contains the conclusion and a few suggestions for potential future work.



CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter introduce a brief description two databases: Native XML Database (NXD) and XML_Enabled Database (XED). The methods for both databases of dealing with XML data (Store, Extract and Search). This chapter also presented some of the previous studies since 1999 to 2014 and focus more on relational database shredding process problem because of the differences in the structures of the XML data. Finally, the briefly review these approaches as they relate to the following issues: representation, compression, labeling, and path processing and indexing

2.2 XML

As XML becomes a standard data representation and exchange format (Harrusi, et.al, 2006; Yang, et.al, 2010), because it's a flexible text format used to create structured documents and the way to describe the data and also contain the data as well. With the development of the Internet and breadth of knowledge, a significant amount of XML documents is being added to the network continuously in various application domains. As a result, effective management of XML data became an important technology, and a lot of difficulties and challenges faced by such as: storage method for XML and queries for these data, and these challenges draw increasing attention and cannot be overlooked. (Jing, et.al; 2011 and Yang, et.al, 2010).

For this purpose, many alternative ways to store XML data. Existing ways can be classified as native, semi structured, object-oriented, relational and object-relational. There common way is storing XML data in relational databases (DeHaan, et al, 2003; Florescu, et al, 1999; Shanmugasundaram, et al, 2001; and Shanmugasundaram, et al, 1999).

Relational approaches proposed to date are predicated on some pre-defined mappings between XML documents and relational tables. These mapping schemas are sometimes tailored to specific documents or document collections by using available DTDs (Document Type Definitions) (Shanmugasundaram, et.al, 1999)), identifying structure patterns in XML documents (Deutsch, et al, 1999), or utilizing XML data statistics and query workloads in a cost based manner (Wang, et al, 2003; and Bohannon, 2002). Alternatively, mappings can be defined generically to reflect the components of an arbitrary XML document (Florescu and Kossmann, 1999; and Florescu, et al 2001). However, the fact that XML documents do not require the existence of DTDs causes the DTD-dependent mapping methods to be inapplicable, and the generic mapping often exhibits poor performance in query processing, especially when "shredded" documents must be recomposed in answer to a query.

Other mapping methods may have some advantages for specific workloads, the common problem with RDBMS solutions in storing XML data is simply that they are relational, not hierarchical which reflects the XML true storage structure. Ultimately, some level of mapping to relational structures is required. RDBMS vendors over the last number of years, particularly the ones indicated in (Ha, Sing Lee, 2011), have appeared to alter their native database engine to allow XML documents to be parsed more like their native XML counterparts. Aside from simply storing whole XML documents in relational data structures, in (Jiang, 2011), it proposed two other methods, relational or table based mapping and object-relational mapping. The relational mapping approach sometimes referred to as "shredding" by decomposing a document, let us say, by element, attributes, and other document information. The databases mentioned above subscribe to this convention, but have recently made product improvements to limit some of the inherent weaknesses. One of these weaknesses, (Cahlander, et.al, 2010) points out, is limited support for complex XML structures. Also proposed, was the O-to-R mapping convention,

which maps the document like a "tree of objects" which could overcome some of the limitations of the shredding approach. However, the performance suffers if they are applied to a wide, unknown range of XML environments. Most unfortunately, in all of these approaches, once a mapping is defined, it cannot be changed without a major data reorganization and corresponding application reimplementation (Chaudhri, et.al, 2003; Cahlander, et.al, 2010; Jiang, 2011; Ha, Sing Lee, 2011).

The entire process of mapping XML is divided into different phases as following:

(i) Parsing The XML document

(ii) Mapping the XML document into the database

(iii) Query implementation on the database

The storage representation affects the efficiency of the query process. For a long time most of the data is stored in SQL databases, managed and queried by using the SQL language (Haw and Lee, 2011). Traditional databases using SQL (Structured Query Language) do not translate easily to transport formats that would allow them to be used across the World Wide Web (Houman, 2004). This thesis shows that Native XML database views combined with XQuery can provide surprisingly effective solutions to the problem of supporting query performance comparing with SQL in relational database.

Shanmugasundaram, et.al, (1999) have used the more conservative approach of using traditional relational database engines for processing XML documents conforming to Document Type Descriptors (DTDs). This approach has developed an algorithm that converts XML documents to relational tuples, translates semi-structured queries over XML documents to SQL queries over tables, and converts the results to XML. It turns out that the approach can handle most (but not all) of the semantics of semi-structured queries over XML data, but is likely to be effective only in some cases.

Yoshikawa and Amagasa, (2001) have used a novel approach called XRel to storage and retrieval of XML documents using relational databases. In this approach, an XML document is decomposed into nodes based on its tree structure, and stored in relational tables according to the node type, with path information from the root to each node. Using XRel enabled to easily construct XQL interface on top of the (object) relational databases. In this research limited extensions to types and functions, and did

not need any special indexing structure for query processing. However, some extensions could be needed; for example, abstract data types for synthesizing query results would be required if it implement XML–QL interface. Further, because the approach does not use a special full text search system, it may not achieve high performance on query retrieval. It is therefore important to develop abstract data types for improvement of performance.

Tatarinov and D. Viglas, (2002) have shown that XML's ordered data model can indeed be efficiently supported by a relational database system. This is accomplished by encoding order as a data value. In addition, they have proposed three order encoding methods that can be used to represent XML order in the relational data model. Furthermore propose algorithms for translating ordered XPath expressions into SQL using these encoding methods. Ordered XPath and XQuery queries were used to measure query performance, while XML element insertions were used to measure update performance. The results show that a relational database system can efficiently support most ordered XML queries.

Zhang and Wm. Tompa, (2004) have proposed an alter Native approach that builds on relational database technology, but shreds XML documents dynamically. This approach is to avoid many problems in maintaining document order and reassembling XML data from its fragments after shredding. Where are querying XML documents by dynamic shredding. Then the algorithm used to translate a significant subset of XQuery into an extended relational algebra that includes operators defined for the structured text data type, which is required for XML to be supported by relational database technology. The documents will be store in a column of text rather than chopping them into pieces to be mapped to relational tables (If a document is often updated, it may be desirable to partition the text into several "update units," such as chapters, that can be stored and modified individually, rather than editing and re-indexing the whole. However, the document doesn't need to be as highly fragmented as is typical when using static shredding). However, this approach is only feasible if appropriate structured text operators are supported.

Xing, et.al, (2007) have used an XML document management system which supports efficient storage, retrieval and key constraints for XML documents. The system is based on a mapping algorithm that translates a DTD to a relational schema. Based on the mapping, node groups are range indexed and shredded into the database. This study has been used two different sizes of datasets (0.5 and 10 MB), as shown in Table 2.1, and evaluate them by 18 Queries. It has been shown that queries are executed several times faster than using the methods in literature, and space usage is significantly reduced. Table 2.2 shows the time.

Table 2.1 Data Size

Name	**Size**
SigMod	0.5 MB
XMark	10 MB

Table 2.2 Query Processing Time

Queries	**Time/Second**
Q1	11
Q6	10
Q8	20
Q13	18

Source: Xing, et.al; (2007)

Dweib, et.al, (2008; 2009) proposed a new method for mapping XML documents to RDB. The method doesn't need a DTD or XML schema. Experimental results on this method shows its ability to maintain document structure at a low cost price and easily, building of the original document is straight forward, performing first level semantic search is achievable either on a single document or on all documents. The proposed method able 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. The study has been used datasets (4, 28, 64, 602 KB and 1 MB). Table 2.3 shows the datasets size and the time for mapping and reconstructing the documents.

Table 2.3 The time spent for mapping XML documents and the time for reconstructing them

	Document size	4 KB	28 KB	64 KB	602KB	1MB
Time/Sec.	Mapping	0.01988238	0.14977736	0 .3551445	3.574335	5.85278136
	Reconstructing	0.018990234	0.44980958	1.926836	18.305544	32.06255104

Source: Dweib, et.al; (2008)

Alghamdi, et.al, (2014) have adopted an optimization approach that takes into consideration the semantics of the data set in order to deal with the complexity of multi-disciplinary domains in Big Data, in particular when the data is represented as XML documents. The method for this study particularly addresses a twig XML query (or a branched path query) in relational database, as it is one of the most costly query tasks due to the complexity of the joint operation between multiple paths. The study focuses on optimizing the structural and the content part of XML queries by presenting a method for indexing and processing XML data based on the concept of objects that is formed from the semantic connectivity between XML data nodes.

The mismatch between the nature of the structure of the relational database and the XML data structure, leads to some problems and important issues such as: involved in storing data, retrieval from databases, and prevents some requests from being instantaneously accessible in the database. Thus, there is a problem of designing the database, and this defect in databases prevents the maximum utilization of flexible XML which leads to problems in performance such as: minimize the expected access time and degrade the performance (Oracle, 2014). Found that traditional databases are dealing with most (but not all) of the data (Such as semi-structured data), and it is suitable only for certain cases, and the disadvantages of these databases appear in the case of big amount data, thus much of the query and retrieval of data. Current research focuses more and more on how to manage XML data effectively. This led to the emergence of many XML database products, and XML is become gaining popularity as a flexible storage format and so are Native XML databases (Yu, 2005).

Table 2.4 Shows the summary of the literature review related to XML databases system issues that has been discussed in 2.2 and 2.6

Author	Year	Contribution
Shanmugasundaram, et.al	1999	In this approach that has been developed an algorithm that converts XML documents to relational tuples, translates semi-structured queries over XML documents to SQL queries over tables, and converts the results to XML. It turns out that the approach can handle most (but not all) of the semantics of semi-structured queries over XML data, but is likely to be effective only in some cases
Yoshikawa and Amagasa	2001	In this approach, an XML document is decomposed into nodes based on its tree structure, and stored in relational tables according to the node type, with path information from the root to each node. Using XRel enabled to easily construct XQL interface on top of the (object) relational databases. The approach does not use a special full text search system. Thus, it may not achieve high performance on query retrieval
Tatarinov and D. Viglas	2002	In this approach shown that XML's ordered data model can indeed be efficiently supported by a relational database system. This is accomplished by encoding order as a data value. The results show that a relational database system can efficiently support most ordered XML queries
Zhang and Wm. Tompa	2004	This approach is to avoid many problems in maintaining document order and reassembling XML data from its fragments after shredding because it is proposed an alter Native approach that builds on relational database technology, but shreds XML documents dynamically.
Xing, et.al	2007	In this approach used an XML document management system which supports efficient storage, retrieval and key constraints for XML documents. It has been shown that queries are executed several times faster than using the method in literature on 2001
Dweib, et.al	2008, 2009	In this approach proposed a new method for mapping XML documents to RDB. The method doesn't need a DTD or XML schema. Experimental results on this method shows its ability to maintain document structure at a low cost price and easily, building of the original document is straight forward.

Table 2.4 Continued

Author	Year	Contribution
Alghamdi, et.al	2014	This approach have adopted an optimization approach that takes into consideration the semantics of the data set in order to deal with the complexity of multi-disciplinary domains in Big Data, in particular when the data is represented as XML documents.

Based on the previous Table 2.4, this research issue is found from the researcher Zhang and Wm. Tompa in 2004 which used a Native XML approach in relational database and as following:

(i) Decline in the performance of the XML_Enabled database

(ii) Degradation of performance of XML_Enabled database, when search XML documents (Group of data)

2.3 Structured and Semi-structured Data

Searching for information is an indispensable component of our lives. Web search engines are widely used for searching textual documents, images, and videos. There are also vast collections of structured and semi-structured data both on the Web and in enterprises, such as relational databases, XML, data extracted from text documents, workflows, etc.

Structured data is a general name for all data that abides by a predetermined set of rules. These rules include defining types of data and also the relationships between them. This includes data contained in relational databases and spread sheets. Structured data first depends on creating a data model- a model with the data that will record and how they will be stored processed and accessed. This includes defining what fields of data will be stored and how that data will be stored data type (numeric, BOOK, alphabetic, name, date, address) and any restrictions on the data input (number of characters; restricted to certain terms such as Mr., Ms. or Dr.; M or F).

Semi-structured data is data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably. In semi structured data, the information that in normally associated with a schema in contained within the data, which is called "schema-less or self-describing," In some forms of semi-structured data

there is no separate schema, in others it exists, but only places loose constraints on the data. It generally has some structure, but does not conform to a fixed schema. The best example of semi-structured data is XML.

2.4 Extensible Markup Language (XML)

XML is the standardized language used for transporting both data and the data definition across the web. This language started out more than fifteen years ago in a design status by the W3C (World Wide Web Consortium) in 1996, and was originally used by very few. However, currently, XML has a permanent place in IT systems and it's hard to imagine any application that doesn't use XML for either its configuration or data to some degree (Houman, 2004). This fast growth is driven by its ability to provide a standardized, extensible means of including semantic information within documents describing semi-structured data. XML came to address the shortcomings of many markup languages such as HTML and support data exchange in e-business environments.

XML itself is a structured description language. XML is a simplified subset of Standard Generalized Markup Language (SGML) that its main purpose is to facilitate data sharing between different systems, which HTML (Hyper Text Markup Language) is also part of it. However, HTML, simply can be good for is formatting text, images, and forms. It can't store variables, while XML in essence designed to describe the data itself with focus on how data looks (Bishop, et.al. 2003 and Akmal, et.al. 2003).

Unlike HTML, XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications with focus on what data is (W3C, 2011). XML readily enables the definition, transmission, validation, description, and interpretation of data between different computing systems and applications. XML uses in many fields, such as chemistry, music, finance, or environmental data collection, simply it's become the universal standard for information interchange (Leigh, 2001, Bray, et.al., 2006).

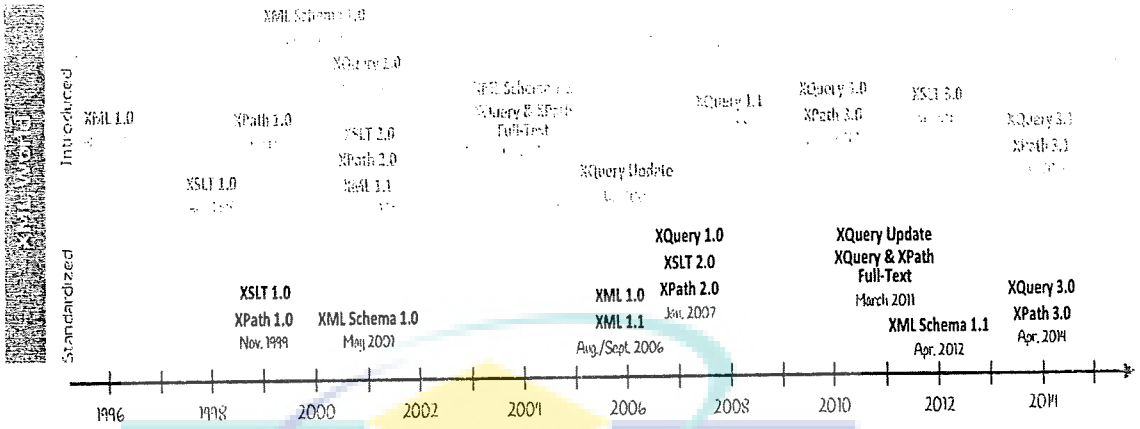


Figure 2.1 Timeline of the *W3C technologies* related to the XML.

Source: W3C.(2011)."News Archive".

2.5 Data-quide for XMLDocuments

Data-Guide is concise and accurate summaries of XML databases which describe different paths of XML documents, enabling schema exploration and improving query processing and processing in an XML database management system. Figure 2.2 show the Data-Guide which are working to ensure that each label path in XML data graph in Figure 2.3 research one node. However the Data-Guide but didn't prevent multiple label paths from reaching the same Data-Guide node (Kalinin, 2009).

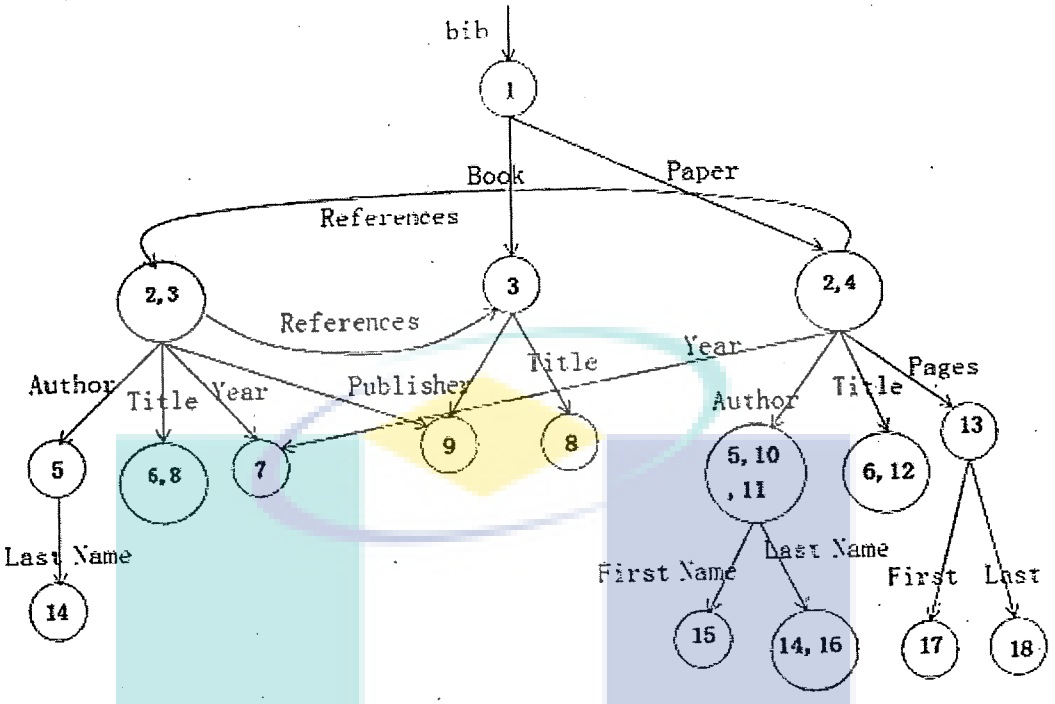


Figure 2.2 Data-Guide s of an *XML tree*

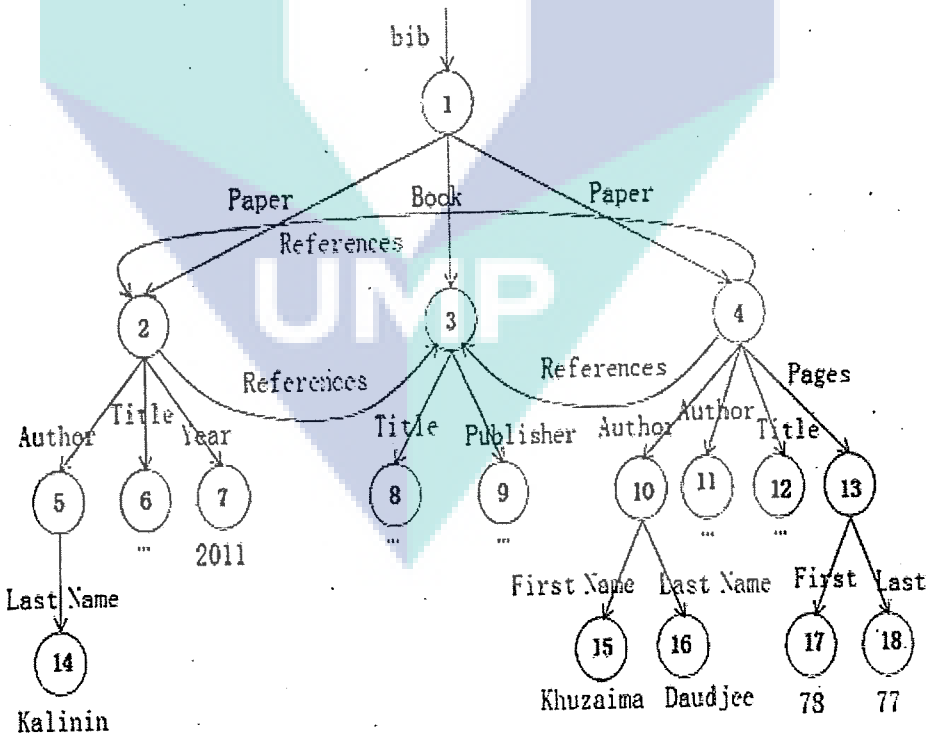


Figure 2.3 Example of *XML Data Graph*

The Figure 2.2 is an example of Data-Guide which one provides a classification of nodes or objects in the data:

(i) Node 2 and 4 are papers
(ii) Node 5, 10 and 11 are authors of papers
(iii) Node 2 and 3 are referenced by papers
(iv) Node 6 and 8 are titles of objects referenced by papers
(v) Node 3 is both a book and an object that is referenced by an object that is referenced in a paper

Data-Guide for XML document properties as follows:

(i) For every path in the document, there is a unique equivalent path in the Data-Guide
(ii) For every path in the Data-Guide there is an equivalent path in the document

2.6 XML Database Systems

A database is a collection of data well organized in digital format, and can access to such data and manipulated by a data - processing system (Tzvetkov and Xiong, 2005; Jeffrey and Jennifer, 1997). Thus, the question arises: "Is XML a database?"

An XML document is a database only in the strictest sense of the term. XML is actually just a file containing a collection of data, and these files can be used as databases, although has no database functionality, because XML is a language made to contain and transport smaller amounts of information, in a logical, human readable, platform independent way (Harold, 2003). The XML used to create files that need to be sent to other applications. XML is more suited as a data interchange format than as data storage format, because it is self-describing, portable (because of the use of Unicode), and that it can describe data in tree or graph structures (Glas, 2002).

The integrity management, such as multi-user access control and security management, backup and recovery management, things that belong to a Database Management System (DBMS) (Rob-Coronel, 2000). The DBMS base on a combination of the XML document with all kinds of XML related tools, like XML schema, Query languages and programming interfaces, still not be as efficient and thorough as a real DBMS. Therefore, DBMS is the best way to store the XML documents for fast and accurate multi user.

2.6.1 Xml-Enabled Database System (XEDs)

Figure 2.4 shows The XML_Enabled database as a relational database that transfers data between XML documents and relational tables. It retrieves data for maintaining the relational properties between tables and fields, rather than to model XML documents (Pardede, 2008).

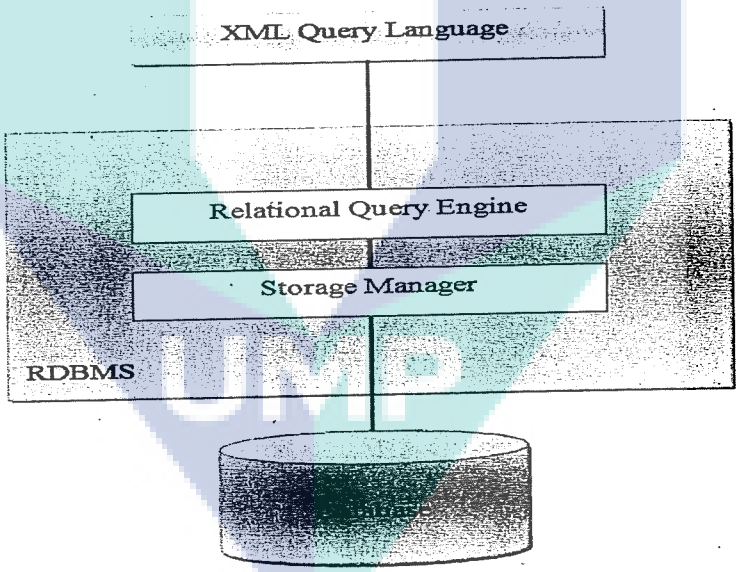


Figure 2.4 Architecture of the *XML*_Enabled Database.

Source : Hiddink (2001) ."ADILE: Architecture of a Database-Supported Learning Environment"

XML_Enabled database (Bourret, 2002) firstly, broken down (Shredding) XML document in small parts into their component elements, attributes, and other nodes for storage these parts into columns as a basic objects data in one or more relational tables as shown (Figure 2.5) (Akmal, et.al., 2003; Leigh, 2001;Papamarkos, 2011). Depend in

Figure (2.6) on mapping the XML document schema (XML Schemas, DTD, etc.) database schema (Papamarkos 2011) in design time which called shredding. The process between XML_Enabled database and XML documents needs schema translation, this translation helps in sharing data with other systems, and interoperability with incompatible systems.

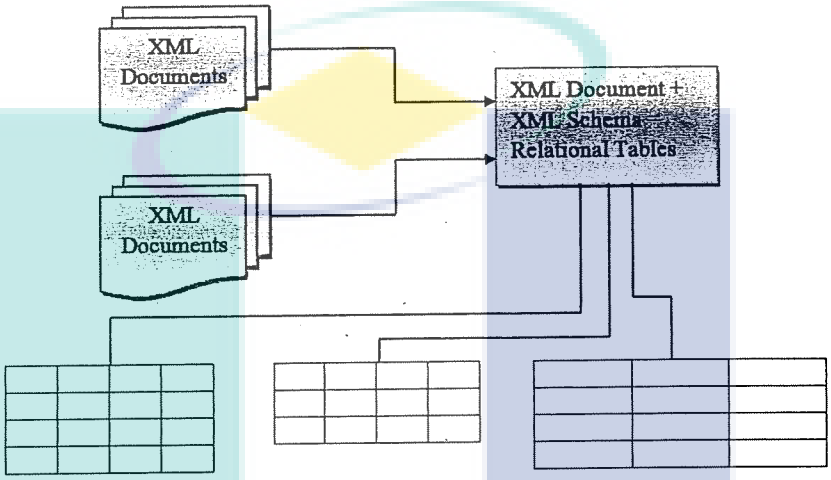


Figure 2.5 *XML* documents need schema translation to decompose *XML* documents into relational tables

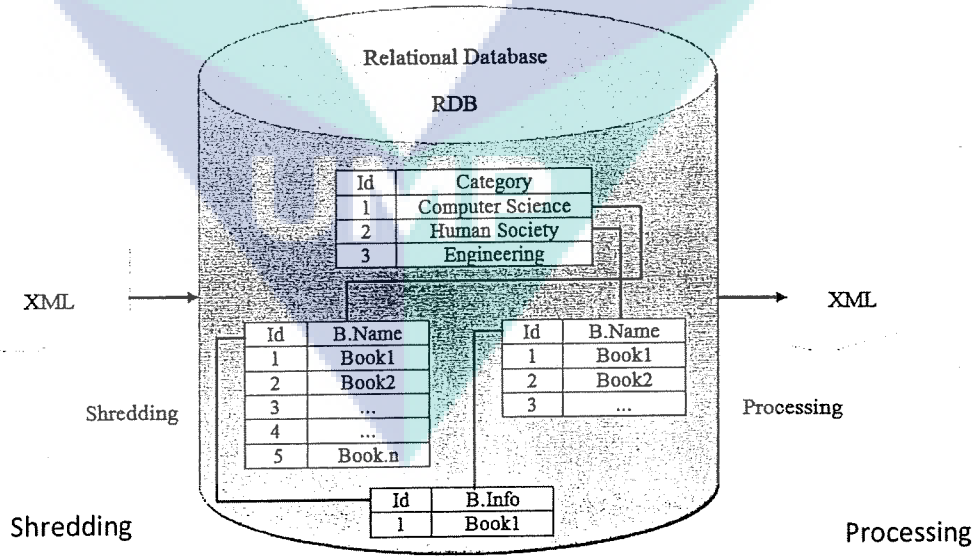


Figure 2.6 *XML*_Enabled database shredding *XML* documents into relational tables

Shredding process means convert the data from a document in the database and then reconstructing the document from that data, often results in a different document.

In addition, some problems that occur are as a result of the need for mapping, because of any changes in the XML schema that will lead to changes in the mapping as well (Henk, 2006).

Shredding can be done in a very naive manner, such as defining a SQL table for each element type (at least those allowed to have mixed content) in a document, with columns for each attribute, the non-element content of those elements, and the content of child elements that are not allowed to have element content themselves (Shanmugasundaram, et.al, 2011). As shown in Figure 2.7.

XML data shredded into relational tables:

1
```
CREATE TABLE book table (
book_id          INTEGER PRIMARY KEY,
FOREIGN KEY(book_id) REFERENCES book_table(book_id) )
```

2
```
CREATE TABLE book_table (
Book_id          INTEGER PRIMARY KEY,
ISBN             INTEGER,
Title            CHARACTER VARYING(100),
Publisher_id     INTEGER,
Year             INTEGER,
Author_id        INTEGER )
```

3
```
CREATE TABLE Author_table (
Author_id        INTEGER PRIMARY KEY,
givenName        CHARACTER VARYING( 50 ),
familyName       CHARACTER VARYING( 50 ) )
```

4
```
CREATE TABLE Publisher_table (
Publisher_id     INTEGER PRIMARY KEY,
Name             CHARACTER VARYING( 50 ),
Year             INTEGER )
```

Figure 2.7. Shredding *XML* document in Figure 2.8 to relational tables

Figure 2.8 is an example the process of reconstructing and assembling the original structure to restore the XML document is a difficult task requires complex multi way joins (Nicola and Kumar, 2010). Figure 2.9 shows a coding of complicated query discover the names of the tables and columns, and then join the various tables together on their respective PRIMARY KEY and FOREIGN KEY relationships. Usually, has provided a variety of ways of restructuring the XML documents from the shredded parts by vendors of the products. However, the large number of shredded data in the tables can lead to a complex and unnatural objects that makes application development difficult and error-prone. In addition, complex XML documents don't lend themselves to naive shredding techniques, and this may expend greater CPU power and cause performance degradation (Henk, 2006; Fernando, 2012; Melton and Buxton, 2006).

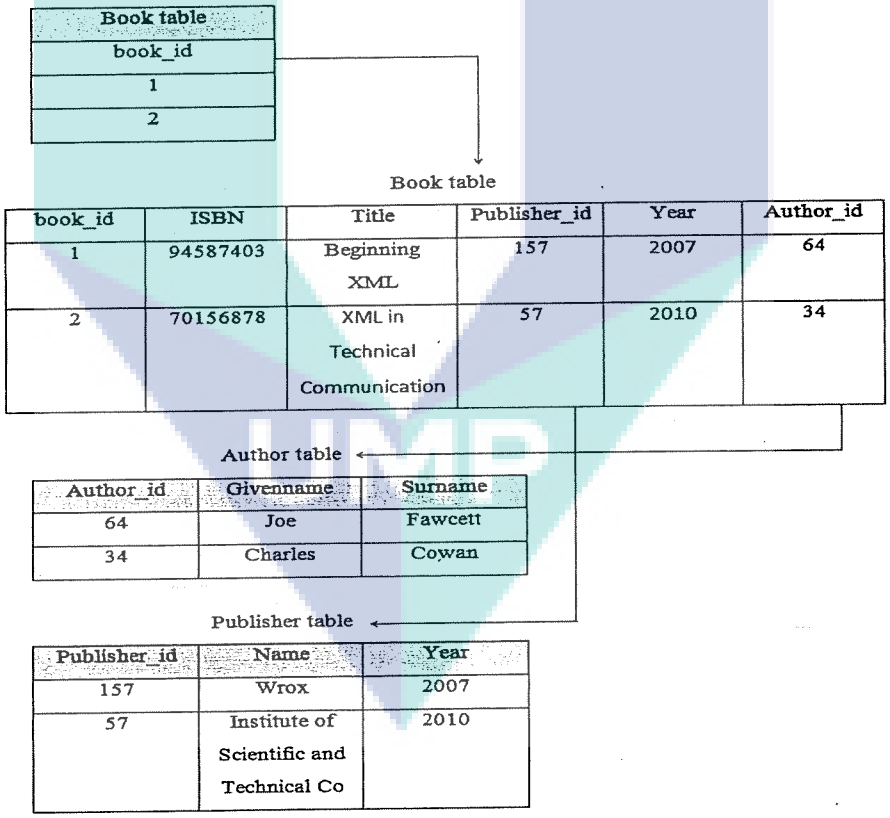


Figure 2.8. Shredding *XML* document to many relational tables

```
1   <books>
2   <book ISBN="94587403">
3    <title>Beginning XML</title>
4    <Publisher>Wrox</Publisher>
5    <year>2007</year>
6   <Author>
7    <givenName>Joe</givenName>
8    <familyName>Fawcett</familyName>
9    </Author>
10   </book>
11  <book ISBN="70156878">
12   <title>XML in Technical Communication</title>
13   <Publisher>Institute of Scientific and Technical Co </Publisher>
14   <year>2010</year>
15  <Author>
16   <givenName>Charles</givenName>
17   <familyName>Cowan</familyName>
18   </director>
19   </book>
20   </books>
```

Figure 2.9 *XML Books Document*

Relational databases are not predictable or reliable in terms of consistent performance for different reasons:

(i) Performance: A major problem and thus disadvantage by using relational databases is system performance. The number and size of tables and also the relationships between the tables to be established affect the performance in responding to the SQL queries.

(ii) Complexity: The strength of the Relational model is the Simplicity of tabular data-model, and it's also in some circumstances its weakness. Because of it is not an easy task to compress some of the complex relationships which exist in the real world into tables. For example, relational database doesn't allow the inclusion of complex object types such as graphics, video, audio, or geographic information. The need to include such complex objects in the database led to the development Native Database.

(iii) Physical Storage Consumption: A relational database requires consume computer memory and processing time (Cugnasco, et.al, 2013 and Xu, 2013).

The data within a traditional relational database is the data that exhibit the following characteristics:

(i) Repeating many fields and structures, even in a simple hierarchical representation of the data, this leads to a large number of tables in the representation of the second and third model.

(ii) The wide variety of structures.

(iii) Sparse tables (Vavliakis, 2013).

2.6.2 Native XMLdatabase System (NXDs)

Native XML Database (Figure 2.11) is a data storage format that appears in the field of XML data processing and allows XML data to be maintained in their format (tree structure). Then the data can be queried and processed as XML without the need to transform it into traditional database. Native XML databases developed from standards developed by the World Wide Web Consortium (W3C). Which create a powerful architecture for connecting XML data management services as shown in Figure 2.10. The XML databases that use and support these standards provide capabilities couldn't find in other database technology (such as relational databases), including efficient access, query, storage, and processing (Mabanza, 2010, Bourret, 2005 and Pokorny, 2008).

Undoubtedly the use of Native NXD is a new challenge for both developers and researches of database systems, and it has been brought great opportunities, making expansion of the database technology and research into data management on the WEB possible (Kolarand Loupal, 2006).

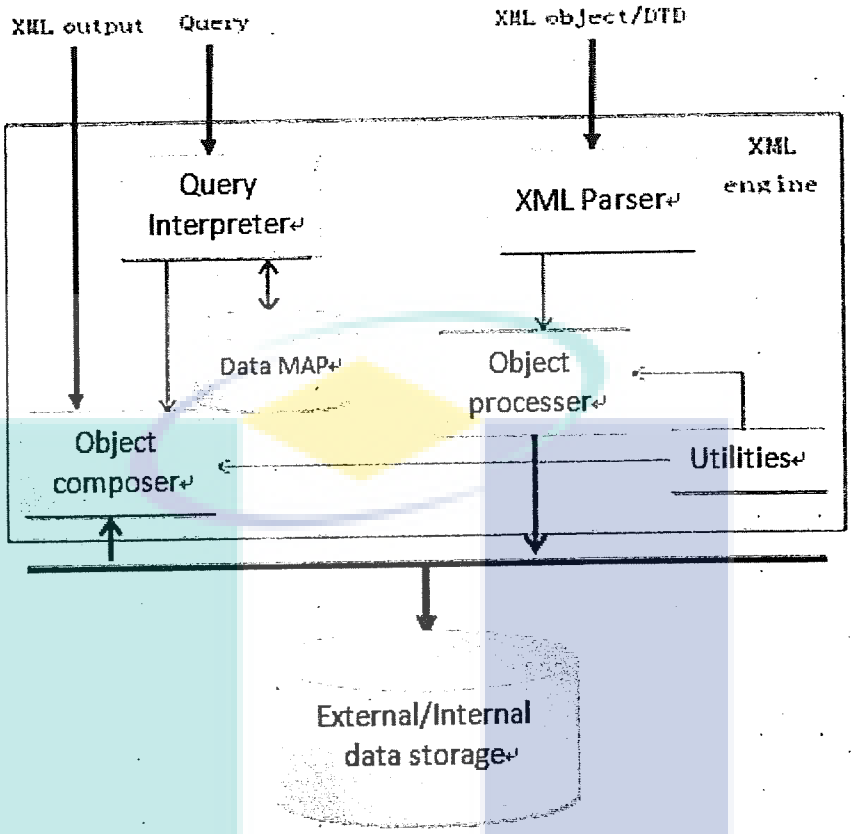


Figure 2.10 Architecture of Native *XML* Database XML Engine

Native XML database is high-speed storage and designed for manipulation of very numbers of XML documents by using a set of fixed structures, and store XML documents in an integrated, highly scalable, and high performance. In addition processing and storage XML data within their applications, because NXD model is based on XML and not something else, and can store any XML document such as structured and semi structured data . (Bourret, 2007;Bourret, 2005; Xiaomei and Heng, 2010; Apache XML Project Group, 2011).

Native XML database store XML documents as a unit and create a model which is consistent closely with them, which allow the query and manipulate those documents as a group. In addition, can be stored any XML document in the group, regardless of the scheme, and this is called schema-independent.

Native XML database is in order to meet the following three conditions:

(i) The pattern is defined as an XML document; the XML document is stored and retrieved according to the model.

(ii) XML document as a basic storage unit, such as the row in the tables as the basic unit of storage in the relational database.

(iii) XML data does not require any special underlying physical storage model (Jiang, 2011; Ha, Sing Lee, 2011).

Native XML database provides numerous features to help you process and handle XML documents, including:

(i) An XQuery engine for retrieving specific parts of a document, whereas almost all Native XML databases support one or more query languages, such as XPath and XQuery.

(ii) A versioning mechanism for tracking differences within your XML data.

(iii) Various indexing methods to optimize access (as a way to increase query speed) to frequently used XML data and to enable full text search.

(iv) A Native XML Database resolves the problem of semi-structured data (Semi-structured data is just data that do not fit neatly into the relational model).

(v) A transformer and formatter for publishing XML data in XHTML or PDF.

(vi) An improved cache page replacement algorithm for enhanced performance.

Where is a Native XML Database used?

(i) Native XML databases are most commonly used to store document-centric documents. The main reason for this is their support of XML query languages, which allow you to ask queries that are clearly difficult to ask in a language like SQL.

(ii) Native XML databases are also commonly used to integrate data. Native XML databases handle schema changes more easily than relational databases and can handle schemaless data as well.

(iii) Native XML Database is used with semi-structured data, such as is found in the fields of finance and biology, which change so frequently that definitive schemas are often not possible.

(iv) Native XML Database is used in handling schema evolution.

2.6.2.1 Native XML Database Products

Table 2.5 shows the two types of Native XML data product. The first type is open source such as: Lore, 4Suite, Xindice, eXist and DBDOM. The second type is commercial products such as: DOM-Safe, Tamino, XIS, GoXML DB, TEXTML Server and Berkeley DB XML. Table 2.5 introduces open source Native XML database products.

Table 2.5 Summary of Native XML Database Products

	Product	Developed by	Year	Data Type	Developer	Platform	Query Languages
Open Source	Lore	Stanford University	2000	Semi-stuctured data	Stanford University	SUN Unix,Linus	Lorel
	eXist	Wolfgang Meier	2009	Stuctured and Semi-Stuctured data	Wolfgang Meier	Windows, Unix,Linus	XPath,XUpte,XQuery
	4Suie, 4Suite Server	Forethouht, Inc	2001	Stuctured and Semi-Stuctured data	FourThouht	Windows, Unix	XPath
	DBDOM	K.An Krupnikov	2002	Stuctured and Semi-Stuctured data	K.An Krupnikov	Windows, Unix,Linus	XPath
	Xindice	Apache Software Foundation	2002		Apache Software Foundation	Windows, Unix,Linus	XPath,XUpte
Commercial	GoXML DB	XML Global Technologies	2002	Stuctured and Semi-Stuctured data	XML Global Technologies	Windows, Unix,Linus	XPath,XQuery
	Tamino	Softeware AG	1999		Softeware AG	Windows, Unix,Linus	XPath,XQuery
	TEXTML Server	DITA CMS Group	1998	Stuctured and Semi-Stuctured data	IXIA.Inc	Windows	XPath

2.6.2.2 EXist-db Architecture

The overview of eXist-db system architecture is shown in Figure 2.11. Be dealt with all calls backend storage by broker class, implementing the database broker. Applications may access a remote eXist server via the interfaces such as: XMLRPC

XML, SOAP, Rest, and WebDAV. API interface supported by eXist, which supports the embedding the database into an application without running an external server.

Currently, eXist's XML RPC interface makes multiple RPC calls for retrieving the XQuery results. The first RPC just gets the ids of the XML document and the positions of the nodes to be displayed within each document. Then a loop is run where in each iteration an RPC call is made by passing the XML document id and the position of the node id to get the actual XML node content. This was modified to have a single RPC call to get all the required nodes of the XML document as a string. This enabled collation of results from various distributed servers without congesting the network. The XPath engine is used to parse the XPath to find the clusters which are to be queried (Chaudhri, et.al, 2003; Cahlander, et.al, 2010).

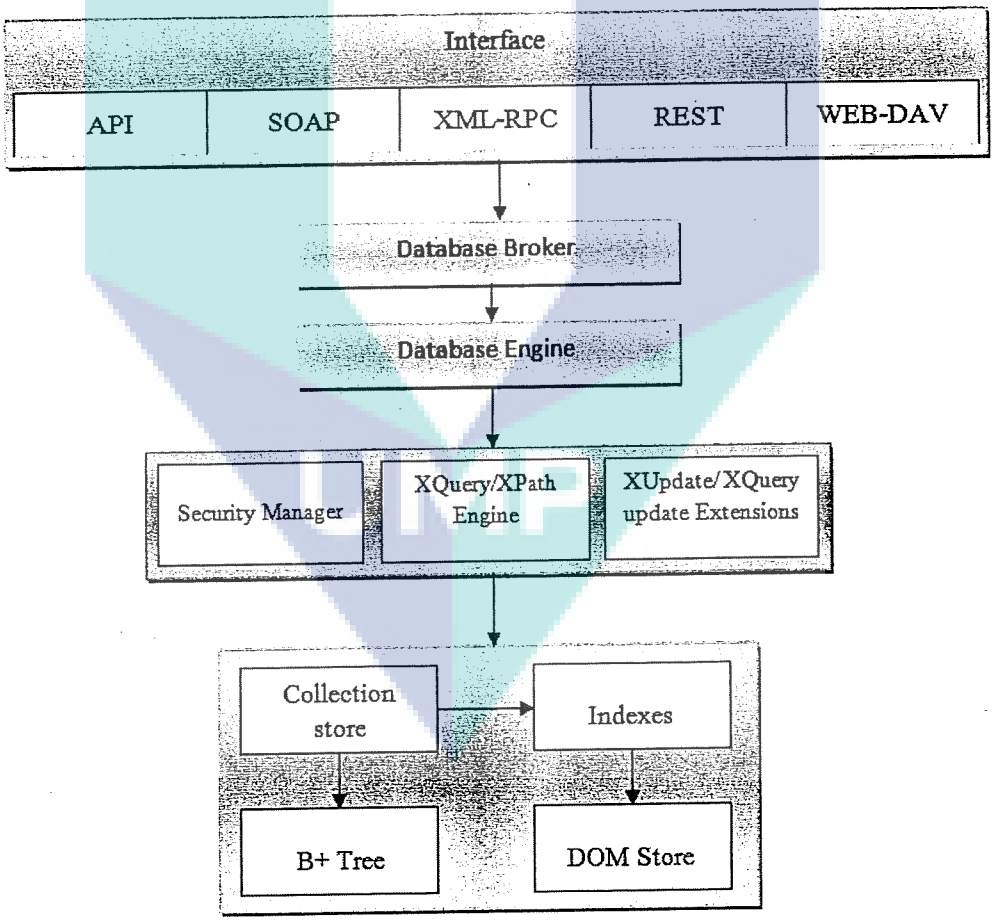


Figure 2.11 *eXist-db Architecture system*

2.6.3 Comparison between Native XML Database and XML-enabled Database Systems

Native XML database and XML_Enabled database are both used to store XML document, but they both have different techniques. XML_Enabled database method is to transform XML documents into relational tables by map document's schema to a database schema, and transfer the data by using this mapping, which can then be queried by using SQL. XML_Enabled database has its own data model to map XML instances into the database, and depending on this model creates tables (Papamarkos et.al, 2011). In another word, XML documents have to map into another data structure that can be stored in a traditional database system (Pavlovic, 2007). While Native XML database method store XML documents by using a fixed set of structures that can store any XML document regardless of the scheme. NXD uses XML model directly to map XML instances into the database. NXD uses tables for holding arbitrary XML document like Elements, Attributes, Text, etc., and XML documents are regarded as a basic storage unit (Bourret, 2007; Egbert, 2007, Yu, 2005).

The storing process for XML data in a relational model that based on two dimensional tables is one of the more main differences between the relational databases model and XML structure. The two dimensional model is not, neither hierarchy nor significant order, while XML is a hierarchical structure (Tree-Structure) in which order is significant, that use the sequence ways to represent the information (Robie, 2003). In addition, storing the XML data in a Native XML database the internal representation of the data retains the structure of the XML data, and this make NXDs useful, because the structure of them is closely matched the structure of XML data. Retaining the structure of the XML data removes the need for creating a mapping for the XML data. Changes in the XML schema or DTD of the XML data do not lead to the problems that are encountered with relational storage. Native XML storage can handle all types of XML documents without any problems (Härder and Haustein, 2008; Egbert, 2007). Table 2.6 summarized all advantage and disadvantage of both systems.

In general, XML-enabled database systems are generally relational RDBMS or object oriented OODBMS systems that contain extensions for transferring data between XML documents and native storage constructs. For the strength of existing algorithms,

there is little doubt that available commercial databases such as Oracle (10G), MS SQL Server, and IBM DB2 have cornered the market with regard to large enterprise database offerings on varied platforms (MAINFRAME, WINTEL, UNIX, LINUX, etcetera). However, until recent years they were under represented when offering native XML storage solutions. This increasing predominance of XML as a growing industry standard for the exchange of data as well as the storage of data in XML format, has led to development of a number of offerings (or extensions to existing DBMSs) that support the native storage of XML. Furthermore, these database systems are generally designed to store and retrieve data-centric documents rather than whole XML documents, although they do have that capability. Both for RDBMS and OODBMS there is inherent need for transformation between XML formats and native formats, and in the case of Oracle l0g, relational table-columns and XML documents.

Table 2.6 Comparison between XML_Enabled databases and Native XML databases

	Type	Description	Advantage	Disadvantage
XML_Enabled Database	Traditional Relational database	Map the data to relational sheet	Guarantee consistency and security, store index and query is very mature from theory to application	Likely to cause data distortion, many null values and redundant in conversion process of the relational sheet
	Object-orinted	Consider the data as CLOB field and store into relational database	Similar to file system storage, but guarantee consistency and security	Difficult to archive mass data index and query
File System	File System storage	Store directly into the file system in document form	Simple storage, lossless data, multi-dimensinonal and network relational preserved	Not guarantee consistency and security, difficult to archieve mass data index and query
Native XML Database	Native XML database	Store directly database dedicated to process XML documents	Has the advantage of the file system, easily index and query the data	Most Native XML database can only return the data as XML

According to analysis of the existing algorithms in Table 2.6, while the major drawback for XML-enabled databases is the representation of XML documents relational form, the major strengths include the impressive support users get with the underlying database, such as Oracle l0g: scalability, concurrency, recovery, multiple support applications, etcetera. This one benefit cannot be overstated, in that it has led to the universal presence of these XML storage solutions in the marketplace. In multi-document scenarios, since NXDs are tasked with storing a collection document, the matter of equivalence is important here. Some NXDs, like eXists, provide utilities to compare stored documents.

2.7 Entity Relationship Algorithm

The researcher used object-relational DBMS schema used to hold the XML document schemas and data, the user-defined types needed to make hierarchical operations efficient, and the algorithm used to map XPath expressions to their corresponding SQL-3 queries.

The figure 2.12 shows Data Definition language that has been required. Each different sub-set of XML data will store in separate pairs of tables. Each of these tables is implemented to model aspect of the XML schema and data model specification. The Specification included the following types of XML document nodes:

(i) Document
(ii) Element
(iii) Attribute
(iv) Namespace
(v) Processing Instruction
(vi) Comment
(vii) Text

In addition, XML Data Model includes a set of 19 primitive atomic types. Each of these atomic type value in the document must be one of these types

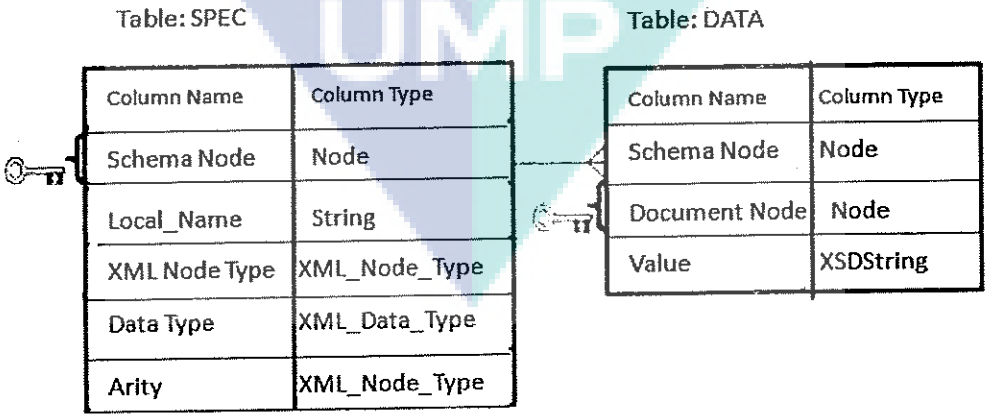


Figure 2:12 *Data Definition language*

2.8 Querying XML

Query languages are specialized language that used to make queries into database and information system (Gabriel and Mycroft, 2013). The query language provides the ability to send data manipulation and formatting requests to the data source, and ensure that the returned data structure and contents match the expected structure (Google Developers Group, 2013).

The eXtensible Markup Language (XML) is a standardized notation for documents and other structured data (Mamčenko, 2004), and XML has become as a standard for the exchange of data via the Internet (Chienping, et. al., 2011), and XML will become the format of tomorrow's data and web resources. Thus, the querying process in XML data has become a challenge (Qtaish and Ahmad, 2014), and there has been a strong demand for improved query languages for processing XML documents. XML data is hierarchical structure (tree structure), and document data represented in XML comprise a sequence of possibly nested tags which can be expressed by a tree structure, and querying and transforming XML data from one format into another will be a frequent task.

The traditional methods suffer from two drawbacks: (1) Weaknesses in enhancing the performance of the query without an adjustment the existing query processing engine; and (2) an Inability to be suitable for different structures and usage characteristics. Query languages can be classified into two types: Structured Query Language (SQL), and XML Query languages (XQuery) (Bourret, 2010).

XQuery and SQL/XML are two standards that use declarative, portable queries to return XML by querying data. In both standards, the XML can have any desired structure, and the queries can be arbitrarily complex. XQuery is XML-centric, while SQL/XML is SQL-centric (Robie, 2003).

2.8.1 Structured Query Language (SQL)

SQL is the most mature query language for XML_Enabled databases (Houman, 2004), and it can only be used with relational databases (Bourret, 2009). SQL cannot

use XML data yet, but with the use a set of extensions that are extending the SQL to handle XML. These extensions give a possibility to SQL for creating XML documents (Ki Min, et.al, 2008). Structured Query Language depend on probabilistic model for information, that's where this model is working to estimate probability of relevance of each document given a query.

SQL queries allow creating XML structures with a few powerful XML publishing functions. SQL is flat and needs to specify a value for each field in a record set (even null in case of an empty field). Thus, SQL is not useful for semi-complex XML files. For example, one characteristic of XML is having an element that can have of sub elements, each sub element can again have sub elements (tree can contain sub trees) and so on. Thus, makes one record set have a large or even theoretically unlimited range of depth, and this is this is impossible for relational databases (Houman, 2004).

The figure describes an XML structure that would be very awkward in a relational database. A warehouse has a category, which has several different products (in this example books and papers). The more data moves into an XML document, the more complex it gets, the limitations of SQL become clearer and the importance for another query language becomes more obvious.

2.8.2 XML Query Language (XQuery)

XQuery is the language of queries, concise and easy to understand, and it's to XML what SQL is to database tables, and it is designed to query XML data (Searching, finding, extracting, retrieval for any elements, attributes, data source within XML documents (Ykhlef and Alqahtani, 2007) with at least the functionality of SQL, where it's flexible enough to query a broad spectrum of XML information sources, such as querying collections of XML data and not only XML files, but anything that can appear as XML, including databases. XQuery is a Native XML query language. XQuery is optimal for processing XML naturally unlike SQL. In addition, XQuery is also useful for process XML data in XED not just in NXD, to query XML stored inside or outside the database. This makes most of the major database vendors intend to support XQuery (Robie, 2003).

XML is an inherently recursive data structure (Trees contain sub-trees), and XQuery is a functional language, where it's built on XPath expressions, so many XQuery functions for transforming documents are best designed using recursion. XQuery is supported by all major databases (Vendors like IBM, Oracle, and Microsoft) because it's a W3C recommendation.

XQuery is unique in the development stacking in that it replaces both SQL and the traditional software layers that convert SQL into presentation formats such as HTML, PDF and ePub. XQuery can both retrieve information from your database and format it for presentation. Much research has been done on XML query languages and XQuery is compatible with several W3C standards, such as XML, Namespaces, XSLT, XPath (Berglund, et.al. 2010), and XML Schema, XQuery (Erwig, 2003), Quilt (El-Sayed, 2005), in addition XML query algebras such as Lore (McHugh. et.al, 1997), XAL (Frasincar, 2002) and YATL (Christophides, 2002). XML algebra as a formal basis for an XML query language provides a solid ground to define the semantics of a query language and describes the procedures to obtain the answer by its power of expression. On the other hand, XML query languages provide the means to extract and manipulate data from XML documents. Although most of the query languages differ in detailed grammars and representation, they share a common feature, that is, queries usually make use of a path expression for query evaluation (Haw and Lee, 2011).

The principal benefits of XQuery are:

(i) Expressiveness - XQuery can query many different data structures and its recursive nature makes it ideal for querying tree and graph structures

(ii) Brevity - XQuery generate summary reports because its statements are shorter than similar SQL or XSLT programs.

(iii) Flexibility - XQuery can query both hierarchical and tabular data

(iv) Consistency - XQuery can used with other XML standards such as XML Schema data types because it is a consistent syntax

(v) Search and extract Web documents for relevant information to use in a Web Service.

XQuery depends on expressions; these expressions include seven major types: path expressions, element constructors, expressions involving operators and functions, conditional expressions, quantified expressions or expressions that test or modify data types, and the most important expression is "FLWOR" expression.

A FLWOR expression is a query construct composed of the following, in order, from which FLWOR takes its name: FOR, LET, WHERE, ORDER BY, and RETURN.

XQuery has become the standard query language for XML databases, and has received the support of most of the XML database systems. However, using XQuery, users need to understand the mode of the XML document and need to grasp the complexity of the query language syntax directly using XQuery XML database query is very difficult for non-professional users (Jiang, and Yang, 2011; Zhongyi, 2009; Jijun, and Shan, 2005; Jingqiang , et.al, 2004; Xiaomin and Yanlin, 2005). There are two basic types of user queries (XML is semi-structured data), keyword-based queries (Full-text search), and structural queries (complex queries specified in tree-like structure) as depicted in Figure 2.13.

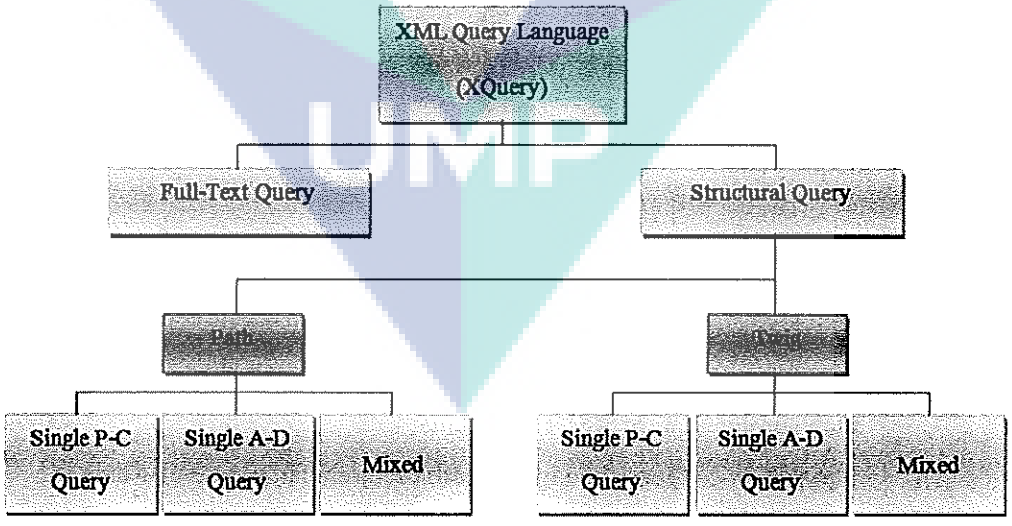


Figure 2.13 Main classification of *query*.
Source: Haw and Lee (2011). Data storage practices and query processing in XML databases:

A keyword search is somehow similar to content retrieval in information retrieval technology, where is a type of search looks for matching documents that contain one or more words specified by the user to find specific passages of text where query keywords co-occur. Conversely, a structural search is to retrieve matches on the tree where it has the tags and structure (relationship) specified in the query criteria. Structural query Structural queries can be classified further into path query and twig query and processing refers to finding all occurrences of a given set of structural relationships, such as: Parent–Child (P-C), Ancestor–Descendant (A-D), or mixed types of both relationships, in an XML database (Haw and Lee, 2011; Xin, et.al, 2010).

2.8.2.1 Keyword Search

The weaknesses of traditional document-oriented search techniques in the traditional databases, which rely heavily on heuristics that are intuitively appealing, that often, lead to retrieve false positive answers that overlook correct answers. In addition to their inability to properly arrange the answers led to the search for high precision query search technique for XML data. The use of XML language interfaces and keyword search techniques in Native XML databases which take advantage of XML structure, make it very easy to query XML databases. The independent design of Native XML databases and its use of the method for XML keyword queries that is based on an extension of the concepts of data dependencies and mutual information solved the problems for data-centric XML.

Keyword search is a practice used by search engine optimization professionals to find and research actual search terms people enter into the search engines when conducting a search. Search engine optimization professionals research keywords in order to achieve better rankings in their desired keywords.

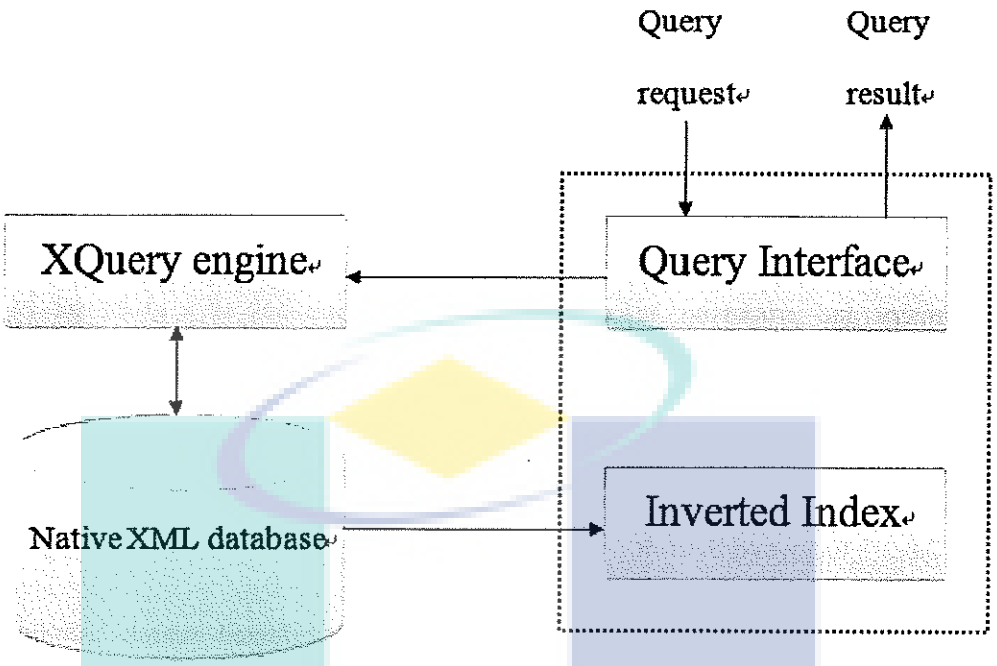


Figure 2.14 Keyword query for *Native XML Database*

Keyword search is considered to be an effective information discovery method for structured and semi-structured data. It allows users without prior knowledge of schema and query languages to search. In XML keyword search, the results of a keyword query are no longer entire XML documents, but instead are XML elements that contain all the keywords.

Keyword search in the Native XML database needs to consider two main things:

(i) How to define the index structure, index construction, and provide a basis for a query based on keywords.

(ii) How to use the index to the query.

Consider the keyword query (XML, data) over the XML document of Figure 1. Nodes 1.1.2.2.1 and 1.1.2.3.2 contain the two keywords, and node 1.1.2 is their lowest common ancestor. So the sub-tree rooted at 1.1.2 contains all the keywords and is expected to be the result.

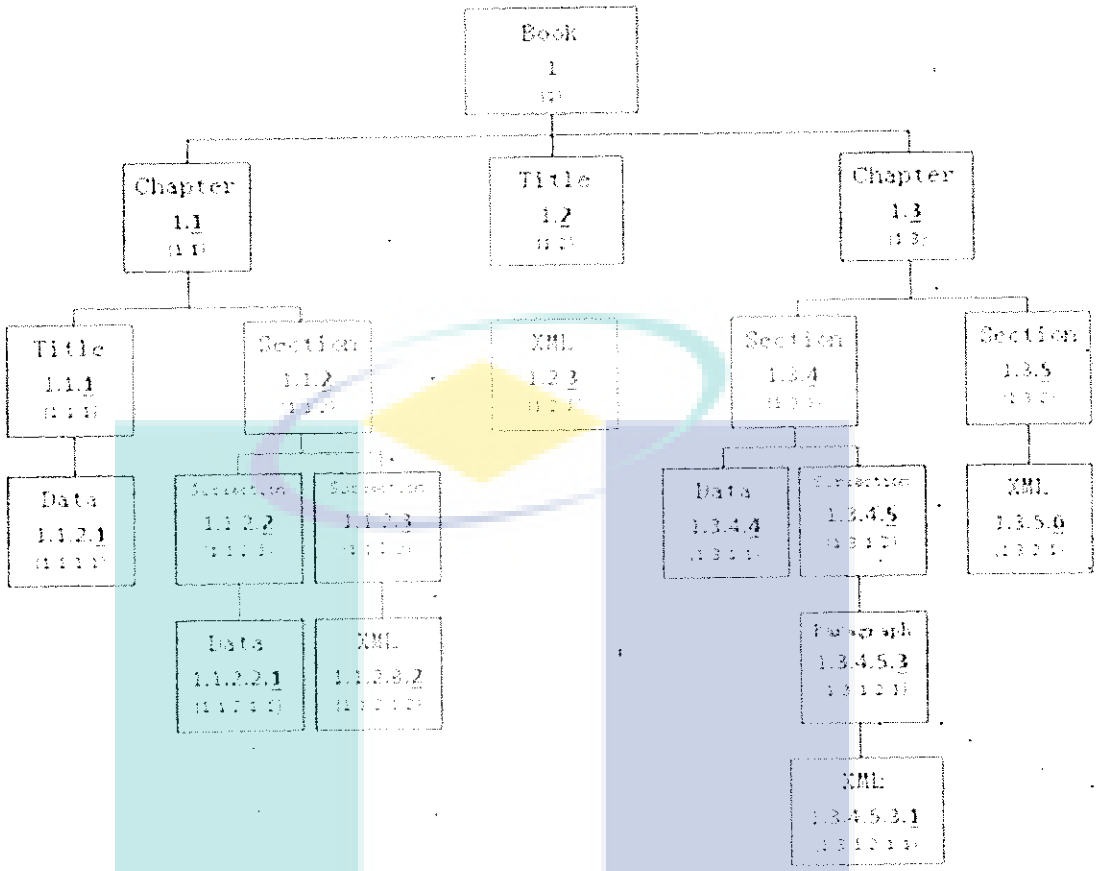


Figure 2.15 *Book.xml* (each node is associated with its Dewey number)

The proposed keyword search returns the set of the smallest trees containing all keywords, where a tree is designated as "smallest" if it contains no tree that also contains all keywords. For example: assume an XML document named "Book.xml", modelled using the conventional labeled tree model in Figure 2.15, that contains information, including Chapter, Title, etc. A user interested in finding the relationships between "XML" and "Data" issues a keyword search "XML, Data" and the search system returns the most specific relevant answers - the sub trees rooted at nodes 1.1.1, 1.1.2 and 0.2.0.0. The meaning of the answers is obvious to the user: "Ben" is a TA for "John" for the CS2A class, "Ben" is a student in the CS3A class taught by "John", both "John" and "Ben" are participants in a project.

2.8.2.2 Structural Search

Structural query processing refers to finding all occurrences of a given set of structural relationships (such as parent–child (P-C) - denoted by "/" - and ancestor–descendant (A-D) - denoted by "//" - relationships), by retrieving collections of information from the Native XML database based on a specification of structures (Xian, 2010; Wang, 1996). The structural relationships are a core component for XPath, XQuery and tree pattern queries. For example:

```
//Chapter//Section [figure AND table]
```

The query in the previous example retrieves all sections of chapters that contain at least one figure and at least one table. XML documents consist of nested elements enclosed by user defined tags, which indicate the meaning of the content contained. There are two types of structural query (As XML a semi-structured data): (1) only one node query that is namely path query, and (2) twig query that consist two or more nodes (Figure 2.16). These queries could be either simple as search and retrieve by using only a simple path, or complex as search and retrieve a small tree-like structure that involves many branches and joins.

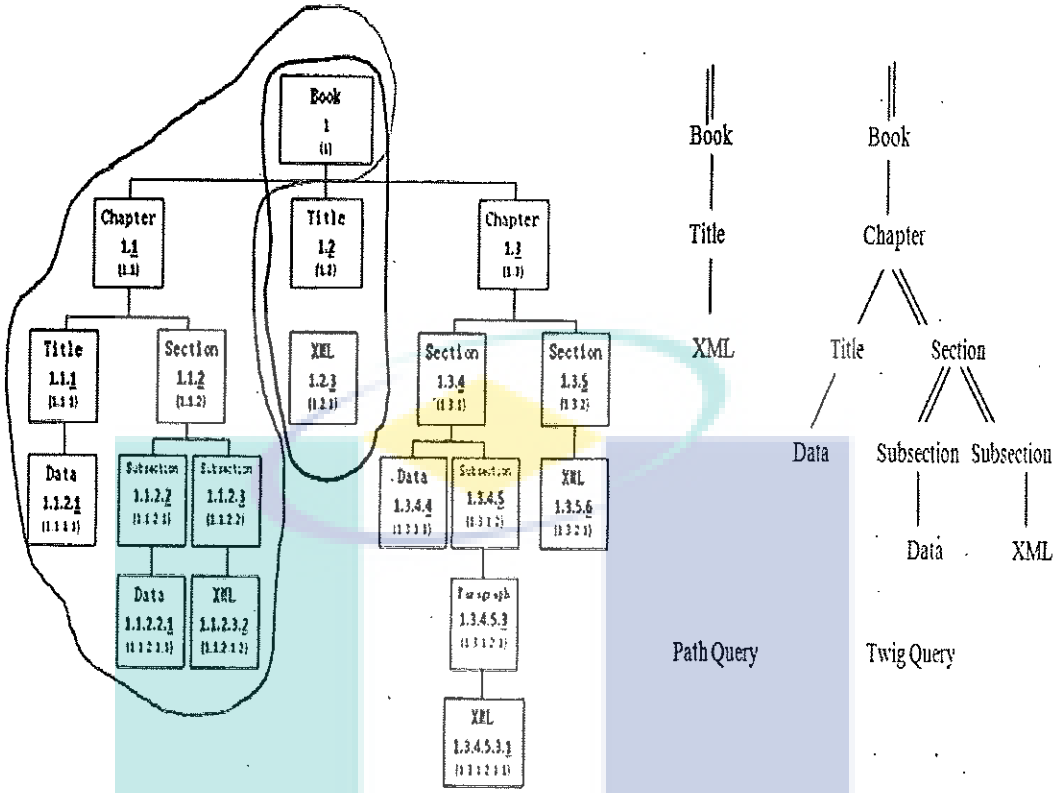


Figure 2.16 Example of an *XML tree*, a path query and a twig query

In an NXD, there are two main approaches to processing such queries, namely:

(i) Traversing the XML database sequentially to find the matching pattern.

(ii) Query processing using the decomposition–matching–merging approach.

For example, consider the following sample complex query in XPath (W3C, 2004) notation, where Query1:

```
/book[/title]/chapter/section/figure/caption.
```

2.8.3 Comparison between SQL vs. XQuery

XQuery is similar to SQL in many ways, but just as SQL is designed for querying structured, relational data, XQuery is designed especially for querying semi-structured, XML data from a variety of data sources. XQuery allows you to work in the

XML world no matter what type of data you're working with - relational, XML or object data (Robie, 2003)

The way between SQL and XQuery to do the tasks is quite different. SQL operates on the borderline between SQL and XML, while XQuery lives in a purely XML world. XQuery has important advantages even with the queries that span relational and XML sources.

XQuery expressions are similar to SQL as show in the following table:

Table 2.7 XQuery & SQL expressions

XQuery	**SQL**
For/let	select/set
Where	where
order by	order by
Return	return

The difference between SQL and XQuery shown in the next table:

SQL is the most mature query language for relational databases. Each database vendor has a slightly or even thoroughly different implementation of SQL, SQL focuses on sets of -flat- rows, and needs to specify a value for each field in a record-set (even null in case of an empty field). SQL supported by Oracle and IBM, but not by Microsoft.

XQuery is strictly a query language and can be seen as a much more generalized language for handling data, and it makes the databases truly transparent and independent from its vendor. Because of, XQuery focuses on sets of 'nested' nodes with an irregular depth. It is supported by Most of the major database vendors.

Table 2.8 XQuery & SQL

XQuery	SQL
XQuery is an expression oriented language	SQL is a statement oriented language.
Concept of nodes is used in XQuery	Concept of rows is used in SQL
A user cannot create and update tables.	A user can create and update tables.
It is a new language.	It is an old language.

2.9 Path Evaluation

After reviewing how XML can be stored natively, the next question to answer is how queries are evaluated. One important part in XML query evaluation is the navigational approach. As suggested by Zhang, et al (Zhang, et al, 2009), there are two basic types of navigational approaches: query-driven and data-driven.

In a query-driven navigational approach, the navigational operator translates each location step in the path expression into a transition from one set of XML tree nodes to anther set (Zhang, et al, 2009). Think of an ordered tree structure; a path step from a parent node to its child actually indicates a transition from the parent sub-tree nodes to the child sub-tree nodes. Native storage adopts this type of approach (Fiebig, et al).

On the other hand, in data-driven navigational approaches like Yfilter (Diao, et al, 2003) and XNav (Josifovski, et.al, 2005), the query is translated into an automaton and the data tree is traversed according to the current state of the automaton. While this data-driven approach can be more complex to implement, it needs only one scan of the input data (Zhang, et al, 2009). Both DB2 and Oracle use this type of approach. DB2's navigational operator is similar to XNav *20+, whereas Oracle's operator is similar to YFilter *19+.

Figure 2.16 give a basic idea about how Oracle evaluates a path expression. Path queries are similar to regular expressions. Regular expressions allow a user to do pattern

matches. For example, if a string matches the pattern "a*bc", the string must contain the character "a" before character "b" and "c", with any characters allowed in between the "a" and "b". A path expression is much like a string pattern.

Take for example a document that matches the path expression /a/b/c: first, it must have an element "a" as the root node, and then "a" must have a child "b" and "b" must have a child "c".

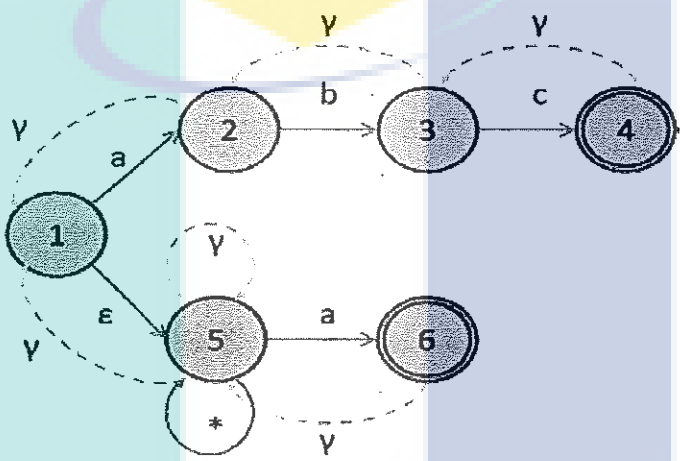


Figure 2.17 NFA Constructed from *Paths* //a and /a/b/c

Since regular expressions can be evaluated by a finite state machine, Oracle builds a non-deterministic finite automaton (NFA) to process a given set of path expressions. For example, Figure 2.17 shows an NFA constructed from the path expressions //a and /a/b/c. This is an example taken from Zhang et al (Zhang, et al, 2009). When processing a path expression using an NFA, it is possible to combine two or more paths together in a single NFA, as is shown in Figure 2.17. Therefore, it is possible to save some physical disk I/Os by evaluating multiple path expressions from a given XML query in a single pass.

To apply a set of paths, the NFA reads in the XML document a node at a time. After each node is read, the NFA either make a transition to another state or stays put. This is done node by node until the NFA reaches the final state (state 4 or 6 in Figure 2.17), at which point a match was found. Figure 2.18 diagrams the overall query

evaluation methodology (Zhang, et al, 2009). There is a decoder module that decodes the input XML stream and feeds the NFA every event that it reads. Each event read is similar to an XML SAX event, and indicates things like the beginning of the document, the beginning of an element and so on, which decides the state of the NFA. (Many optimizations were made for the input stream decoders in Zhang et al (Zhang, et al, 2009).

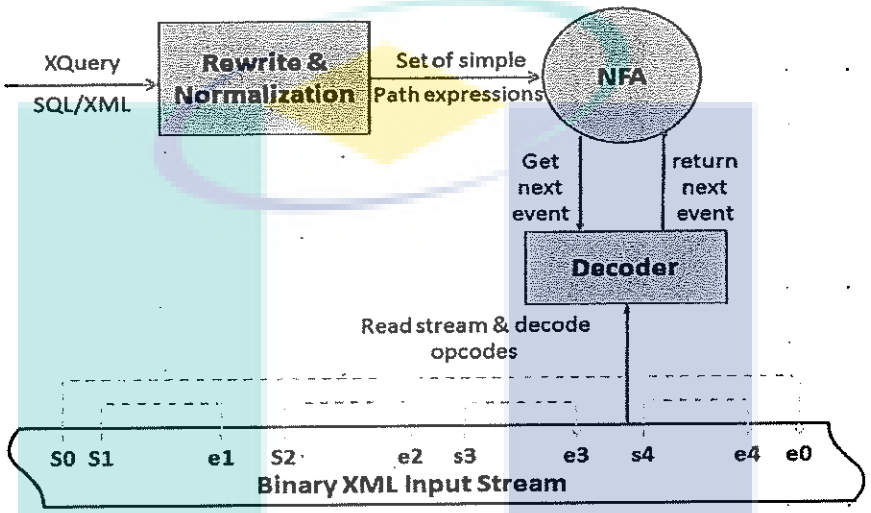


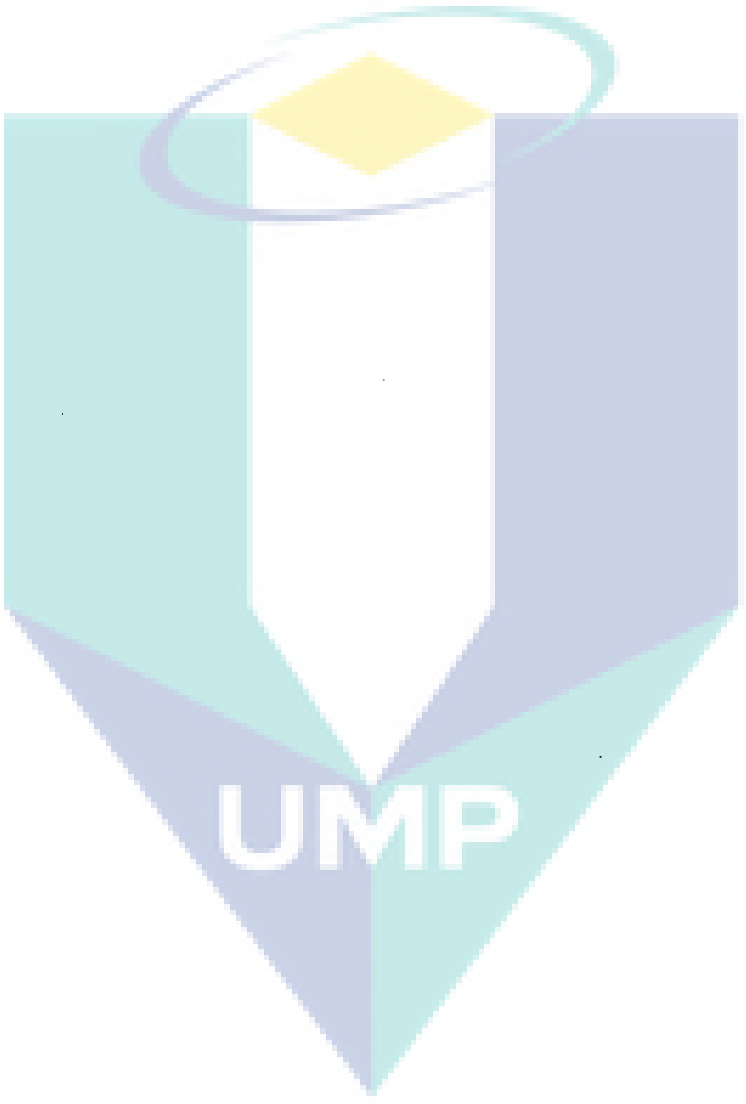
Figure 2.18 Oracle Binary *XML* Streaming Evaluation Architecture

2.10 Conclusion

This chapter presented two databases: Native XML Database (NXD) and XML_Enabled Database (XED). And the methods for both databases of dealing with XML data (Store, Extract and Search). This chapter also presented some of the previous studies since 1999 to 2014 (Presented in 2.2 BACKGROUND) , and the studies show that, relational database facing difficulties to dealing with the XML data, because of the differences in the structures which leads to shredding process (Presented in 2.6.1 XML_Enabled Databases).

In this chapter of the literature review, we conclude that, XML documents do not require the existence of DTDs causes the DTD-dependent mapping methods to be inapplicable, and the generic mapping often exhibits poor performance in query processing, especially when "shredded" documents must be recomposed in answer to a query (Discussed in 2.2). Other mapping methods may have some advantages for

specific workloads, but performance suffers if they are applied to a wide, unknown range of XML environments. Most unfortunately, in all of these approaches, once a mapping is defined, it cannot be changed without a major data reorganization and corresponding application reimplementation (Discussed in 2.6).



CHAPTER 3

METHODOLOGY

XML DATABASE SYSTEM BENCHMARK

3.1 Introduction

This chapter presented and analyzed the performance of both databases (XML_Enabled databases and Native XML databases) by using the database benchmark. A series of experiments have been conducted by using one of Native commercial XML database system, and a leading commercial XML_Enabled database system. The aim of the experiment is to evaluate the efficiency of path expressions in the databases, to compare the insert, update, delete and search performance of both databases. In addition, this chapter provides the details on the concrete implementation of the storage and indexing components by using XML database system benchmark. Finally, the benchmark query engine that is used in the data sets collected is discussed.

The methodology of the study applied the method that has been developed by Zhang and Tompa in 2004. This study is to evaluate: insert, update, delete and search performance in both databases XML_Enabled database and Native XML database. For XML-Enabled database mapping, this chapter applied the method that has been developed by K. Williams with his research teams his research teams in 2001 (Williams et al. 2001) that converts a relational schema into a DTD (Figure 3.3). Then use this DTD to generate a XML schema and store it in a database. A fixed number of records in text format are assigned to be imported into a database. The data sets measurements are repeats from 65.8, 101, 117, 127, 183 MB of books records.

3.2 Setting and Setup

3.2.1 Experimental Setting

All experiments were run on the following setting:

Table 3.1 Experimental setting

Item	Describation
CPU	Intel® Core™ i7-3770 CPU @ 3.40GHz 3.40 GHz
OS	Windows 7 Professional
RAM	4.00 GB
System Type	32-bit operating system
Hard Disk	931.51 GB
Java	JDK 1.6.0

3.2.2 System Setup and Memory Setting

For both database systems, the default settings are used during the software installation. Java -Xmx is an effective way to determine the maximum size of the memory as which is the configuration parameter to control the amount of memory. Thus, Native XML database automatically will not use all of the memory available on the machine. For XML_Enabled database, after upload the documents, it needs to update all the databases to provide the most current information for the database.

For Native XML database, if you launch the database via one of the shell or batch scripts, you need to change -Xmx in there. If you launch database as a service, all Java settings will be controlled by the service wrapper.

```
# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=128
```

In Native XML database, the nodes buffer attribute has been used to set the database's temporary internal buffer to a fixed size. The buffer is used during indexing to cache nodes before they are flushed to disk. Because of each of the core database files and indexes has a page cache. The main purpose of this cache is to make sure that the most frequently used pages of the DB files are kept in memory. If a file cache becomes too small, the database may start to unload pages just to reload them a few moments later. This "trashing effect" results in an immediate performance drop, in particular while indexing documents.

3.2.3 Data Sets and Characteristics

This section discuss of the characteristics of XML data sets. The structure of a relevant data set must be complex enough to capture all characteristics of XML data representation, which can have a significant impact on the performance of query operations. The scalability of a system can be measured by using data sets of different sizes. Since XML data can be represented as a tree, the depth and width of the tree should be adjustable. This can be achieved as follows:

Table 3.2 Characteristics of the data set

Factor	Size (MB)	Nodes
0.01	65.8	117 132
0.1	101	567 865
0.3	117	801 498
0.5	179	1 539 911
1	183	1 995 315

There are various ways of indexing XML data, and one of the most effective approaches is to index the total number of root-to-leaf paths in the input file.

3.2.3.1 Data Depth and Fan-out

The depth and fan-out are important to tree-structured data. The depth of a node is the length of the path of its root (i.e., its root path), and it can have a significant performance impact, for example when evaluating indirect containment relationships between ancestor and descendant nodes in the tree. Fan-out of the nodes refers to this number of pointers per node, and it can impact the way in which the database system stores the data and responding to queries that are based on selecting children in a specific order (for example, selecting the last child of a node).

It has been created a base benchmark data set of a depth of 6. Then, using a "level" attributes. This can restrict the scope of the query to data sets of a certain depth, thereby, quantifying the impact of the depth of the data tree. Similarly, we specify high (7) and low (2) fan-outs at different levels of the tree as shown in Table 3.3.

Table 3.3 The Nodes in the Base Data Set

Level	Fan-out	Nodes	% of Nodes
1	2	1	0.0
2	2	2	0.0
3	2	4	0.0
4	2	8	0.0
5	4	16	0.0
6	7	208	0.0

3.2.3.2 Data Set Granularity

It was chosen as a single document tree as the default data set to keep the benchmark simple. If it is important to understand the effect of document granularity, one can modify the benchmark data set to treat each node at a given level as the root of a distinct document. One can compare the performance of queries on this modified data set against those on the original data set.

3.3 Methods

The methodology of the study applied the method that has been developed by Zhang and Tompa in 2004. For XML-Enabled database mapping, applied the method that has been developed by K. Williams with his research teams his research teams in 2001 (Williams et al. 2001) that converts a relational schema into a DTD that shown in (Figure 3.3) as follows:

Step 1	Database Design
Step 2	The methodology of this study is based on XMark and XMach-1 (3.4.1 and 3.4.2). In the methodology, a number of performances have been measured: (i) Document generation by queries (ii) Measure the size of the databases (XED and NXD) to evaluate how efficient the database server stores XML documents.
Step 3	Explore the functionality of the database (XED and NXD) in handling XML documents. (i) Store XML documents. (ii) Extract XML documents.
Step 4	Evaluate the performance of the data processing operations to data of different sizes, start from the smallest to the largest (65.8, 101, 117, 127, 183 MB). In comparison between Native XML database and XML_Enabled database. (i) Insert single record of XML document. (ii) Insert mass records of XML document. (iii) Delete single record of XML document. (iv) Delete mass records of XML document. (v) Update single record in the table of XML document. (vi) Update mass record of XML document. (vii) Search XML document.

The methodology for measuring the performance of both databases being tested XML_Enabled database and Native XML database are similar, and the added data is the same in both databases. For XML_Enabled database, has been used DTD in Figure 3.3 that converted from the relational schema Figure 3.1.

For inserting, deleting and updating single record of XML document, it has identified by the position of elements in the document. The evaluation of each operation was by using a number of reports (21 reports. Table 3.6) and queries (20 queries Table 3.4 and Table 3.5) for each operation.

The size of databases and the added data are measured in megabytes. A fixed number of XML documents are assigned to be imported into the both databases (XED and NXD). The documents are repeated from the smallest to the largest (65.8, 101, 117, 127, 183 MB). Each measurement is repeated three times, and the average value is taken. Examine the efficiency and the time for restore and rebuild the XML document from both database (XED and NXD) in seconds.

Java DOM APIs were used to directly access the Native XML database server. XQuery was used instead of SQL for reporting, because XQuery was required to access the server using the HTTP protocol

To compare the different data sets size inside the databases needs to run a set of tests that do the following:

(i) Store XML documents
(ii) Extract complete XML documents
(iii) Delete complete XML documents
(iv) Extract parts of documents identified by the position of elements in the document
(v) Replace parts of documents

3.3.1 Database Design

Firstly, it has been designed a database that uses XML features as much as possible (Complex XML data). This chapter implemented a system to model (Amazon book sales and reports). Figure 3.1 contains the core relational schema and figure 3.2 is Entity Relationship (EER) diagram of the relational schema that used in this chapter.

```
Relation Client (Client-no, Client-name, Sex, Email, Telephone
Postalcode)
Relation Client-add (Client-no, Address, City, State, Country,
Is_default)
Relation Bill (Bill-no, Client-no, Quantity, Bill_amount, Bill_date,
Shipment-type, Shipment_date)
Relation Bill-substance (Bill-no, Substance _no, quantity, Price,
Bill-price, Discount)
Relation Substance (Substance _no, Substance _name, Catalog_type,
Author, Publisher, Substance _price)
Relation Category (Catalog_type, Catelog_description)
Relation Shipment (Shipment-type, Shipment-description)
Relation Monthly_sales (Year, Month, Quantity, Total)
Relation Client_sales (Year, Month, Client-no, Quantity, Total)
Relation Substance _sales (Year, Month, Substance _no, Quantity,
Total)
```

Figure 3.1 Relational Schema of *Sale and Report*

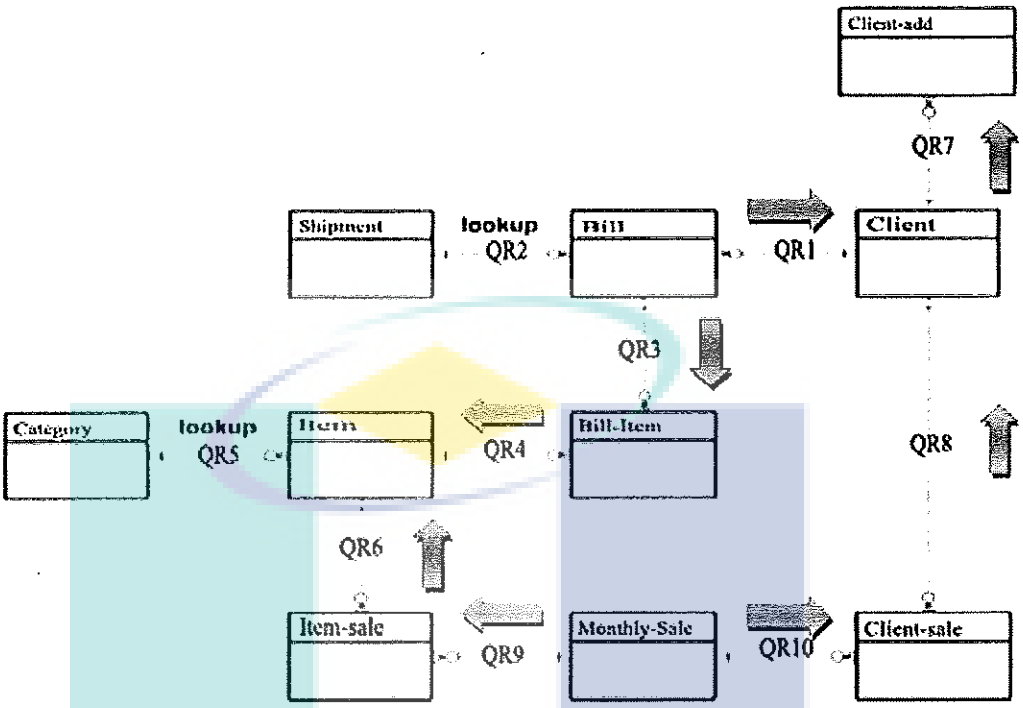


Figure 3.2 *Enhanced Entity Relationship (*EER) diagram of the relational schema

The next step was to translate the relational schema into an XML schema. It is adopted the methodology used by K. Williams (Williams et al. 2001). Figure 3.3 contains the final DTD produced from this process.

```
<!ELEMENT Sales (Bill*, Client*, Substance *,Monthly-sales*)>
<!ATTLIST Sales
Status (New|Updated|History) #required>
<!ELEMENT Bill (Bill-substance *)>
<!ATTLIST Bill
Quantity CDATA #REQUIRED
Bill_amount CDATA #REQUIRED
Bill_date CDATA #REQUIRED
Shipment-type (Post|DHL|UPS|FedEx|Ship) #IMPLIED
Shipment_date CDATA #IMPLIED
Client_idref IDREF #REQUIRED>
<!ELEMENT Client (Client-add*)>
<!ATTLIST Client
```

```
Client_id ID #REQUIRED
Client-name CDATA #REQUIRED
Sex CDATA #IMPLIED
Telephone CDATA #IMPLIED
Email CDATA #IMPLIED>
Postalcode CDATA #IMPLIED
<!ELEMENT Client-add EMPTY>
<!ATTLIST Client-add
Address NMTOKENS #REQUIRED
City CDATA #IMPLIED
State CDATA #IMPLIED
Country CDATA #IMPLIED
Is_default (Y|N) "Y">
<!ELEMENT Bill-substance  EMPTY>
<!ATTLIST Bill-substance
Quantity CDATA #REQUIRED
Price CDATA #REQUIRED
Bill-price CDATA #REQUIRED
Discount CDATA #REQUIRED
Substance _idref IDREF #REQUIRED>
<!ELEMENT Substance  EMPTY>
<!ATTLIST Substance
Substance _id ID #REQUIRED
Substance _name CDATA #REQUIRED
Category_type (Art|Comp|Fict|Food|Sci|Sport|Trav) #REQUIRED
Author CDATA #IMPLIED
Publisher CDATA #IMPLIED
Substance _price CDATA #REQUIRED>
<!ELEMENT Monthly_sales (Substance _sales*, Client_sales*)>
<!ATTLIST Monthly_sales
Year CDATA #REQUIRED
Month CDATA #REQUIRED
Quantity CDATA #REQUIRED
Total CDATA #REQUIRED>
<!ELEMENT Substance _sales EMPTY>
<!ATTLIST Substance _sales
Quantity CDATA #REQUIRED
```

```
Total CDATA #REQUIRED
Substance _idref IDREF #REQUIRED>
<!ELEMENT Client_sales EMPTY>
<!ATTLIST Client_sales
Quantity CDATA #REQUIRED
Total CDATA #REQUIRED
Client_idref IDREF #REQUIRED>
```

Figure 3.3 *DTD Corresponding* to the Relational Schema

Source: Williams (2000). Professional XML Databases

3.4 XML Database System Benchmark

Benchmark is a popular tools which test Relational XML database and Native database. In gauging performance of NXDs, like other technology systems there are benchmarking utilities available to evaluate performance of typical processing scenarios. The benchmarked data set must represent and incorporate data characteristics that are likely to have an impact on the performance of query operations. Although, it also must be easy to understand, and expose the component of the system that is a bottleneck. In general, benchmark testing mechanism should be: relevant, portable, scalable, and simple. Indeed the two mechanisms we considered for our evaluation and research efforts detailed later in this document, XMark and XBench, certainly meet the criteria. XMark and XMark-1 are two main tools of Benchmarks in the research field. XMark and XMark-1 have been used for the data set.

There are a number of existing benchmarks available to measure XQuery support and XML database processing performance, including XMach-1 (Böhme & Rahm, 2001), XMark (Schmidt & Wass, 2010) and XBench (Schroeder & Hara, 2014). These benchmarks are predominantly application-oriented. Usually the testing performance is measured by using a serial of data factors, which has varying sizes、multi-feature documents; the performance of a system can be measured by using data sets of varying sizes, different documents with different features as shown in the Table 3.2. The size of XML documents and the number of elements have been determined by the factor of the main driver of generation.

3.4.1 XMark

XMark is a reported tool in the benchmark for XML data management and it's designed to generate XML documents. The XMark framework and benchmark suite allows users and developers to gain insights into the characteristics of their XML repositories. It provides a mechanism that assesses the abilities of XML databases to handle different query types typically encountered in real-world scenarios. The suite provides for a set of queries where each query is intended to challenge a particular aspect of the DBMS query processor. The benchmark suite uses XMLgen, a document generator that generates scalable XML documents for platform independent implementations. The generation and engine and instructions are freely available for researchers and evaluators. The framework offers observances of referential constraints concerning ID/IDREF pairs and assures reproducibility across. We considered the XMark benchmark not only for the integrity of its framework, but for its simplicity of use and acceptance within the XML industry. Although it comes with a rather complex DTD, which defines a set of trees with varied structures, for the indexing and evaluating which do not use the schema (Schmidt & Wass, 2010). The scalability of a system can be measured by using data sets of varying sizes, different documents with different features as shown in the Table 3.4.

Xmlgen is an XML data generator for XMark, which produces xml documents. Xmlgen is a part of the benchmark suite and provides a template to set the number and type of elements, and makes probability distributions by setting the parameters; Xmlgen also provides DTD and schema. Xmlgen was developed in ANSIC and can be used on windows and has been used on a number of platforms, including Windows, Solaris, various Linux distributions, and IRIX.

Xmlgen was designed to produce large and very large XML documents in an efficient manner with low constant main memory requirements. The current version of Xmlgen require less that 2MB of main memory. The normal XML document size 100MB with a scaling factor 1.0, and users can create lager documents b changing the scaling factor.

Likewise, XMark provides 20 queries that cover most of XQuery's functionality. These queries mainly contain simple relational queries, order preserving,

navigational queries, aggregate, references and storing operations. All of the benchmark queries are formulated in XQuery.

Table 3.4 Queries Specified in the XMark Benchmark

Group	Query	Function	Comment
1	Query 1	Return the name of the client with ID 'customer7'. The client registered in east Chine.	Check the ability for handling strings with a fully specified path.
	Query 5	Return how many substance s that sold and cost more than 67.	Check the performs of DBMS when XML model is a document oriented.
	Query 14	Return all the substance s that Contain the word 'book'.	Text search but narrowed by combining the query on content and structure.
2	Query 2	Return to the initial increases of all open auctions.	Evaluate cost of array lookups. The authors cite that a relational backend may have problems determining the first element. Essentially query is about order of data which relational systems lack.
	Query 3	Return IDs of all open auctions whose current increase is at least twice as high as initially.	More complex evaluation of array lookup.
	Query 4	List reserves of those open auctions where a certain person issued bid before another person	Querying tag values capturing document orientation of XML.
	Query 11	Check and list the number of substance is currently on sale whose price does not exceed 0.07 % of each person's income.	Value-based joins. The authors feel this query is a candidate for optimizations.

Table 3.4 Continued

Group	Query	Function	Comment
2	Query12	For each richer than-average person list the number of substance is currently on sale whose price does not exceed 0.02% of the person's income.	Value-based joins. The authors feel this query is a candidate for optimizations.
	Query17	Which persons don't have a home page?	Determine processing quality in presence of option AL parameters.
3	Query6	How many substance s are listed on all continents?	Test efficiency in handling path expressions
	Query7	How many pieces of XML books are in the database?	The query is the answerable using cardinality of relations. Testing implementation.

3.4.2 XMark-1

By comparison, the Xmark-1 is another multi-user tool of benchmark that was developed at the University of Leipzig in 2000 (Franceschet, 2005). The system architecture includes the following components: an XML database, application server, and the interfacing software which can process XML documents and communicate with the database. XMark-1 was developed in Java and the current version has been implemented in some XML database systems.

```
<!ELEMENT   document (title, chapter+)>
<!ATTLIST   document author CDATA #IMPLIED
            doc_id ID   #IMPLIED>
<!ELEMENT   author (#PCDATA)>
<!ELEMENT   title (#PCDATA)>
<!ELEMENT   chapter (author?, head, section+)>
<!ATTLIST   chapter id ID #REQUIRED>
<!ELEMENT   section (head, paragraph+, subsection*)>
<!ATTLIST   section id ID #REQUIRED>
<!ELEMENT   subsection (head, paragraph+, subsection*)>
<!ATTLIST   subsection id ID #REQUIRED>
<!ELEMENT   head (#PCDATA)>
<!ELEMENT   paragraph (#PCDATA | link)*>
<!ELEMENT   link EMPTY>
<!ATTLIST   link xlink:type (simple) #FIXED "simple"
xlink:href  CDATA #REQUIRED>
```

Figure 3.4 DTD of *XMach-1 Database*

The XMark-1 has a document generator which can synthetically generate XML documents. The system consists of two parts: XML files and data-centric directory. The text contents are chosen from 10,000 of the most frequently used words in English. Each of the XML files simulates as article, which includes title, chapter, section, paragraph, and so on. The directory system is based on XML files, and mainly contains metadata of the other documents.

The XMark-1 supports both schema-based and schema-less variants of the benchmark. The file sizes vary from 2 KB to 100 MB, and the structures vary flat to deep hierarchy. XMark-1 controls the size of the whole database by changing the number of XML files. There are four database sizes in XMark-1, which are created by varying the numbers in the files: 65.8, 101, 117, 127, and 183. XMark-1 utilizes an algorithm to generate documents with different depth and size. The process of generating documents can be controlled through setting parameters. XMark-1 is also developed with Queries, it is specified in the Table 3.5:

Table 3.5 Queries Specified in the XMark-1 Benchmark

Group	Query	Function	Comment
1	Query1	Get document with the given URL.	Return a complete document (complex hierarchy with the original ordering preserved).
	Query2	Get doc_id from documents containing a given phrase.	Text retrieval query. The phrase is chosen from the phrase list.
	Query5	Get doc_id and id of the parent element of author element with given content.	Find chapters of a given author. Query across all DTDs/text documents.
	Query3	Return leaf in tree structure of a document given by doc_id following the first child in each node starting with document root.	Simulates is exploring a document with unknown structure (path traversal).
2	Query4	Get the document name (last path element in the directory structure) from all documents that are below a given URL fragment.	Browse directory structure. Operate on structured unordered data.
	Query6	Get doc_id and insert date from documents having a given author (document attribute).	Join operation.
3	Query7	Get doc_id from documents that are referenced by at least four other documents.	Get important documents. Needs some kind of group by and count operation.
	Query8	Get doc_id from the last 100 inserted documents having an Author attribute	Needs count, sort, and join operations and accesses metadata.

3.4.3 XBench Benchmark

XBench is a family of benchmarks that capture different XML application characteristics (Yao et al., 2004). Like XMark, these applications are categorized as data-centric or text-centric (document-centric) and the corresponding XML can consist of single documents or multiple documents. Data-centric (DC) application examples include ecommerce catalog data or transactional data. The text-centric (TC) applications examples include book collections in a digital library, or news article archives. Support for both is in the form of single document (SD) or multiple (MD) documents which can be generated with the tool. As part of this benchmark suite, the database generator that can build four cases: DC/SD, DC/MD, TC/SD, and TC/MD. The XBench database generator can generate databases in any of these classes ranging from 10MB to 10GB in size. A good source of reference regarding XBench testing is provided in (Yao et al., 2004) XBench research paper, where they benchmarked DBMS systems that store, retrieve, and update XML documents. For tackling the fragmentation problem for transactional databases which are highly distributed, in (Schroeder & Hara, 2014), it applied the fragmentation approaches on the XML schema and workload provided by the XBench benchmark. From each approach, the XBench dataset was fragmented and stored in a cloud data store. It executed XBench using six different cluster sizes of eight Amazon EC2 nodes allocated in a single region. In (Thomas et al., 2014), it applied the XBench benchmark which has been proposed for applications categorized as text-centric and that involve multiple documents. The benchmark provides an XML Schema that models part of the Springer digital library, a workload with 19 queries and a database generator that can be configured to output datasets containing a user defined number of articles. However, one of the main shortcomings of XBench is that the System Under Test (SUT) Description works under single user mode. No concurrent access to the system is supported. The multiple users work mode is supposed to be implemented in the future. Another weakness of XBench is that only query workloads are supported in the first version, but the designer plans to include the update and bulk-loading workloads (Yao et al., 2004).

3.4.4 Performance Consideration

When we consider native XML databases as a means of storing XML data, one of the major considerations must be the speed of retrieval and manipulation of store data. Depending on the stored document structure (well type data-centric vs. document-centric), how the native XML database physically stores data (file system or other DOM like method), and the application of indexing schemes as identified in Chapter 3, it is possible under many application that one can retrieve data faster than other storage alternatives like XML-enabled alternatives. For example, if retrieving lots of data in a singular XML document stored in an NXD (perhaps in a single physical file),versus retrieving equivalent data from a highly normalized relational database (shredding) requiring traversing relational physical joins, and amassing other overhead, we may see faster response times from NXDs. In this case, the database can perform a single index lookup, and assuming that the necessary fragment is stored in contiguous bytes on the disk, it can retrieve the entire document or fragment in a single read. As we have seen above, with relational databases this may require multiple index lookups and multiple disk reads. However, RDBMS are much evolved, and offer such incredible performance that this overhead is sometimes obscured by these efficient processing engines.

From a performance standpoint, model-based native XML databases that use a proprietary storage formats are likely to have performance similar to text-based native XML databases when retrieving data in the order in which it is stored. This is because databases use physical pointers between nodes, which should provide performance similar to retrieving text. Which is faster also depends on the output format. Text-based systems are obviously faster at returning documents as text, while model-based systems like DOMs are obviously faster at returning documents as DOM trees, assuming their model maps easily to the DOM model.

Because these physical links are faster to navigate than logical links, native XML databases, like hierarchical databases, can retrieve data more quickly than relational databases. Because of this, they should scale well with respect to retrieving data. In fact, they should scale even better than relational databases in this respect, since scalability is related to a single, initial index lookup rather than the multiple lookups required by a relational database. Like hierarchical databases, the physical wiring in

native XML databases applies only to a particular hierarchy. That is, retrieving data in the hierarchy in which it is stored is very quick, but retrieving the same data in a different hierarchy is not. Because native XML databases make heavy use of indexes, often indexing all elements and attributes, it helps with performance traversing this tree-based hierarchy. Unfortunately, while it may help with query performance, it does increase update time, since mentioning such indexes can be expensive.

3.5 Benchmark Querises

The expressive strength of XML effects greatly on the performance of the implementations of query languages for XML. A list of "Must have" requirements for XML query languages have been published by W3C XML Query Language Working Group and have been shown in Table 3.6 and analyze the impact of these various requirements on the performance.

XML can represent structured and unordered data, and an XML query language must be as expressive as a structured query language such as SQL is for relational databases. Have noticed that benchmark query cannot evaluate the database systems that implement XPath, because XPath can only count the function not the average, and benchmark queries cannot impractical on such systems. This problem can solve easily by using benchmark queries, for each test a different kind of aggregation. In addition, in case of test only a subset of the functionalities covered by some queries. For example: sometimes there is no need to restructure the retrieved results while others never need to update the database. Thus, it is important to distribute the various functionalities into different queries so that users can always choose the queries according to the functionalities they need.

Separating the functionalities also facilitates the analysis of the experiment results since it will be very clear which feature is being tested. Finally, the benchmark queries should allow the range of values of selected attributes to be varied in order to control the percentage of data retrieved by queries—that is, the selectivity of queries.

Table 3.6 Desired Functionalities of XML Query Languages

ID	Description
QR1	To query all data types and collections of possibly multiple XML documents.
QR2	Allow data-oriented, document-oriented, and mixed queries.
QR3	Accept streaming data.
QR4	Support operations on various data models.
QR5	Allow conditions/constraints on text elements.
QR6	Support hierarchical and sequence queries.
QR7	Manipulate NULL values.
QR8	Support quantifiers (∃, and ~) in queries.
QR9	Allow queries that combine different parts of document(s).
QR10	Support for aggregation.
QR11	Able to generate sorted results.
QR12	Support composition of operations.
QR13	Allow navigation (reference traversals).
QR14	Able to use environmental information as part of queries (current date, time etc.)
QR15	Able to support XML updates if data model allows.
QR16	Support type coercion.
QR17	Preserve the structure of the documents.
QR18	Transform and create XML structures.
QR19	Support ID creation.
QR20	Structural recursion.
QR21	Element ordering.

For example: QR9, QR10, QR11, which represents join operations, aggregation and sorting, use data-centric capabilities or relational queries. QR17 uses document-centric capabilities to keep the documents structure in some form. QR13 and QR20 are requirements that need to traversal of XML document structure using references or links as supported by XLink/XPointer specification. While QR21 uses open element ordering the functionality, which is an important feature of XML representation that have a significant impact on the of query languages to represent.

Used XQuery by pre-indexing the structure of the data in the Native XML database. The Problem of the performance has been solved by use structural indexing to allow queries to implementation of optimal performance. For example:

```
//Book[ISBN = "1558746218"]
```

The database will retreat and search over all <ISBN> elements in the database. This will lead to a decline in performance because of slow implementation and limits conBOOK. To query more efficiently and to improve the performance, we define an index (in /db/system/config/db/collection.xconf) and add range indexes for the most frequently used comparisons.

3.6 Algorithm

The algorithm for this study enhanced the method that has been developed by Zhang and Tompa in 2004. The researcher's method used object-relational DBMS schema used to hold the XML document schemas and data, the user-defined types needed to make hierarchical operations efficient, and the algorithm used to map XPath expressions to their corresponding SQL-3 queries.

The figure shows Data Definition language that has been required. Each different sub-set of XML data will store in separate pairs of tables. Each of these tables is implemented to model aspect of the XML schema and data model specification. The Specification included the following types of XML document nodes:

(i) Document
(ii) Element
(iii) Attribute
(iv) Namespace
(v) Processing Instruction
(vi) Comment
(vii) Text

In addition, XML Data Model includes a set of 19 primitive atomic types. Each of these atomic type value in the document must be one of these types (Figure 2:12, mentioned in chapter 2).

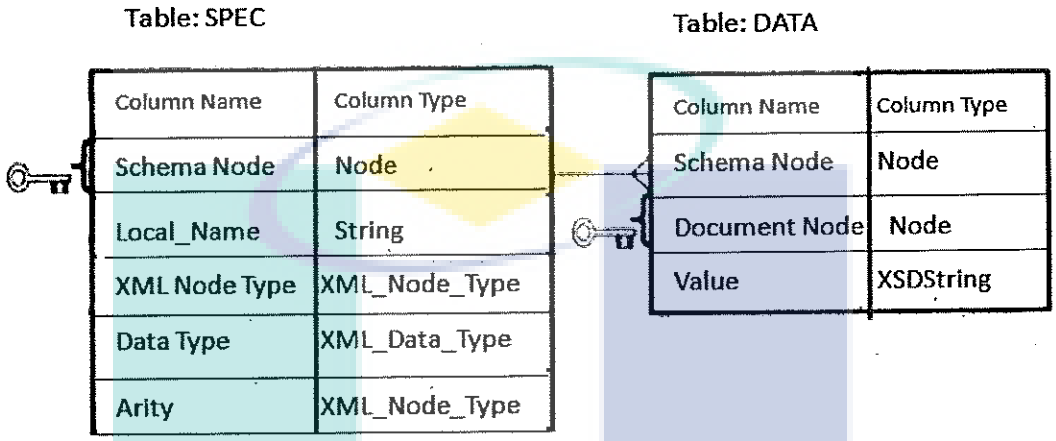


Figure 2:12 *Data Definition language*

The researcher's algorithm used only a single pair of tables. In practice, an XML repository would need considerably more. XML's use of namespaces to assign semantic intent to tag names, for example, suggests that some additional columns are necessary. And although XML's "wire" format consists of ASCII strings, care must be taken to ensure that XPath expressions with range predicates (>, <, etc.) and the more complex set-membership operations evaluate correctly.

While the algorithm in this study used indexing to all the element text, and attribute nodes, to fast XPath queries. All indexes are managed by the database engine. However, it is possible to restrict the automatic full-text indexing to defined parts of a document (shown in 2.8.2). Evaluating structured queries against possibly collections of unconstrained documents poses a major challenge to storage organization and query processing. To speed up query processing, some kind of index structure is needed. Thus, the algorithm used a numerical indexing scheme to identify XML nodes in the index. The indexing scheme not only links index entries to the actual DOM nodes in the XML store, but also provides quick identification of possible relationships between nodes in the document node tree, such as parent-child or ancestor-descendant relationships, while

conventional approaches are typically based on top-down or bottom-up traversals of the document tree.

The method in this study can store the whole document in a column of type text and don't require the documents to be chopped into small pieces to be squeezed into relational tables.

The method define an extraction operator, χP1, P2 (T). This operator takes a table T as input and two parameters, P1 and P2, where P1 is a column of table T of type text and P2 is a tree pattern to match against each text entry in the given column P1. The pattern matching language is a variant of XQuery that describes tree patterns instead of path patterns. Therefore, it differs from an XQuery expression by identifying several nodes in a tree that correspond to a single match rather than extracting only the last node in some path.

As a result, χP1, P2 (T) is computed by considering each row of T in turn, as shown in Figure 3.5 for the pattern corresponding to //book#/author# to extract book-author pairs simultaneously.

The tables are then "attached" to T as if by a join that correlates each row in T with all rows produced from the P1-value in that row. Hence the result of applying this operator is an expanded and untested table. The new column names by default are the same as the root names of the extracted texts, with suitable renaming as necessary.

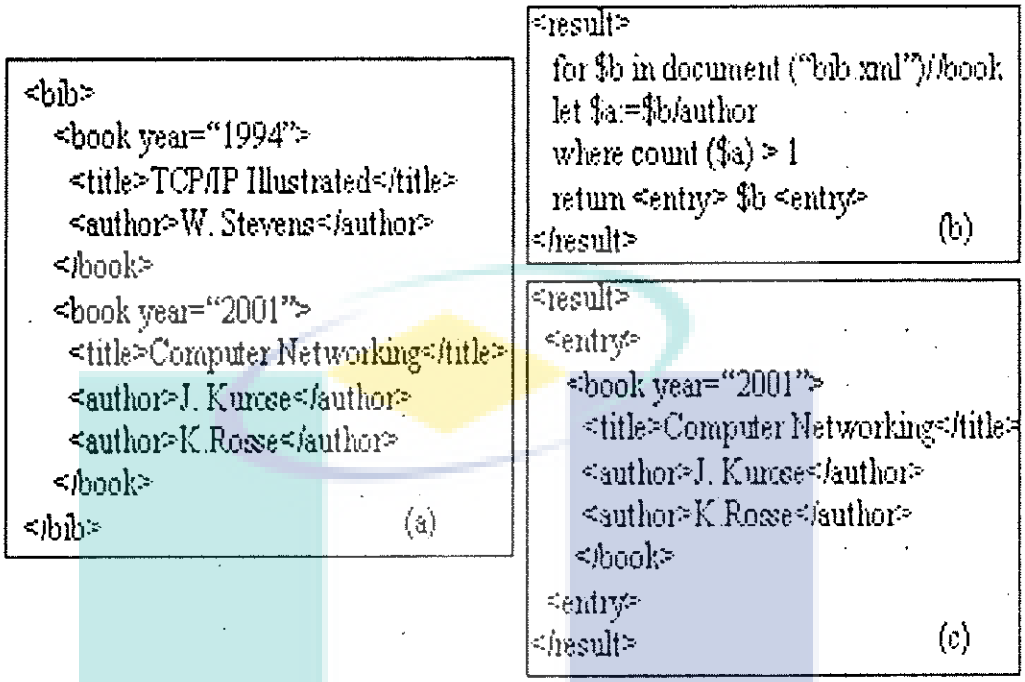


Figure 3.5 (a) AN *XML* bib text (b) an *XQuery* query posted on bib text and (c) the generated XML result.

bib	book'	book	author'	author
<bib> ... </bib>	<bib> <book year="1994">... </book>...</bib>	<book year="1994">... </book>	<bib> <book year="1994">... <author>W.Stevens</author> </book>...</bib>	<author> W.Stevens </author>
<bib> ... </bib>	<bib>... <book year="2001">... </book></bib>	<book year="2001">... </book>	<bib>... <book year="2001">... <author>J.Kurose</author>... </book>...</bib>	<author> J.Kurose </author>
<bib> ... </bib>	<bib>... <book year="2001">... </book></bib>	<book year="2001">... </book>	<bib>... <book year="2001">... <author>K.Rosse</author> </book>...</bib>	<author> K.Rosse </author>

Figure 3.6 Results of *extraction* with a tree pattern that flags book and author nodes.

3.7 Processing Operations

Evaluation is necessary for all databases with different tools at benchmark environment. Namely a serial of determination step will carried out on the two databases; the details is as follows:

In XML_Enabled database needs mapping (SQL to XML and XML to SQL). The mapping between the database columns and XML documents (elements or attributes) is defined by means of AS aliases in a SELECT:
<Database column> AS [Element Name! Nesting Level! Attribute Name! Directive]

3.7.1 Storing XML Documents

The process is done by use OPENXML function. OPENXML is suited to load XML data into the database, because it will load the whole XML document into the memory before perform the insertion of the data. The process has three steps:

(i) Obtain an XML document handler by compiling XML document into internal DOM representation, using the stored procedure sp_xml_preparedocument.

(ii) Construct a schema by associating schema fields with XML elements. Where XML elements are defined by a path pattern in addition to a relative element path.

(iii) Remove the compiled XML document from memory by using the stored procedure sp_xml_removedocument.

3.7.2 Extracting XML Documents

(i) XML_Enabled Database

It is necessary to generate Xml documents again for the databases, it has two steps:

(a) Create aliases to the elements in the desired output XML. The alias define the parent/child relationships between elements.

```
FXTRADE         /* LEVEL=1 */
    AUTHOR1     [FXTRADE!1!AUTHOR1]
    AUTHOR2     [FXTRADE!1!AUTHOR2]
    TILTE        [FXTRADE!1!TILTE]
    ISBN   [FXTRADE!1!ISBN]
    QUANTITY    /* LEVEL=2  */
        NUMBER          [QUANTITY    !2!NUMBER]
        PRICE       [QUANTITY    !2!PRICE]
```

(b) Define the output tree structure in SQL. Each level of the tree is defined through a SELECT statement, thereafter the levels are combined together into the tree by means of a UNION ALL statement. The level-1 SELECT statement introduces the names of atomic elements on all levels. Each SELECT statement introduces a tree level tag and its parent tag. There is a single record in the result set corresponding to the tree root.

```
SELECT
    1           AS    Tag,
    NULL        AS    Parent,
    NULL        AS    [FXTRADE!1!AUTHOR1],
    NULL        AS    [FXTRADE!1!AUTHOR2],
    NULL        AS    [FXTRADE!1!TITLE],
    NULL        AS    [FXTRADE!1!ISBN],
    NULL        AS    [QUANTITY    !2!NUMBER],
    NULL        AS    [QUANTITY    !2!PRICE]
FROM
    FXTRADE
UNION ALL
SELECT
    2,
    1,
    FXTRADE.AUTHOR1,
```

```
    FXTRADE.AUTHOR2,
    FXTRADE.TITLE,
    FXTRADE.ISBN,
    QUANTITY    .NUMBER,
    QUANTITY    .PRICE
FROM
    FXTRADE,     QUANTITY
WHERE
    FXTRADE.QUANTITY     = QUANTITY    .ID
ORDER     BY     [QUANTITY    !2!ISBN],
          [QUANTITY    !2!PRICE]
FOR     XML     EXPLICIT, ELEMENTS
```

(ii) Native XML Database

For extract document (A SELECT-FROM-WHERE) has been used. For example, the query computes a list of the books for the customer 12.

```
SELECT x.txt.extract('Customer//Book)
FROM XMLTypeTab x
WHERE x.txt.existsNode('Customer//Order') = 1
AND x.txt.extract('/Customer/CNo/text()').getNumberVal() = 12
```

Extract-condition is used also in UPDATE and DELETE documents.

3.7.3 Inserting XML Documents

(i) XML_Enabled Database

For single record, insert XML data into a XED done by using an INSERT statement and the OPENXML function.

For Mass record, the method for inserting as following:

(i) Load the XML documents as a stream and reads it.

(ii) Identifies the database table and generates the appropriate records from the XML.

(iii) Send the records to the database for insertion.

3.7.4 Deleting and Updating XML Documents

(i) XML_Enabled Database

For UPDATE and DELETE XML documents, the given document is taken as a pattern for qualifying documents, then the he document determines the query values of the resulting WHERE condition. Where use <xsql:update-request>, and <xsql:delete-request> to update and delete documents.

(iii) Native XML Database

The delete process has been done by using delete XML DML statements as following:

(a) An XML is assigned to variable of xml type.

(b) Delete various nodes from the document.

3.7.5 Searching XML Documents

(i) XML_Enabled Database

This study has been used an approach to search XML documents in the relational database as follows:

(a) Process the DTD file (figure 3.4) to generate a relational schema

(b) Load XML documents into relational tables in the XML_Enabled database after Parse and conforming to DTDs.

(c) Translate semi-structured queries over XML documents into SQL queries over the corresponding relational data.

(d) Convert the results back to XML

(iv) Native XML Database

This study has been used several techniques for the searching process in Native XML database for better performance:

(a) Shortest Paths

In Native XML database there's no need to pass the entire document tree because of the database directly determines an element and attribute by its name by using the indexing. This means, to search about the title for a book in the cookbook catalog (single record), the direct selection of a node through a single descending step:

```
doc('cookbook.xml')/cookbook/recipe/title
```

(a) Process the most selective filter/expression first

For multiple steps to select certain nodes from a larger node set (mass record), will process the most selective steps, by reducing the node set to be processed, thus will increase the speed of queries:

```
/dblp/*[year > 2003][author = 'Kanda Runapongsa']
```

The database has 568824 records matching year > 2003, but only 53 of them were written by Stroustrup. For better performance, need to move the author name to the front of the expression:

```
/dblp/*[author = 'Kanda Runapongsa'][year > 2003]
```

Native XML database a kind of optimization automatically, and it is recognized more cases for intelligent searching rewritings than XML_Enabled database. For example, has already transformed the Boolean expression

```
/dblp/*[author = 'Kanda Runapongsa' and year > 2003]
```

(a) Avoid unnecessary nested filters

Unnecessary nesting should be avoided because of its negative impact on the searching. The variant with only one filter is easier to optimize for the database, whereas the nested filter implies a performance penalty. Likewise, if you are calling one of the optimized functions (contains, matches, ft:query ...), need to make sure not nest them unless really required:

```
//Books[book/Title[contains(., "ISBN")]]
//Books[contains(book/Title, "ISBN")]
```

(a) General comparison to compare an substance to a list of alter natives

General comparisons are used to compare a given substance to several alternative values. For example, you could use an "or" to find all <book> children whose string value is either "Title" or "ISBN".

```
//Books[book eq 'Title' or book eq 'ISBN']
```

A shorter way to express this is:

```
//Books[book = ('Title', 'ISBN')]
```

The comparison will be true if b's string value matches one of the strings in the right hand sequence. If an index is defined on <book>, the database will need only one index lookup to find all book's matching the comparison. The equivalent "or" expression needs 2 separate index lookups.

3.8 Conclusion

This chapter presented a series of experiments that has been applied in XML_Enabled database and Native XML database. The experiments have been tested using database benchmark (XMark and XMark-). The performance is measured using different sizes of data sets (65.8, 101, 117, 127, 183 MB) as shown in the Table 3.2. The size of XML documents and the number of elements have been determined by the factor of the main driver of generation. This chapter also presented the methodology of this study to evaluating: insert, update, delete and search performance in both databases XML_Enabled database and Native XML database) which applied the method that has

been developed by K. Williams with his research teams his research teams in 2001 (Williams et al. 2001) (Figure 3.4).

CHAPTER 4

EXPERIMENTATION AND DISCUSSION

4.1 Introduction

The study implemented the data sets in both databases, Native XML database and XML_Enabled database, and used the PC machine with Intel Core i7-3770 processor (3.40 GHz), 4GB main memory, and Windows 7 Professional. The data sets are used from smallest 65.8 MB to largest 183 MB. Compare both databases in terms of time complexity when evaluating the performance.

4.2 Discussions

For modeling the columns in XML_Enabled databases as XML DTD structure, there are two approaches:

1) Element approach: Define the table name as the root element. It is nested by its columns, which are also defined as elements. An example is:
   (i) <! ELEMENT Person (Name, Sex, Age)>
   (ii) <! ELEMENT Name (#PCDATA)>
   (iii) <! ELEMENT Sex (#PCDATA)>
   (iv) <! ELEMENT Age (#PCDATA)

2) Attribute approach: The columns are defined as attributes of the root element. The previous example, becomes

(i) <! ELEMENT Person>

(ii) <! ATTLIST Person

(iii) Name CDATA #REQUIRED

(iv) Sex CDATA #REQUIRED

(v) Age CDATA #REQUIRED>

In XML_Enabled database, data and structures are defined. Columns represent data. Tables and relationships form structure. This can be managed well in searching for data and for database navigation. XML attributes refer to the data. XML elements and sub-elements build the structure. In addition, attributes do not have the concept of ordering. This is similar to columns in a table. No matter how one changes the position of a column in a table, the data content inside a table does not change. For the first approach, tables and columns are both defined as element types. It may be ambiguous to a parser to decide the role of an element. The flexibility of searching for child elements is less than the attribute approach. This is because an element does have ordering meaning. Hence, it cannot fully represent the location-independence of data from an XED concept.

The performance is an important issue. There are two technologies in parsing XML documents: DOM and SAX. The thesis's methodology used only Native XML database Java DOM API.

Firstly, DOM technology pulls XML documents into the memory and presents it as the tree later. The process of converting the document to a tree structure involves traversing through the document. For example, the steps for retrieving the title of the 5th substance from books are:

1) Go to parent element Book
2) Go to 5th book_ISBN child of Books
3) Get the title name from this book_ISBN.

If the element approach is used in the sample database, more steps are involved:

1) Go to parent element Books.

2) Go to second book_ISBN child.
3) Go to title name child of the second book_ISBN.
4) Get the name portion of book title.

Coding may be simpler if the attribute approach is used. Also, when using attributes, there is the option of using enumerated types such that the value of a column can be constrained by a defined value.

The size of the documents: For element approach, it is necessary to be (Starting tag+Content+ Ending tag). But this is not necessary to attribute approach (attribute_name = attribute_value). When increasing of the database, the difference could be significant. For Parsing documents, the element approach cost more time. So, need more disk space to store the tags, especially in when storing a group of data (mass record). Thus, the performance will be affected.

In defining the relationships between elements, containment is used for one-to-one and one-to-many cases. The ID/IDREF pointer approach is not recommended because XML is designed with the concept of containment. Using pointers costs more processing time, because DOM and SAX do not provide efficient methods to handle ID/IDREF relationships. Furthermore, navigating from an ID field to an IDREF field may not be easy. This becomes more difficult for IDREFS, since all IDREFS fields need to be tokenized. Each token is examined for all possible ID fields. Hence, containment is introduced to build relationships at the start. The pointer approach is used for those relationships that can go either way.

4.3 Experimental Results

4.3.1 Storing and Extracting Complete XML Documents

Native XML database provides efficient methods for inserting an XML document into the database and extracting the complete document from the database, the following table shows the result. The results for table 4.1 indicate that, Native XML database has better performance than XML_Enabled database to storing XML documents (Because of indexing in Native XML database allows to perform operations

in memory (which is fast), rather than reading from disk (which is slow) (eXist-db, 2013; Oracle group, 2011)). While the result in table 4.2 shown that XML_Enabled database has better performance to extracting XML documents, because in XML_Enabled database, each level of the tree is defined through a SELECT statement, thereafter the levels are combined together into the tree by means of a UNION ALL statement (see 3.9.2)

Table 4.1 Storing Complete XML Documents

Size/MB	**Time/Second**	
XML Documents	**XML_Enabled Database**	**Native XML Database**
65.8	32.0	8.6
101	69.9	9.4
117	280	12.7
127	1732	31.5
183	2057	38.2

Table 4.2 Extracting Complete XML Documents

Size/MB	**Time/Second**	
XML Documents	**XML_Enabled Database**	**Native XML Database**
65.8	3.45	9
101	4.1	10.2
117	4	12
127	4.1	17.9
183	4.2	32

Table 4.3 Provides a summary of the tests that have been used performance evaluation.

Test	Test Description
Query1, Query2	Evaluate the time for Insert single record performance
Query3, Quer4	Evaluate the time for Insert mass records performance
Query5, Quer6	Evaluate the time for Update single record performance
Query7, Query8	Evaluate the time for Update mass records performance
Query9, Query10	Evaluate the time for Delete single record performance
Query11, Query12	Evaluate the time for Delete mass records performance
Query13, Query14, Query15	Evaluate the time for searching for record by using index key

4.3.2 Inserting Process

(i) Single Records

The objective of Query1 and Query2 are to measure insert performance Query1 is a single insert statement with only one substance (item) involved. As the results indicate, when data size increases, XML_Enabled database's performance always seems efficient, whereas it consume about 75% time that the Native XML database consume, due to the technique of storing and organizing the huge data inside Native XML database. The XML-enabled database has better performance than the native XML database in all cases. However, both products have steady figures no matter how large the database is. See Figure 4.1.

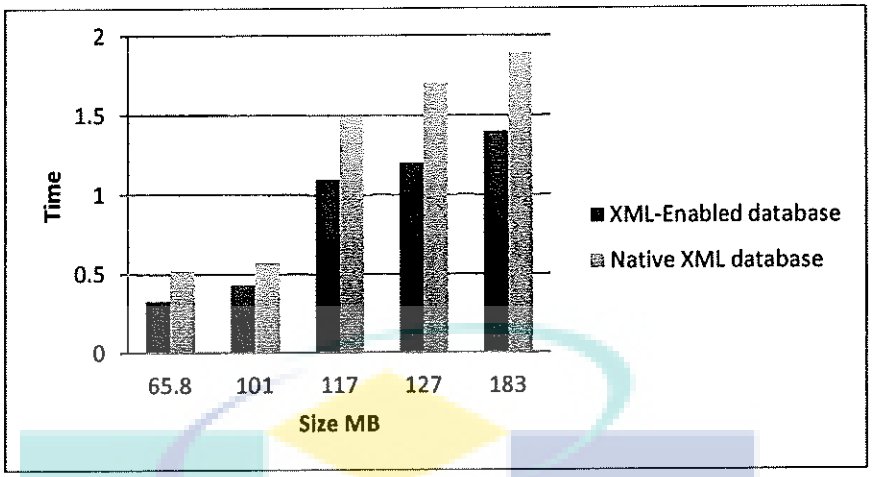


Figure 4.1 *Query1*. Insert one substance record into the table

Query2 consists of a master-details relationship (Client and Client-add). In Query2, concluded that insert operation performance in not effected by the database size. Furthermore, Query2 costs more time than Query1 as Query2 needs to handle more than one substance (Item). All in all, the same result that the XML_Enabled database has better performance than the Native XML database. The XML-enabled database has better performance than the native XML database in all cases. However, both products have steady figures no matter how large the database is. See Figure 4.2.

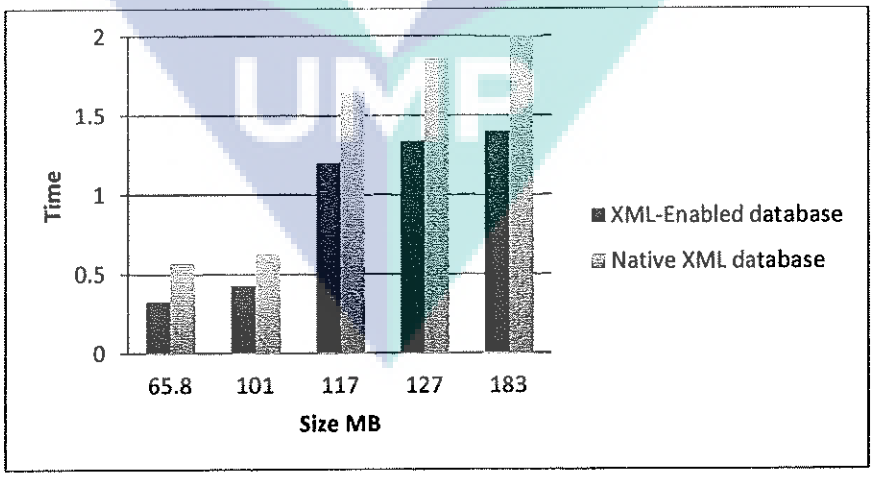


Figure 4.2 *Query2*. Insert complete customer record into the tables

(ii) Mass Records (group of Data)

Query3 and Query4 are to evaluate the inserting process for group of data. XML_Enabled database, has better performance than Native XML database for data size (<= 101 MB). See Figure 4.3.

As data size increases more than 101 MB, Native XML database's performance will be better choice, whereas it consume about half the time that the XML_Enabled database consume, due to the technique of storing and organizing the huge data inside Native XML database. Moreover, the API in XML_Enabled database, cause the problem of congestion and accumulation for huge size data. See figure 4.4

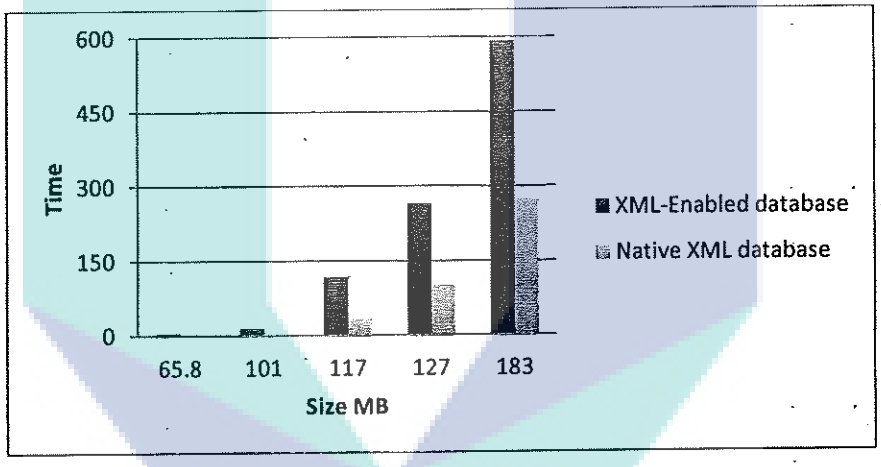


Figure 4.3 *Query3.* Insert Group of data

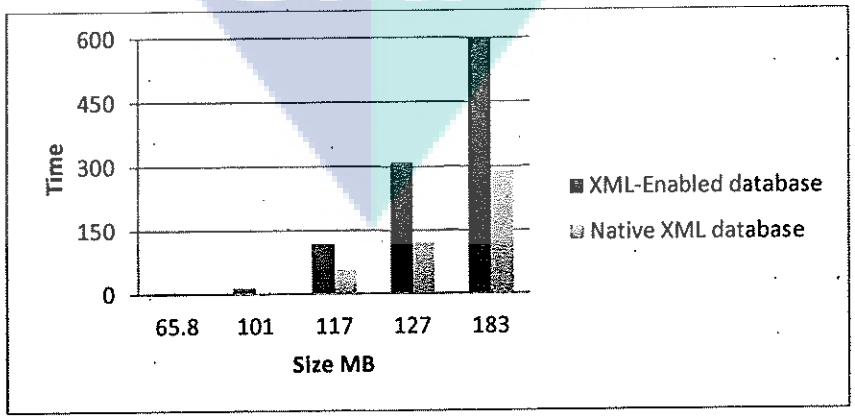


Figure 4.4 *Query4.* Insert complete customer record

4.3.3 Updating Process

(i) Single Records

The objective of Query5 and Query6 are to measure update performance Query5 is a single update statement. As the results indicate, when data size increases, XML_Enabled database's performance always seems efficient, whereas it consume about 50% time that the Native XML database consume, due to the technique of storing and organizing the huge data inside Native XML database. The XML-enabled database has better performance than the native XML database in all cases. However, both products have steady figures no matter how large the database in Figure 4.5.Query5 and Query6 to evaluate update performance. The consumed time in the Update process is less the consumed time in insert process. The reason behind this is that, insert examines and check carefully data validity and unique indexing process, while in Update and delete process, the data have been taken from the database, and so the time is saved.

Same results have been obtained (acquired) for the process insert single record in Figure 4.6, since XML_Enabled database has better and faster performance than a Native XML database. Update process has the same time for execution regardless the size of the database.

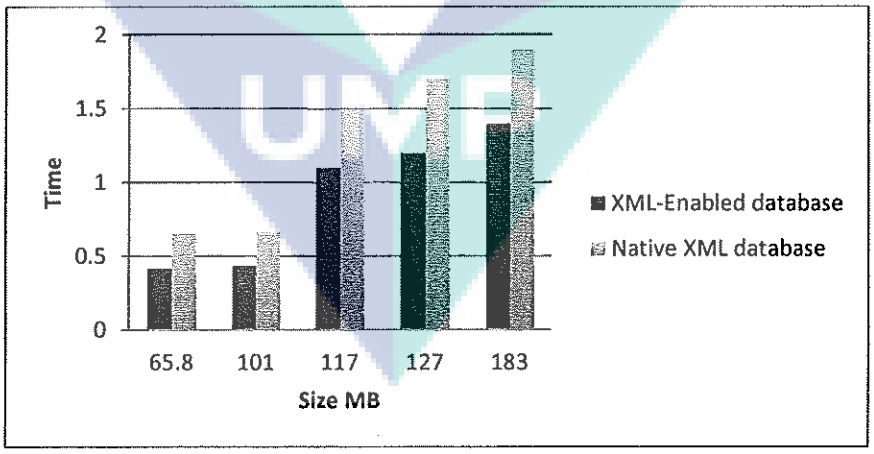


Figure 4.5 *Query5*. Update one record in the table

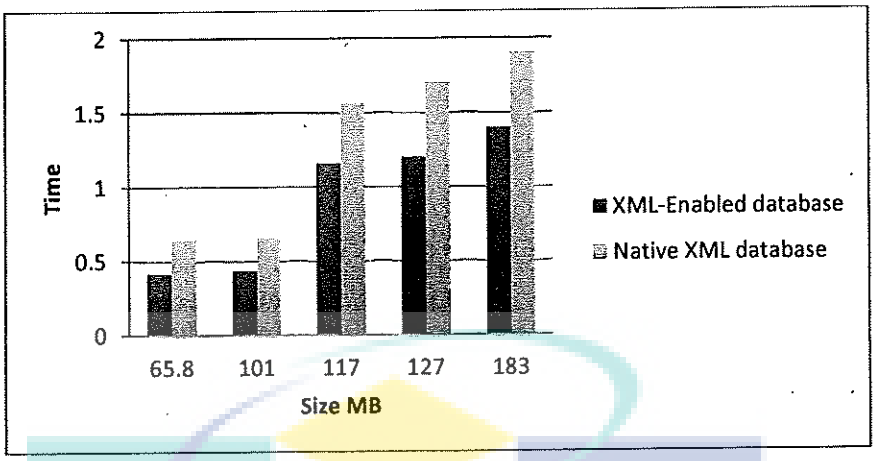


Figure 4.6 *Query6.* Update complete client record in the tables

(ii) Mass Records (Group of Data)

Query7 and Query8 are to evaluate the time for Update mass records performance. In Update Mass record process, the performance of XML_Enabled database still better than Native XML database for the data (<=127 MB). Native XML database's performance is better for huge data (>=183 MB). This is because XML_Enabled database has simple query and effective structure to perform Updata/Mass record, while Native XML database need to execute additional query process before retrieval the data whereof cause consumption of more time with data (<=127 MB)

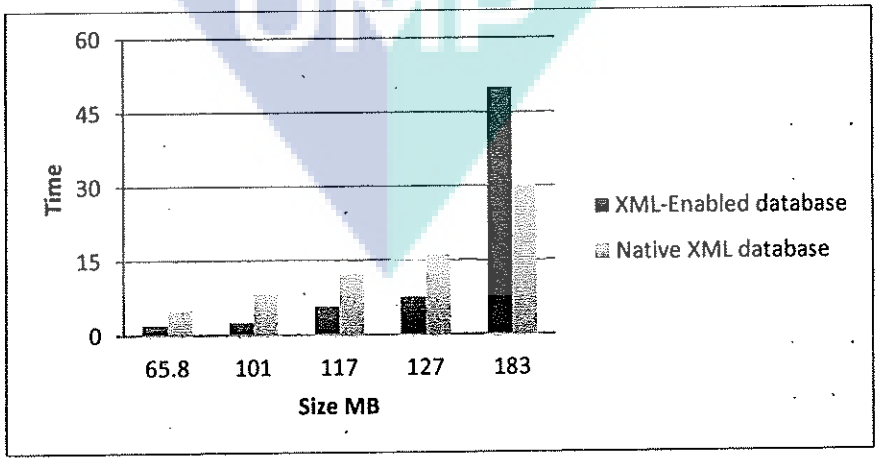


Figure 4.7 *Mass Update Substance*

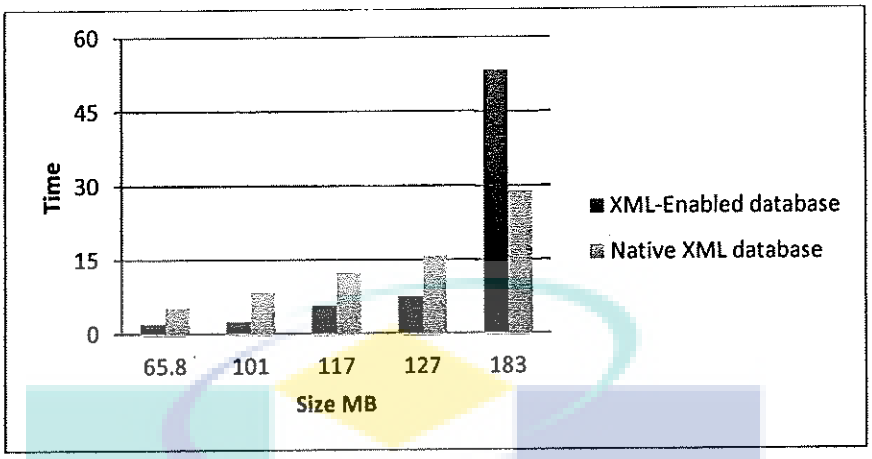


Figure 4.8 Mass Update complete *client record*

4.3.4 Deleting Process

(i) Single Records

Query9 and Query10 are to evaluate Deleting single record performance. Results have been obtained similar to the results obtained by query Insert (Query1, Query2), and Update (Qery5, Quer6). XML_Enabled database has better performance than Native XML database.

Delete process has similar functions to Update process, where there is no big difference in the performance between the two processes (Delete process and Update Process, and both processes to be effected with database's size.

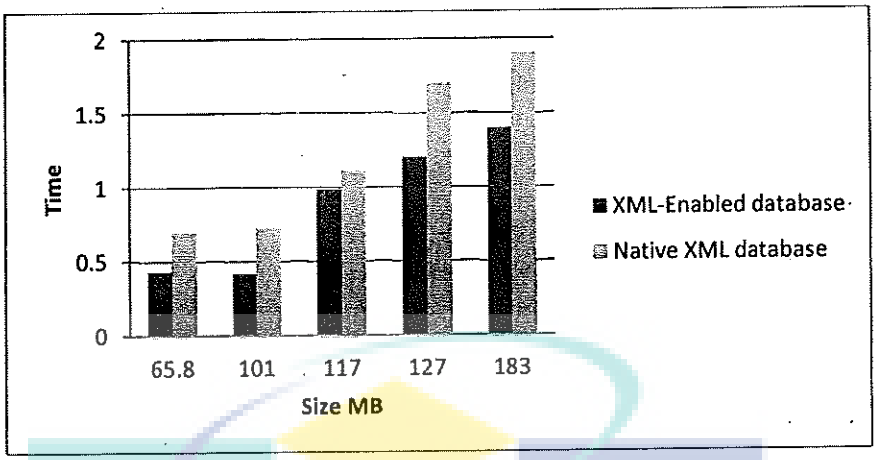


Figure 4.9 *Query9.* Delete one Substance record from the table

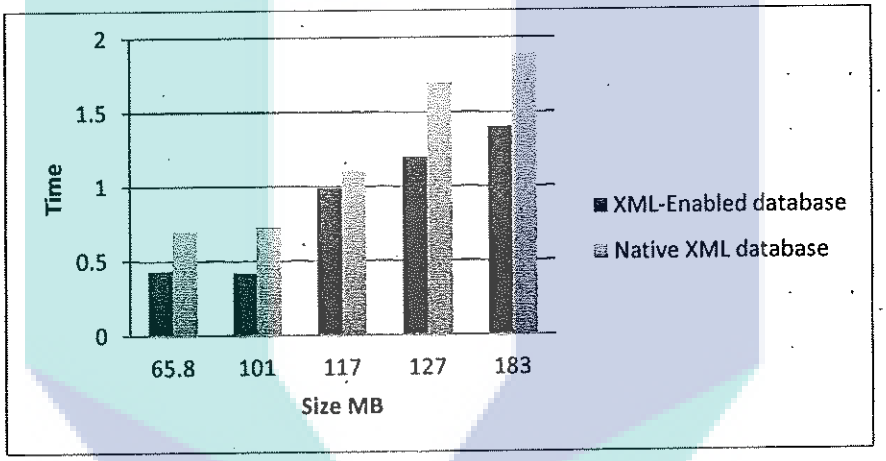


Figure 4.10 *Query10.* Delete complete client record from the tables

(ii) Mass Records (Group of Data)

Query11 and Query12 are to evaluate the performance of Delete group of data. Delete operations for Substance and Client records (see Figure 11). In Delete process (Mass record), as shown in figure11 and figure12, the XML_Enabled database has much better performance than Native XML database with all data size which have been tested in this research except data size (<183), and the reason behind this has already explain earlier in (Update Mass record).

For the XML_Enabled database, a simple structural and powerful SQL query can perform a mass delete. In contrast, the servlet program for the Native XML database

needs to execute an additional query prior to retrieving all possible Clients/Substance s. Then the program uses the temporary list to remove records.

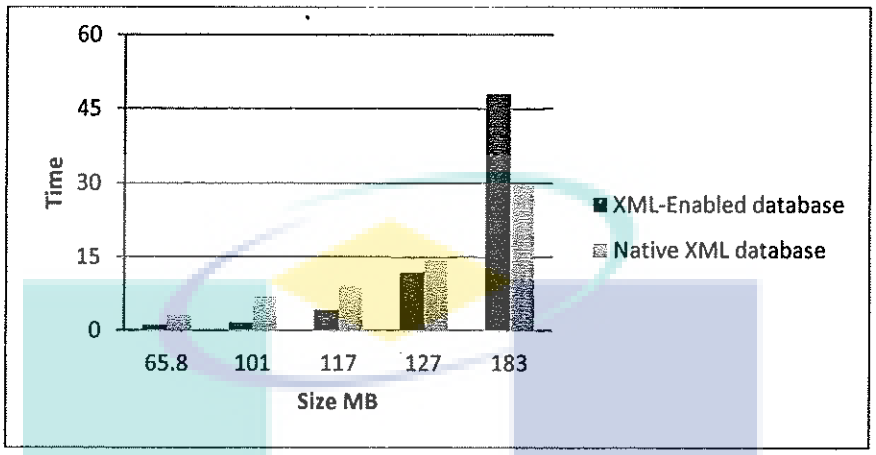


Figure 4.11 *Query11*. Mass delete Substance

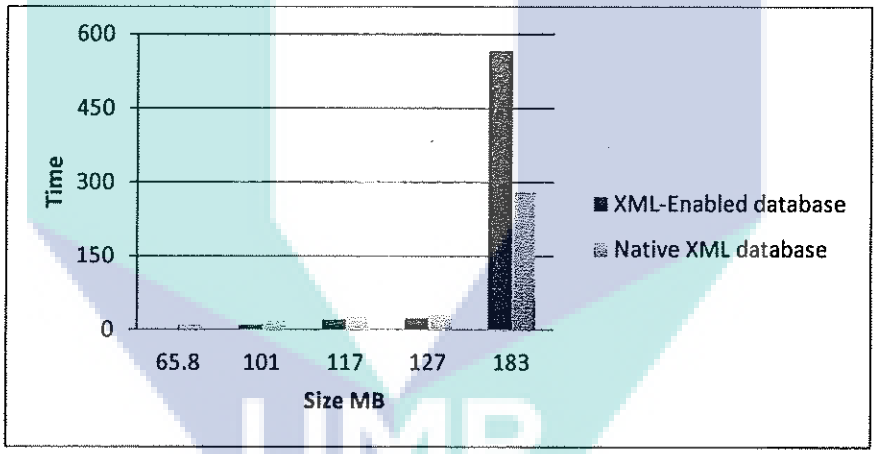


Figure 4.12 *Query 12*. Mass delete complete client record

4.3.5 Searching Process

Query13, Query14, and Query15 evaluate the time to search for a record using an index key (Figure 4.14). For 65.8 or 101MB, the results for both products are very similar. From 183 MB, the Native XML database outperforms the XML_Enabled database. The Native XML database provides steady performance in all cases. We conclude that the Native storage strategy and indexing approach are efficient enough for searching in a database.

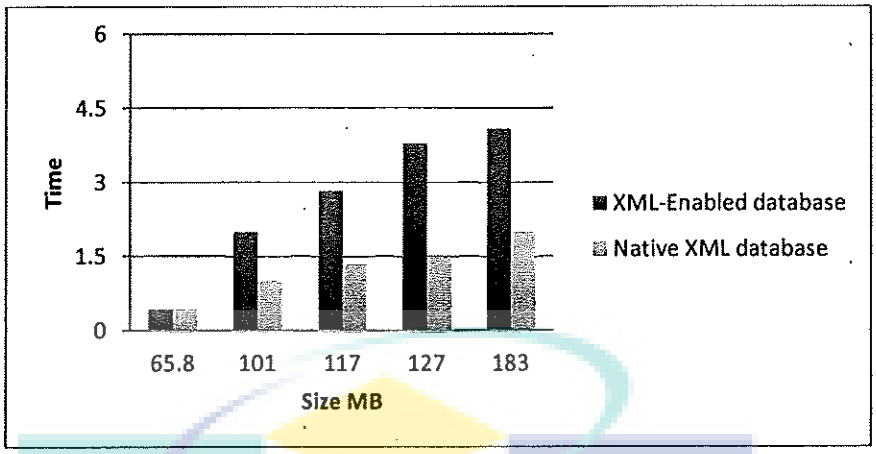


Figure 4.13 *Query13.* Searching an Substance in the table

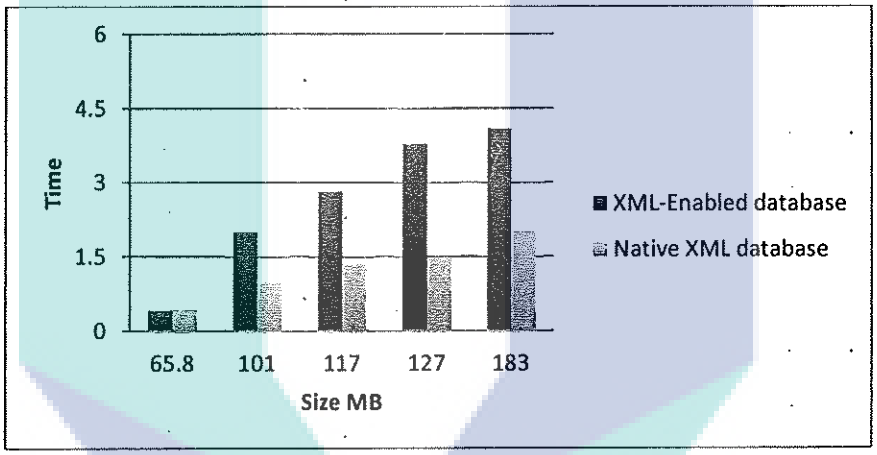


Figure 4.14 *Query14.* Searching complete client record in the tables

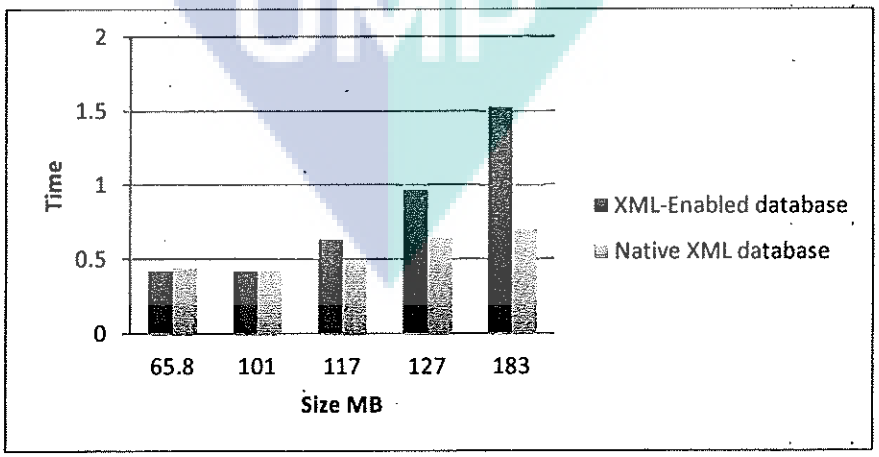


Figure 4.15 *Query15.* Searching complete bill record in the tables

The above result because in a relational database, data and structures are defined. Columns represent data. Tables and relationships form structure. This can be managed well in searching for data and for database navigation. XML attributes refer to the data. XML elements and sub elements build the structure. In addition, attributes do not have the concept of ordering. This is similar to columns in a table. No matter how one changes the position of a column in a table, the data content inside a table does not change. For the first approach, tables and columns are both defined as element types. It may be ambiguous to a parser to decide the role of an element. The flexibility of searching for child elements is less than the attribute approach. This is because an element does have ordering meaning. Hence, it cannot fully represent the location-independence of data from an RDBMS concept.

4.3.6 Results Discussions

In conclusion, the results implemented the data sets in both databases, Native XML database and XML_Enabled database. Native XML database has better performance than XML_Enabled database to storing XML documents (Because of indexing in Native XML database allows to perform operations in memory (which is fast), rather than reading from disk which is slow.

In Single Records of Inserting Process, as the results indicate, when data size increases, XML_Enabled database's performance always seems efficient, due to the technique of storing and organizing the huge data inside Native XML database. The XML-enabled database has better performance than the native XML database in all cases. However, both products have steady figures no matter how large the database is. In Mass Records of Inserting Process, Native XML database's performance will be better choice, whereas it consume about half the time that the XML_Enabled database consume, due to the technique of storing and organizing the huge data inside Native XML database. Moreover, the API in XML_Enabled database, cause the problem of congestion and accumulation for huge size data. The consumed time in the Update process is less the consumed time in insert process. The reason behind this is that, insert examines and check carefully data validity and unique indexing process, while in

Update and delete process, the data have been taken from the database, and so the time is saved.

In Update Mass record process, the performance of XML_Enabled database still better than Native XML database for the data (<=127 MB). Native XML database's performance is better for huge data (>=183 MB). This is because XML_Enabled database has simple query and effective structure to perform Updata/Mass record, while Native XML database need to execute additional query process before retrieval the data whereof cause consumption of more time with data (<=127 MB)

Delete process has similar functions to Update process, where there is no big difference in the performance between the two processes (Delete process and Update Process, and both processes to be effected with database's size. In Delete process (Mass record), the XML_Enabled database has much better performance than Native XML database with all data size which have been tested in this research except data size (<183), and the reason behind this has already explain earlier in (Update Mass record). For the XML_Enabled database, a simple structural and powerful SQL query can perform a mass delete. In contrast, the servlet program for the Native XML database needs to execute an additional query prior to retrieving all possible Clients/Substance s. Then the program uses the temporary list to remove records.

In Searching Process, the Native XML database outperforms the XML_Enabled database. The Native XML database provides steady performance in all cases. We conclude that the Native storage strategy and indexing approach are efficient enough for searching in a database.

4.4 Native XML Databases Validation

DTDs and XML schemas, to date, have not been critical or necessary components of most available NXD or XML-enabled database solutions. While validation may be useful in initial loading of XML collections, subsequent processing behaviors once loaded into DBMS system do not require a DTD to be in place. NXDs assume XML well-formedness (Schmidt et al., 2002). For example, on updates that

might validate the referential integrity of between elements or attributes, that pre-update or pre-insert transactional processing is supported. The alternative brute force approach would be to update the XML document and then check for constraint violations. If a constraint is violated, a transactional rollback is initiated. This approach requires a great deal of DBMS overhead and available resources required to support the rollback (Arenas et al., 2002).

Using our proposed method of using Native XML databases to enforce key constraints, both unique key constraints and foreign key constraints, this section starts with the personal-address.xml document. For example, given that address name is a unique key with our data model, it would implement the following statement on the supported nextgen NXD to enforce uniqueness:

```
CREATE
BEFORE INSERT OF document ("personal-address.xml")//address
FOR EACH NODE
DO
{
for $p in doc("pa.xmry/address/@name
where $p = $NEW
return
if(count($p)>0)then
$p
else <error>Unique Key Constraint Violation</error>
}
```

Next, it assumes a DTD segment for personal-address having the specified attribute constraint, where only those specified zip codes are allowed.

```
<!ATTLISTcity
zip (94783 194303 | 90472 | 90092) #REQUIRED
```

Those NXDs that do not support DTD validation would have the following statement for check constraint validation, validating reference data, similar to a foreign key:

```
CREATE TRIGGER TRGJ»AJ)2
BEFORE UPDATE, INSERT ON document ("personal-address.xml")//address
FOR EACH NODE
DO
I
for $p in doc("pa.xml")//address/city/@zip
where ($NEW/@zip - 94783) or ($NEW/@zip = 94303) or
($NEW@2ip = 90473) or ($NEW/@zip = 90092
Return
if (fn:empty($p)) then
<error>Check Constraint Violation</error>
else $p
}
```

In the above example, if the city zip code in not in the list of valid values, the XML document would fail the DTD validation because it does not match type defined in the enumeration list. We added a check constraint to validate reference data. Alternatively, by modifying our original DTD to the following, we can implement foreign key constraint checking

```
<?xml version-"].0" encoding="UTF-8"?>
<!--DTD generated by XMLSpy v2009 spl (http://www.altova.com)->
<!ELEMENT street (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT personal-address ((address+,validzip))>
<!ELEMENT count}' (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

```
<!ATTLTST city
zip (94783 194303 190472 | 90092) #REQL'IRED
>
<!ELEMENT address ((street, state, city, county, country))>
<!ELEMENT address ((street, state, city, county, country))>
<!A II LIST address
name CDATA #REQU1RED
>
<!ELEMENT zipcode (#PCDATA)>
<!ELEMENT validzip ((zipcode))>
```

The augmented DTD of personal-address might require the following in an NXD database to validate valid zip codes similar to relational foreign key validation. Note the use of XQuery to achieve this in the code below.

```
CREATE A_03
BEFORE UPDATE. INSERT ON document ("personal-address.xml")//address
FOR EACH NODE
DO
{
for $p in doc("pa.xml")//address/eity/@zJp
for $x in doc("pa.xml")//validzip/zipcode
where $NEW - p$ and $p = $x
return
if (fn:empty($p)) then
<error>Constraint Violation</error>
else $p
}
```

While it is clear through varied research in the field of apply key constraints to XML documents (Arenas et al., 2002; Schmidt et al., 2002), it is also still true that implementation of unique key and foreign key constraints is a valuable mechanism to enforcing data integrity. With the use of triggering mechanisms added to a native XML

DBMS, providers can offer developers with a useful tool in enforcing data integrity using a standard query language (XQuery), when manipulating XML data.

The validation results can be summarized as follows. Firstly, this section starts with the personal-address.xml document. Next, it assumes a DTD segment for personal-address having the specified attribute constraint, where only those specified zip codes are allowed. Those NXDs that do not support DTD validation would have the statements for checking constraint validation and validating reference data. Finally, the augmented DTD of personal-address might require the following in an NXD database to validate valid zip codes similar to relational foreign key validation. Note the use of XQuery to achieve the validation. The result summarized as shown in the table 4.4.

Table 4.4 Summarized result

Indicadition	**Native XML database**	**XML_Enabled database**
Commands	DTD	XDR
Performance (single record)	XQL	XPath
Performance (mass records)	Low	High
User Interface:	High	Low
User Friendliness	Good	Good
Implementation	Good	Fair
Maintenance	Fair	Good

4.5 Conclusion

This chapter implemented the data sets in both databases. The data sets are used from smallest 65.8 MB to largest 183 MB. Each measurement is repeated three times, and the average value is taken. Examine the efficiency and the time for restore and rebuild the XML document from both database (XED and NXD) in seconds. Compare both databases in terms of time complexity when evaluating the performance.

CHAPTER 5

EXTENSIVE DISCUSSION

5.1 Introduction

With the popularity of XML and increasing amount of information stored and exchanged using XML, efficient hosting of XML stores and efficient processing of XML queries becomes important in the database community. Conventional database vendors provide relational support for XML data, e.g., DB2 (IBM), Microsoft SQLXML (Microsoft) and Oracle 9i (Oracle). However, there is inherent impedance mismatch between the relational (sets of tuples) and XML (unranked trees) data models. An alternative approach to using relational databases for XML data is to build a specialized XML data manager, i.e., one which can reflect the hierarchical structure of the XML data. This is referred to as a native XML database. In native XML databases, XML data is generally modelled as trees, where tree nodes represent XML elements, attributes and text data, and edges for element/sub-element relationship. In this chapter, the extensive discussion is presented for storing and extracting complete xml documents, inserting process, updating process, deleting process and searching process. Also, validation results is further analyzed. The three objectives has been achieved by the proposed algorithms.

5.2 Analysis and Discussion for Evaluating the Native XML Database Performance

There are several desirable features for the XML database storage systems. First, the difficulty of converting the shredding of XML data into relational tables increased with increasing the data size (Zhang and Tompa, 2004). Secondly, the storage system should be robust enough to store any XML documents with arbitrary tree depth or width, with any element-name alphabet, and with or without associated schemata. Moreover, local update to the document should not cause drastic changes to the whole storage system. Therefore, the design of the storage system should trade-off between the query performance and update costs.

In the first research question, it is difficult to find the storage system for XML documents to support efficient evaluation as well as storing and updating XML documents. Therefore, to evaluate the Native XML database performance in a comparison with XML_Enabled database, the proposed algorithm determines the points of difference between the structure of XML documents and the structure of the database that contain, understand negative effects of increasing the number of these data within databases. These points were determined by added a fixed number of XML documents are assigned to be imported into the databases. The documents are repeated from the smallest to the largest for three times, and the average value is taken. The first objective is achieved.

5.3 Analysis and Discussion for Enhancing Entity Relationship Algorithm of the Relational Schema

In the second research question, how to evaluate path expressions efficiently for different types of queries? Therefore, to enhance entity relationship (EER) algorithm of the relational schema to improve Insert, Delete, Update and Search XML document and compare the performance of XML_Enalbled database and Native XML database, by implementing the same command and control data model. The experimental results show that Native XML database has better performance than XML_Enabled database to storing XML documents (Because of indexing in Native XML database allows to perform operations in memory (which is fast), rather than reading from disk which is slow. In Single Records of Inserting Process , as the results indicate, when data size increases, XML_Enabled database's

performance always seems efficient, due to the technique of storing and organizing the huge data inside Native XML database. The XML-enabled database has better performance than the native XML database in all cases. However, both products have steady figures no matter how large the database is. In Mass Records of Inserting Process, Native XML database's performance will be better choice, whereas it consume about half the time that the XML_Enabled database consume, due to the technique of storing and organizing the huge data inside Native XML database. Moreover, the API in XML_Enabled database, cause the problem of congestion and accumulation for huge size data. The consumed time in the Update process is less the consumed time in insert process. The reason behind this is that, insert examines and check carefully data validity and unique indexing process, while in Update and delete process, the data have been taken from the database, and so the time is saved. In Update Mass record process, the performance of XML_Enabled database still better than Native XML database for the data (<=127 MB). Native XML database's performance is better for huge data (>=183 MB). This is because XML_Enabled database has simple query and effective structure to perform Updata/Mass record, while Native XML database need to execute additional query process before retrieval the data whereof cause consumption of more time with data (<=127 MB). Delete process has similar functions to Update process, where there is no big difference in the performance between the two processes (Delete process and Update Process, and both processes to be effected with database's size. In Delete process (Mass record), the XML_Enabled database has much better performance than Native XML database with all data size which have been tested in this research except data size (<183), and the reason behind this has already explain earlier in (Update Mass record). For the XML_Enabled database, a simple structural and powerful SQL query can perform a mass delete. In contrast, the servlet program for the Native XML database needs to execute an additional query prior to retrieving all possible Clients/Substance s. Then the program uses the temporary list to remove records. In Searching Process, the Native XML database outperforms the XML_Enabled database. The Native XML database provides steady performance in all cases. We conclude that the Native storage strategy and indexing approach are efficient enough for searching in a database. In all, it explores the functionality of the databases (Native XML Databases & XML_Enabled Databases) in handling big sizes of XML documents. By evaluate and compare the performance of the data queries in both databases. Then create a Native XML method that Insert, Delete, Update and Search Big sizes of XML document. The second objective is achieved.

5.4 Analysis and Discussion for Validating the Algorithm in Native XML Databases

In the third research question, the thesis solved how to validate the algorithm in Native XML databases by an example. It is clear through varied research in the field of apply key constraints to XML documents, it is also still true that implementation of unique key and foreign key constraints is a valuable mechanism to enforcing data integrity. The validation results can be summarized as follows. Firstly, this section starts with the personal-address.xml document. Next, it assumes a DTD segment for personal-address having the specified attribute constraint, where only those specified zip codes are allowed. Those NXDs that do not support DTD validation would have the statements for checking constraint validation and validating reference data. Finally, the augmented DTD of personal-address might require the following in an NXD database to validate valid zip codes similar to relational foreign key validation. Note the use of XQuery to achieve the validation. The third objective is achieved.

Table 5.1 The summary on the objectives and the outcomes

Objective	Outcome
The evaluation the Native XML database performance in a comparison with XML_Enabled database	Determining the points of difference between the structure of XML documents and the structure of the database that contain, understanding negative effects of increasing the number of these data within databases
The enhancement entity relationship (EER) algorithm of the relational schema to improve Insert, Delete, Update and Search XML document	Insert, Delete, Update and Search XML document and compare the performance of XML_Enalbled database and Native XML database, by implementing the same command and control data model. The experimental results show that Native XML database has better performance than XML_Enabled database to storing XML documents rather than reading from disk which is slow. It can be concluded that the Native storage strategy and indexing approach are efficient enough for inserting, deleting, updating and searching in a database.
The validation the algorithm in Native XML databases	It is clear through varied research in the field of apply key constraints to XML documents, it is also still true that implementation of unique key and foreign key constraints is a valuable mechanism to enforcing data integrity.

5.5 Conclusion

In this chapter, the extensive discussion have been presented for achievements of the three objectives. In section 5.2, the objective of the evaluation the Native XML database performance in a comparison with XML_Enabled database was achieved. In section 5.3, the objective of the enhancement entity relationship (EER) algorithm of the relational schema to improve Insert, Delete, Update and Search XML document. In section 5.4, the objective of the validation the algorithm in Native XML databases was achieved. In all, the proposed method has better performance compared with the existing schema. The summary on the objectives and the outcomes in Table 5.1.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

Both the XML_Enabled database and the Native XML database provide good graphical user interfaces. The Native XML database uses a Web-based application, which acts as the centralized database administration software, while the XML_Enabled database is a Windows-based application.

The main aim of this thesis was to evaluate the path expressions in Native XML databases, and then enhance entity relationship (EER) algorithm of the relational schema to improve Insert, Delete, Update and Search XML document (XML files with a large number of elements) in Native XML databases. Firstly, start by measure the size of both databases (XED and NXD). Second, insert XML document one by one. Then, start the functions for Update, Delete and Search for parts of XML document and whole XML document.

In a relational database, data and structures are defined. Columns represent data. Tables and relationships form structure. This can be managed well in searching for data and for database navigation.

XML attributes refer to the data. XML elements and sub elements build the structure. In addition, attributes do not have the concept of ordering.

This is similar to columns in a table. No matter how one changes the position of a column in a table, the data content inside a table does not change. For the first approach, tables and columns are both defined as element types. It may be ambiguous to a parser to decide the role of an element. The flexibility of searching for child elements is less than the attribute approach. This is because an element does have ordering meaning. Hence, it cannot fully represent the location-independence of data from an RDBMS concept.

After analyzing the above results, conclude that XML_Enables database has better performance than Native XML database for small data size of XML documents (<=117 MB). XML_Enabled database performance starts to deteriorate with the large data size (>127 MB), because XML_Enabled database cannot handle the large data size of XML documents as efficiently due to conversion overhead. Whereas, Native XML database has better performance than the XML_Enabled database for handling XML documents with larger data sizes, because Native XML database engine directly accesses XML data without conversion.

XML_Enabled database and Native XML database have steady and almost same performance single record: insert, delete, and update. From 65.8 to 183 MB, though the XML_Enabled database record slightly better performance

The Native XML database shows advantages in handling XML documents for mass record. Whereas it provides better scalability as the database grows. Both inserting and deleting are almost similar results. For mass updates, the Native XML database still has advantages, but the difference is not as obvious as in the previous case. As the XML_Enabled database needs one SQL statement to perform mass updates, the Native XML database achieves this indirectly. We have tried to discover any API of the Native XML database that provides mass update functionality but without success. It seems that update functionality is a weakness for this Native XML database. Instead, we have to retrieve a single document, change it by another API, and then return the results to the database or display them through XSL. This consumes quite a lot of running time.

The Native XML database produced better results in the reporting section, which implies that the Native XML database X-Query has performance gains from query optimization. Most of the figures show that the XML_Enabled database starts better, but

becomes worse as data size grows. The difference becomes obvious as the query becomes more complicated. Query 14 and Query 15 in figure 4.15 and 4.16 show this.

6.2 Research Contributions

The main objective of this research work was to evaluate and optimization of path expressions in Native XML databases, by create a Native XML method that Insert, Delete, Update and Search XML document. The objective has been realized through the following contributions:

(i) Determine the points of difference between the structure of XML documents and the structure of the database that contain, to understand negative effects of increasing the number of these data within databases. These points were determined by added a fixed number of XML documents are assigned to be imported into the databases. The documents are repeated from the smallest to the largest for three times, and the average value is taken.

(ii) Explore the functionality of the databases (Native XML Databases & XML_Enabled Databases) in handling big sizes of XML documents. By evaluate and compare the performance of the data queries in both databases. Then create a Native XML method that Insert, Delete, Update and Search Big sizes of XML document.

6.3 Recommendation of Future Work

Traditional relational databases (XML_Enabled databases) suffer of rigidity, in the sense that suffers in dealing with semi-structured data (XML data). In spite of expressive power XML querying languages, they also present this problem. This study gives a future idea to deal with the problem of giving more flexibility to Native XML database:

(i) They are much simpler to use and integrate into applications who deal predominantly with XML data sources. Even though the scalability, feature sets, and database infrastructure support will need to evolve, they are still

competitive alternatives to RDBMS based solutions and without a doubt, useful solutions for storing XML data. In regard to database support, we found that all major XML-enabled databases supported the DOM, Oracle 10G, IBM DB2 UDB, and MS SQL Server 2005. Each of the navigation and traversal techniques described above are generally supported using separate XML indexes that describe the node stored in the CLOB or LOB. Many NXDs also support the DOM, some of these include eXist, DOMSafeXML, Infonyte, and Ozone .However, with increasing data size, how to offer more scalable, faster implementations. It is also the further research work.

(ii) Improvement of the effectiveness of hybrid and Native XML retrieval system by allows efficiently finding the position of a node given its unique id value in these systems.

(iii) Use an algorithm that optimizes the binary search in the function to traverse a smaller range of tuples each time by investigating the optimal combination of Coherent Retrieval and matching elements in the final answer list.

(iv) ML has a reputation for being big and unwieldy, but the reputation isn't entirely deserved. Many of the size and processing requirements for XML files are the result of inefficient development tools. Since VTD-XML 2.4, XimpleWare introduces an extended version of VTD-XML capable of processing XML documents up to 256 GB in size. In the future work, the research will focus on the large size of dataset in GB/large amount of data, and provide an explanation of the issues involved in file size and execution requirements, and how to streamline those to bring XML in line with other file formats.

REFERENCES

Abiteboul, S. (1996). Querying Semi-Structured Data. ICDT. *Vol (2)*. 1-18.

Abiteboul, S., Buneman, P. and Suciu, D. (1999). Data on the Web: From Relations to Semistructured Data and XML (The Morgan Kaufmann Series in Data Management Systems. Burlington: Morgan Kaufmann.

Akmal B.C., Awais R. and Roberto, Z. (2003). XML Data Management: Native XML and XML_Enabled Database Systems. USA: Addison Wesley.

Algerawy, A., Mesiti, M., Nayak, R. and Saake, G. (2011). XML Data Clustering: An Overview. ACM Computing Surveys, *Vol. 43, No. 4.*

Alpuente, M., Ballis, D., Falaschi, F. and Romero, D. (2013). Rewriting-based repairing strategies for XML repositories. The Journal of Logic and Algebraic Programming. 82: 326-352.

Amjad Q. and Kamsuriah, A. (2014). Model-Mapping Approaches for Storing and Querying XML Documents in Relational Database: A Survey. Journal of Convergence Information Technology(JCIT). 9(2): 148-155.

Arenas, M., Fan, W., and Libkin, L. (2002, September). *What's hard about xml schema constraints?. In International Conference on Database and Expert Systems Applications* (pp. 269-278). Springer Berlin Heidelberg.

Berglund, A., Fernandez, M., Malhotra, A., Marsh, J., Nagy, M. and Walsh, N. (2010). XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). USA: W3C MIT, ERCIM, Keio.

Bishop, A.P., Van House, N. and Buttenfield, B.P. (2003). *Digital Library Use: Social Practice in Design and Evaluation.* USA: Cambridge, MA: The MIT Press.

Bourret, R. (November 2009). XML database products. www.rpbourret.com/xml/XMLAndDatabases.htm.

Bourret, R. (March 2005). Going Native: Making the Case for XML Databases. http://www.xml.com/pub/a/2005/03/30/Native.html.

Bourret, R. (September 2005). XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

Bourret, R. (March 2007). Going Native: Use cases for Native XML databases. http://www.rpbourret.com/xml/UseCases.htm.

Bourret, R. (November 2009). XML and databases (general). http://www.rpbourret.com/xml/index.htm.

Bourret, R. (June 2010). XML Database Products. http://www.rpbourret.com/xml/XMLDatabaseProds.htm.

Bray, T., Paoli, J., Sperberg, C. M., Maler, E. and Yereau, F. (September 2006). Extensible Markup Language (XML) 1.1. http://www.w3.org/TR/xml11/.

Böhme, T., & Rahm, E. (2001). XMach-1: *A benchmark for XML data management. In Datenbanksysteme in Büro, Technik und Wissenschaft* (pp. 264-273). Springer Berlin Heidelberg.

Cahlander, L., Cross, P., Geldiyev, Z., Stallman, A. and Turpin, M. (2010). EXIST-DB OPEN SOURCE XML DATABASE SOFTWARE ARCHITECTURE DESCRIPTION. Introduction to Software Architecture.

Chaudhri, A., Rashid, A. and Zicari, R. (March 2003). XML Data Management: Native XML and XML_Enabled Database Systems. Addison-Wesley Professional; 1 edition.

Chienping, C., Kuenfang, J. and Henghsun, L. (2011). A syntactic approach to twig-query matching on XML streams. The Journal of Systems and Software. 84:993–1007.

Christophides, V., Cluet, S. and Simeon, J. (2002). On wrapping query languages and efficient XML integration. The ACM SIGMOD. pp. 141–152.

Clark, J. (1997). Comparison of SGML and XML World Wide Web Consortium Note. http://www.w3.org/TR/NOTE-sgml-xml-971215. (15 December 1997).

Cugnasco, C., Hernandez, R., Becerra, Y., Torres, J. and Ayguad, E. (2013). Aeneas: a tool to enable applications to effectively use non-relational databases. J. Procedia Computer Science. 18: 2561 – 2564.

Damiani, E. and Tanca, L. (2000). Flexible Query Techniques for Well-formed XML Documents. Fourth International Conference on knowledge Based Intelligent Enginem'ng Systems &Allied Technologies, Vol (2), 708 – 711.

DeveloperWorks, Processing WSDL in Python. http://eduunix.ccut.edu.cn/index2/pdf/python_wdsl.pdf.

Deutsch, A., Fernandez, M. F., and D. Suciu. (1999). Storing semistructured data with STORED. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 431–442, Philadelphia, June.

Duka, A.V. (2010). Modelling of an Electromagnetic Levitation System using a Neural Network . Automation Quality and Testing Robotics (AQTR), IEEE International Conference, Volume: 3 ,pp.1 – 6.

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

Egbert, M. (2007). Introduction to XML Database Technologies. EPCC. 1-34.

El-Sayed, M., Dimitrova, K. and Rundensteiner, E. (2005). Efficiently supporting order in XML query processing. Data & Knowledge Engineering, 54: 355–390.

Erwig, M. (2003). Xing: avisual XML query language. Journal of Visual Languages and Computing. 14: 5–45.

Fayolle, A. (2002). XML and Python Tutorial. euro python conference. http://download.logilab.org/pub/talks/XMLTutorial.pdf.

Fernando, C. M. (2012). HandSpy: managing experiments on writing studies. Master. Thesis. Portugal: Universidade of Porto.

Franceschet, M. (2005, August). XPathMark: an XPath benchmark for the XMark generated data. In International XML Database Symposium (pp. 129-143). Springer Berlin Heidelberg.

Frasincar, F., Houben, G. and Pau, C. (2002). XAL: an algebra for XML query optimization. The 13th Australasian Database Conference. pp. 49–56.

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

Gabriel, R.U. and Mycroft, A. (2013). Source-Code Queries with Graph Databases—with Application to Programming Language Usage and Evolution. Science of Computer Programming. 1-9.

Gerrit, H. (2001).Solving reusability problems of online learning materials. Journal of Campus-Wide Information Systems:18(4):146-152.

Glas, W. (2002). *Xml And Databases*. Master. Thesis. University of Liverpool, UK.

Google Developers Group. (October 2013). Query Language Reference (Version 0.7). https://developers.google.com/chart/interactive/docs/querylanguage .

Zhang, H. and Tompa, F. W. (2004) "Querying XML Documents by Dynamic Shredding," in DocEng'04, Milwaukee: Wisconsin: USA.

Haiwei, Z. and Xiaojie, Y. (2009). Schemas Extraction for XML Documents by XML Element Sequence Patterns. The 1st International Conference on Information Science and Engineering (ICISE). 5096- 5099.

Harold, E. R. (2003). Effective XML: 50 Specific Ways to Improve Your XML. USA: Addison- Wesley.

Harrusi, S., Averbuch, A. and Yehudai, A. (2006). XML Syntax Conscious Compression. Data Compression Conference, pp. 10.

Haw, S. and Lee, C. (2011). Data Storage Practices and Query Processing in XML Databases: A Survey. J. Knowledge-Based Systems. 24: 1317–1340.

Henk, J. G. (2006). Native XML databases. 5th Twente Student Conference on IT, Pp. 765-771.

Hiddink,G. (2001). ADILE: Architecture of a Database-Supported Learning Environment. University of Twente, Netherlands JILR Volume 12, Number 2; ISSN 1093-023X.

Houman, M. K. (2004). Performance Analysis of Xquery vs. SQL. Master. Thesis. University of Wisconsin Platteville, USA. https://pypi.python.org/pypi/4Suite/1.0b1.

Hunter, D., Cagle, K., Chris, D., Kovack, R., Pinnock, J. and Rafter. (2001). Beginning XML, 2nd Edition. UK: Wrox.

Ishikawa, H., Kubota, K., Kanemasa, Y. and Noguchi, Y. (2007). The Design of a Query Language for XML Data. http://ieeexplore.ieee.org.ezproxy.ump.edu.my/stamp/stamp.jsp?tp=&arnumber=795304.

Jagadish, H., Al-Khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos, S., M.Patel, J., Srivastava, D., Wiwatwattana, N., Wu. Y. and Yu, C. http://www.eecs.umich.edu/db/timber/files/timber.pdf.

Jeffrey, D. U. and Jennifer, W. (1997). First Course in Database Systems. USA: Prentice Hall Science and Math.

Jelena Mamčenko. (2004). INFORMATION RESOURCES, Introduction to Data Modeling and MSAccess Code. Lecture Notes. Vilnius Gediminas Technical University: FMITB0.

Jiang, H. and Yang, Q. (2011). A Keyword-based Query Solution for Native XML Database. IEEE.

Jijun, W. and Shan, W. (2005). SEEKER: Relational database information retrieval based on keyword. J. Software. vol. 16, No. 7.

Jing Z., Bo, L. and Yawei, D. (2011). An XML Data Placement Strategy for Distributed XML Storage and Parallel Query. 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp 433-439.

Jingqiang,H., Dongqing, Y., Tengjiao, W. and Shengyue, J. (2004). XML full-text search design and implementation in relational database CoDB. J. Computer Research and Development. vol. 41 Supplement.

Jinsha, Y., Xinye, L. and Lina, M. (2008). An Improved XML Document Clustering Using Path Feature.Fifth International Conference on Fuzzy Systems and Knowledge Discovery, Vol (2), pp.400-404.

Jonathan, Robie. (2003). SQL/XML, XQuery, and Native XML Programming Languages.

Kalinin, A. (2009). Data Guide-based Distribution for XML Documents. Colloquium on Database and Information Systems, Saint-Petersburg, Russia. http://cs.brown.edu/~akalinin/papers/dataguide-xml.pdf.

Ki Min, J., Hee Lee, C. and Wan Chung, C. (2008). XTRON: An XML data management system using relational databases. J. Information and Software Technology. 50: 462–479.

Kolar, P. and Loupal, P. (2006). Comparison of Native XML Databases and Experimenting with INEX. V. Sn´aˇsel, K. Richta, J. Pokorn´y (Eds.). pp. 116–119.

Leigh, D. (October 2001). XML and Databases? Follow Your Nose. J. XML- Deviant. http://www.xml.com/pub/a/2001/10/24/follow-yr-nose.html?page=1.

Leonardi, E., Hoai, T., Bhowmick, S. and Madria, S. (2007). DTD-DIFF: A change detection algorithm for DTDs. J Data & Knowledge Engineering 61: 384–402.

Lin, X., Wang, N., De Xu. And Zeng, X. (2010). A novel XML keyword query approach using entity subtree. The Journal of Systems and Software. 83: 990–1003.

Lin, X., Wang, N., Zeng, X. and Sun, Y. (2013). XML normalization based on entity segments. J. Information Sciences. 239: 85-95.

Liu, R. (2004). Xindice: XML Database. Independent Research: UW Messenger. http://depts.washington.edu/dslab/reports/ryan_wi04.pdf.

Lohrey, M., Maneth, S. and Mennicke, R. (2013). XML tree structure compression using RePair. J. Information Systems. 38: 1150-1167.

Mabanza, N. (2010). Analyzing the Impact of XML Storage Mode is on the Performance of Native XML Database Systems – A Case Study. Seventh International Conference on Information Technology.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J. and Lore. (1997). A database management system for semistructured data, ACM Sigmod Record. 26: 54–66.

Melton, J. and Buxton, S. (2006). Querying XML XQuery, XPath, and SQL/XML in context. USA: Morgan Kaufmann Publisher.

Nicola, M. and Kumar, P.C. (2010). DB2 pure XML cookbook: Master the power of the IBM Hybrid Data Server. USA: IBM Press.

Papamarkos, G., Zamboulis, L. and Poulovassilis, A. (2011). XML Databases. Report. UK: University of London.

Pardede, E., Rahayu, J.W. and Tania, D. (2008). XML data update management in XML_Enabled database. Journal of Computer and System Sciences 74:170–195.

Pavlovic, G. L. (2007). Native Xml Databases vs. Relational Databases in Dealing with XML Documents. Kragujevac J. Math.181-199.

Pokorny. J. (2008). XML Databases: Principles and Usage. Information systems development series. Pp. 37 -38 Proceedings by deepX Ltd., pp. 1-18.

Schmidt, A., Waas, F., Kersten, M., Carey, M. J., Manolescu, I., & Busse, R. (2002, August). XMark: A benchmark for XML data management. In Proceedings of the 28th international conference on Very Large Data Bases (pp. 974-985). VLDB Endowment.

Schmidt, A., & Wass, F. (2010, April). Kersten," XMark: a benchmark for xml data management. In Proceedings of IEEE international conference on Advanced Information Networking and Applications (pp. 1012-1019).

Schroeder, R., & Hara, C. S. (2014). Towards Full-fledged XML Fragmentation for Transactional Distributed Databases.

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

Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, E., Naughton, J. and Tatarinov, I. (2011). A General Technique for Querying XML Documents using a Relational Database System. SIGMOD Record, Vol.30, No.3.

Shipman, J. (2007). Python and the XML document Object Oriented Model (DOM) with 4Suite. New Mexico Tech.

Sipani, S., Verma, K., John A. M. and Aleman, M. B. (2004). Designing a high-performance database engine for the Db4XML Native XML database system. The Journal of Systems and Software. 69: 87–104.

Stayton, B. (2007). DocBook XSL: The Complete Guide Fourth Edition. USA: Sagehill Enterprises.

The Apache XML Project Group. (August 2011). Apache Xindice. http://xml.apache.org/xindice/.

Thomas, S. W., Snodgrass, R. T., & Zhang, R. (2014). Benchmark frameworks and τBench. Software: Practice and Experience, 44(9), 1047-1075.

Tzvetkov, V. and Xiong, W. (2005). DBXML - Connecting XML with Relational Databases.Proceedings of the 2005: The Fifth International Conference on Comp uter and Information Technology (CIT'05).

Vaidya, P. and Plale, B. Benchmark Evaluation of Xindice as a Grid Information Server. http://www.cs.indiana.edu/pub/techreports/TR585.pdf.

Vavliakis, K., Grollios, T. and Mitkas, P. (2013). RDOTE– Publishing Relational Databases into the Semantic Web. The Journal of Systems and Software. 86: 89–99.

Wang, J., Horng, J., Liu, B. and Fan, K. (1996). A Genetic Algorithm for Structural Query Processing in Hypertext Systems. IEEE. 506-511.

Williams, K, Brundage, M., Dengler, P., Gabriel, J., Hoskinson, A., Kay, M., Maxwell, T., Ochoa, M., Papa, and J., Vanmane, M., (2000).Professional XML Databases, Wprox Press Limited, Page(s) 47-64

W3C. 2011Timeline of the W3C technologies related to the XML. News Archive.

Xiaomei, Y. and Heng, D. (2010). Native XML Database Design and Realization based on MDA. IEEE, pp. 1-4.

Xiaomin, W., Yanlin, S. (2005). XQuery full-text search. Computer Engineering and Applications.

Xin, Y., He, Z. and Cao, J. (2010). Effective pruning for XML structural match queries. J. Data & Knowledge Engineering. 69: 640–659.

Xinping, G., Cordy, J.R. and Dean, T.R. (2003). Unique Renaming of Java Using Source Transformation. The Third IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'03). 151 – 160.

XML News. (October 1999). XML basics. http://www.xmlnews.org/docs/xml-basics.html..

Xu, Y., Guan, J., Li, F. and Zhou, S. 2013. Scalable continual top-k keyword search in relational databases. J. Data & Knowledge Engineering. 86:206-223.

Yan, L. and Ma, Z.M. (2013). Formal Translation from Fuzzy EER Model to Fuzzy XML Model. J. Expert Systems with Applications.

Yang, Y., Hai-ge, L. and Xin, H. (2010). Querying XML Data Based on Improved Prefix
Encoding. International Conference on Computer Application and System Mode ling, vol 10. Pp. 521-525.

Yao, B. B., Ozsu, M. T., & Khandelwal, N. (2004, March). XBench benchmark and performance testing of XML DBMSs. In Data Engineering, 2004. Proceedings. 20th International Conference on (pp. 621-632). IEEE.

Yipeng, J., Yan, C. and Yan, C. (2009). Study on XML Traffic Information Basic Data Effectiveness Validation. The 1st International Conference on Information Science and Engineering (ICISE). 5166-5168.

Ykhlef, M. and Alqahtani, S. (2007). A survey of graphical query languages for XML data. Journal of King Saud University – Computer and Information Sciences. 23: 59–70.

Yu, Y. (2005). Benchmarking of Native XML Database Systems. Master. Thesis. University of Wollongong, Australia.

Zhongyi, W. (2009). Full-text search method and prototype implementation based on XML. Huazhong Normal University.

Zhou, C., Liu, Z. and Gao, L.(2009). Declaration of RoHS Compliance based on Smart Document and XML Database. The Ninth International Conference on Electronic Measurement & Instruments (ICEMI). 4-1063.

APPENDIX A

LIST OF PUBLICATIONS

H.1: HOW DOSE XML LANGUAGE HELP THE DIGITIL LIBRARIES- The National Conference on Postgraduate Research 2012 (NCON-PGR), 8th-9th September 2012, Universiti Malaysia Pahang, organized by UMP Post Graduate Office and Jabatan Hal Ehwal Akademik dan Antarabangsa (JHEAA)

H.2: GOING NATIVE: Native XML Database vs. Relational Database - International Conference on Computational Science and Information Management (ICoCSIM), 3rd-5th December 2012, Toba Lake, Indonesia.

H.3: Going Native: Native XML database vs. XML_Enabled database - International Conference on Software Engineering & Computer Systems (ICSECS - 2013), 20-22 August 2013, Organized by Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Gambang, Pahang, Malaysia.

H.4: Going Native: Indexing architecture of eXist-db_ An Open Source Native XML database system - IEEE TENCON 2013, 22nd - 25th October, 2013, The IEEE Region 10 Conference, Xi'an, China.

H.5: PERFORMANCE ANALYSIS OF XML_Enabled database VS. Native XML database - The CREATION, INNOVATION TECHNOLOGY & RESEARCH EXPOSITION (CITREx 2014), 5th-6th March 2014, University Malaysia Pahang, Malaysia.

APPENDIX B

Native XML Database (eXist db)