A Comparison of Open Source Native XML Database Products

A thesis submitted in fulfilment of

the requirements for the

degree of

MASTER OF SCIENCE

In the Department of

Computer Science

University of Fort Hare

Alice

South Africa

By

Ntima Mabanza

December 2005

KEYWORDS

Collection, documents, text-based, model-based, generated XML documents, Xindice, eXist, dbXML, Berkeley DB XML, Open Source Native XML databases, popularity, benchmark techniques, metric performance comparison, response time, check point, basic tasks, storage, queries, modifications, deletions.

University of Fort Hare
Together in Excellence

ABSTRACT

Since the advent of XML technology, one of the areas where it has made its mark is in data interchange between applications or businesses. As a result, there has been an increase in the number of XML documents generated from business transactions. Consequently, there is considerable work being done in the database research community with the aim to develop software to effectively process and manage XML documents. Open Source Native XML database systems are one example of a new generation of database technologies developed to address this issue. The decision to adopt these technologies can be based on several factors depending on the user's needs. Among these factors are functionality, market share, support, maintenance, performance, scalability, usability, security, and flexibility.

This research focuses on one of these key factors, namely metric performance when executing basic database tasks (i.e. storage, [redacted] modification, and delete) using several Open Source Native XML databases i.e. BerkeleyDB XML, dbXML, eXist, and Xindice. Each product is deployed and tested using benchmark techniques. A number of interesting insights are gained from the experiments and a summary of the strengths and weaknesses of each of these products is presented.

ACKNOWLEDGEMENTS

I glorify GOD Almighty for giving me strength, courage, and wisdom to carrying out this work. This would not have been possible without His Grace and His divine guidance.

I would like to express my deepest appreciation to my supervisor, Prof. J. Chadwick for the privilege of working under his supervision. Your unique perspectives, continual support throughout this project, your positive feedback and insightful discussions have kept the excitement alive. Without your invaluable assistance this thesis would not have been possible. You have played a vital role in my growth and maturity as an academic researcher.

To my parents (Mr. & Mrs. Mabanza), brothers (Zouka, Jonas, and Moni), sisters (Theresse, Niclette, and Sarah), whose pride and joy, it is to support me in whatever I do. Thank you for always believing in me. You have made me strive to be a better person in life.

Thanks to Mrs. Formson for taking time to proof read this thesis. It was much appreciated.

To my big brother Taluanga Matiki and his family thanks for all you have done for me. Without your support big brother I would not have reached this far.

To my classmates "Heita Guys" thanks for your consistent encouragement and your friendship.

I am very grateful to the support that the Center of Excellence (CoE) and Computer Science staff have given me during this project, especially Prof Muyingi, and Dr. Terzoli.

My thanks also go to the Open Source Projects (Apache Software Foundation, dbXML Group, Sleepycat Software, and Wolfgang Meier) for making their products freely available.

Last but far not least, a hearty thank to my Sweetheart, Sylvie Mabanza, my comrade B Ingila, my cousins A Muyingi, R Bubu, my best friend Pastor A Mputu for their undying support and encouragement.

--------- Ntima Mabanza ----------

DEDICATION

To my beautiful Wife, Mrs. M K Sylvie Mabanza for her support, patience, encouragement, endless love, and understanding.

To my Mother, Mrs. M M Yvonne Mabanza for her infinite affection, very positive thoughts about my life.

To my Uncle Mundundu Ndonambah, who is a role model for our generation.

TABLE OF CONTENTS

CHAPTER 1 ..... **1**

INTRODUCTION **1**

1.1. INTRODUCTION AND MOTIVATION 2

1.2. RELATED WORK 3

1.3. SCOPE OF THE THESIS 4

1.4. RESEARCH METHODOLOGY 5

1.5. ORGANIZATION OF THE THESIS 7

CHAPTER 2 ..... **9**

OVERVIEW OF XML DATABASES AND XML-RELATED TECHNOLOGIES **9**

2.1. INTRODUCTION 10

2.2. XML DATABASES 10

2.2.1. What is an XML Database? *10*

2.2.2. Why XML Databases? *12*

2.2.3. Types of XML Databases *14*

2.3. INTRODUCTION TO NATIVE XML DATABASES 15

2.3.1. Introduction *15*

2.3.2. Why do Native XML Databases exist? *16*

2.3.3. Storage in Native XML Databases *17*

2.3.4. Features of Native XML Databases *19*

2.4. XML-RELATED TECHNOLOGIES FOR NATIVE XML DATABASES 21

2.4.1. Introduction *21*

2.4.2. XML:DB API *22*

2.4.3. XML Parser *24*

2.4.3.1. Document Object Model (DOM) API 26

2.4.3.2. Simple API for XML (SAX) API 29

2.4.3.3. DOM vs. SAX 31

2.4.4. XPath *33*

2.4.5. XUpdate *37*

2.5. OPEN SOURCE NATIVE XML DATABASES 40

2.6. SUMMARY 41

CHAPTER 3 ..... **42**

EVALUATION CRITERIA AND SELECTION OF PRODUCTS **42**

3.1. INTRODUCTION 43

3.2. EVALUATION CRITERIA 44

3.2.1. Functionality 44
3.2.2. Market Share 45
3.2.3. Support 45
3.2.3.1. Documentation 46
3.2.3.2. Installation 47
3.2.3.3. Mailing List 48
3.2.4. Maintenance 48
3.2.5. Performance 49
3.2.6. Scalability 49
3.2.7. Usability 49
3.2.8. Security 50
3.2.9. Flexibility 50
3.3. OPEN SOURCE NATIVE XML DATABASE PRODUCTS 50
3.3.1. Introduction 50
3.3.1.1. Selection Criteria 51
3.3.2. Previous studies using the selected Open Source Native XML databases 52
3.3.3. Xindice 56
3.3.3.1. Overview and Features 56
3.3.3.2. Functionality 57
3.3.3.3. Market Share 57
3.3.3.4. Support 58
3.3.3.4.1. Documentation 58
3.3.3.4.2. Installation 58
3.3.3.4.3. Mailing List 59
3.3.3.5. Maintenance 59
3.3.3.6. Scalability 59
3.3.3.7. Usability 60
3.3.3.8. Security 60
3.3.3.9. Flexibility 60
3.3.4. eXist 60
3.3.4.1. Overview and Features 60
3.3.4.2. Functionality 62
3.3.4.3. Market Share 62
3.3.4.4. Support 62
3.3.4.4.1. Documentation 62
3.3.4.4.2. Installation 63
3.3.4.4.3. Mailing List 63
3.3.4.5. Maintenance 63
3.3.4.6. Scalability 63
3.3.4.7. Usability 64
3.3.4.8. Security 64
3.3.4.9. Flexibility 64
3.3.5. dbXML 65
3.3.5.1. Overview and Features 65
3.3.5.2. Functionality 66
3.3.5.3. Market Share 66
3.3.5.4. Support 67
3.3.5.4.1. Documentation 67
3.3.5.4.2. Installation 67
3.3.5.4.3. Mailing List 67
3.3.5.5. Maintenance 68
3.3.5.6. Scalability 68
3.3.5.7. Usability 68
3.3.5.8. Security 69
3.3.5.9. Flexibility 69
3.3.6. Berkeley DB XML 69
3.3.6.1. Overview and Features 69
3.3.6.2. Functionality 70
3.3.6.3. Market Share 71

3.3.6.4. Support 71
3.3.6.4.1. Documentation 71
3.3.6.4.2. Installation 71
3.3.6.4.3. Mailing List 72
3.3.6.5. Maintenance 72
3.3.6.6. Scalability 72
3.3.6.7. Usability 72
3.3.6.8. Security 73
3.3.6.9. Flexibility 73
3.3.7. Comparison 73
3.4. SUMMARY 77

CHAPTER 4 79

EXPERIMENTAL EVALUATION OF THE CHOSEN OPEN SOURCE NATIVE XML DATABASE PRODUCTS 79

4.1. INTRODUCTION 80
4.2. MOTIVATION 80
4.3. AIM 81
4.4. HYPOTHESIS 81
4.5. METHODOLOGY AND TEST SUITE SET-UP IN 82
4.5.1. Timing Methodology 82
4.5.2. Experimental set-up 84
4.5.2.1. Hardware Environment 84
4.5.2.2. Software Environment 85
4.5.2.3. Rationale for Choosing to use a single machine 87
4.5.3. Description of the tests 87
4.5.3.1. Storage 88
4.5.3.2. Queries 89
4.5.3.3. Modifications 92
4.5.3.4. Deletions 94
4.6. SUMMARY 96

CHAPTER 5 97

OUTCOMES AND DISCUSSION OF THE EXPERIMENTS 97

5.1. INTRODUCTION 98
5.2. PREVIOUS RESEARCH FINDINGS 98
5.3. EXPERIMENTAL RESULTS 99
5.3.1. Storage 100
5.3.2. Test results for storage 104
5.3.2.1. The storage for fifty part elements 104
5.3.2.2. The storage for hundred and fifty part elements 107
5.3.2.3. The storage for thousand part elements 108
5.3.3. Queries 110
5.3.4. Test results for queries 113
5.3.4.1. The query /parts/part 114
5.3.4.2. The query /parts/part[75]/desc 118
5.3.4.3. The query /parts/part[@sku='7199']/desc 120
5.3.4.4. Other queries 122

5.3.5. Modifications *124*
5.3.6. Tests results for modification *126*
5.3.6.1. The modification 127
+ "<xu:update select="/parts/part[75]/price">" + "$700.00" + "</xu:update>" 127
5.3.6.2. The modification 129
+"<xu:update select="/parts/part/@sku[.='7199']"> + "509" + "</xu:update>" 129
5.3.6.3. Other modifications 131
5.3.7. Deletions *132*
5.3.8. Tests results for deletion *134*
5.3.8.1. The deletion "<xu:remove select="/parts/part[75]/price"/>" 135
5.3.8.2. The deletion "<xu:remove select="/parts/part/@sku[.='509']"/>" 136
5.3.8.3. Other deletions 138
5.4. COMPARISON OF FINDINGS 139
5.5. SUMMARY 140

CHAPTER 6 142

CONCLUSION AND FUTURE WORK 142

6.1. INTRODUCTION 143
6.2. THESIS CONTRIBUTION 143
6.3. WORK COVERED 143
6.3.1. Overall impression about Open Source Products *146*
6.4. LIMITATIONS OF THIS PROJECT 147
6.5. FUTURE WORK 148
6.6. CONCLUSION 149

REFERENCES 151

APPENDICES 159

APPENDIX A 160
APPENDIX B 161
B.1. Java Code for generating part elements *161*
B.2. Databases Code used for storage operation *162*
B.2.1. Berkeley Database 162
B.2.2. dbXML database 164
B.2.3. eXist database 167
B.2.4. Xindice database 170
B.3. List of query operations considered in the performance test *173*
B.3.1. Berkeley DBXML 173
B.4. Database Code used for query operation *174*
B.4.1. Berkeley DBXML 174
B.5. List of query operations considered in the performance test *176*
B.5.1. dbXML, eXist, and Xindice 176
B.6. Database Codes used for query operation *177*
B.6.1. dbXML database 177
B.6.2. eXist database 180
B.6.3. Xindice database 182
B.7. Modification operations considered in the performance test *184*
B.7.1. dbXML, eXist, and Xindice 184
B.8. Delete operations considered in the performance test *185*
B.8.1. dbXML, eXist, and Xindice 185

B.9. Database Codes used for modification and delete operation *186*
B.9.1. dbXML database 186
B.9.2. eXist database 189
B.9.3. Xindice database 191
APPENDIX C 193

C.1. Timing Results for storage operations *193*
C.1.1. Berkeley DBXML 193
C.1.2. dbXML 193
C.1.3. eXist 194
C.1.4. Xindice 194
C.2. Timing Results for Query operations *195*
C.2.1. Berkeley DBXML 195
C.2.2. dbXML 195
C.2.3. eXist 195
C.2.4. Xindice 196
C.3. Timing Results for the Query re-run operations *197*
C.3.1. eXist 197
C.3.2. Xindice 197
C.4. Timing Results for Modification operations *198*
C.4.1. dbXML 198
C.4.2. eXist 198
C.4.3. Xindice 198
C.5. Timing Results for Delete operations *199*
C.5.1. dbXML 199
C.5.2. eXist 199
C.5.3. Xindice 199
C.6. Timing Results for the Modification re-run *200*
C.6.1. eXist 200
C.6.2. Xindice 200
C.7. Timing Results for the Delete re-run *201*
C.7.1. eXist 201
C.7.2. Xindice 201

LIST OF FIGURES

Figure 2.1: XML parsing process diagram 25
Figure 2.2: Simple XML document.....27
Figure 2.3: DOM tree of simple XML document.....27
Figure 2.4: SAX events generated in simple XML document.....30
Figure 2.5: An Xupdate rename.....39
Figure 4.1: Response Time.....83
Figure 5.1: Sample section of generated XML document.....101
Figure 5.2: Text-based storage representation of generated XML documents.....102
Figure 5.3: Model-based storage representation of generated XML documents.....103
Figure 5.4: Timing comparison for storing generated XML document containing.....105
fifty part elements.....105
Figure 5.5: Timing comparison for storing generated XML document containing.....107
hundred and fifty part elements.....107
Figure 5.6: Timing comparison for storing generated XML document containing.....109
thousand part elements.....109
Figure 5.7: XPath tree for the stored generated XML documents.....111
Figure 5.8: Comparison of time taken for executing query Q1b.....115
Figure 5.9: Comparison of time taken for executing query Q(3b).....119
Figure 5.10: Comparison of time taken for executing query Q3e.....121
Figure 5.11: an XUpdate update.....125
Figure 5.12: Comparison of time taken for executing modification (M1b).....128
Figure 5.13: Comparison of time taken for executing modification (M1e).....130
Figure 5.14: an XUpdate delete.....133
Figure 5.15: comparison of time taken for executing delete (D1b).....135
Figure 5.16: comparison of time taken for executing delete (D2b).....137

LIST OF TABLES

Table 2.1: DOM classes and interfaces.....28
Table 2.2: Methods invoked by the SAX parser.....30
Table 2.3: SAX and DOM comparison.....32
Table 2.4: Some location-path abbreviations.....34
Table 2.5: XPath example queries.....36
Table 2.6: Various XML Technologies used.....39
Table 3.1: key features and platforms comparison.....74
Table 3.2: attributes comparison.....160

Listing 4.1: Fragment of XML document used.....86

GLOSSARY OF ACRONYMS

API	Application Programming Interface
BLOB	Binary Large Object
CDATA	Character DATA
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DBMS	Database Management System
DOM	Document Object Model
DOM4J	Document Object Model for Java
DTD	Data Types Definition
GB	Gigabyte
GHz	Gigahertz
GUI	Graphical User Interface
HTML	HyperText Markup Langua
HTTP	HyperText Transport Protocol
IBM	International Business Machines
J2SDK	Java 2 Software Development Kit
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JDBC	Java Database Connectivity
JDOM	Java Document Object Model
JVM	Java Virtual Machine
MB	MegaByte
NXD	Native XML Database
ODBC	Open Database Connectivity
OS	Operating System
OSS/FS	Open Source Software/Free Software
PCDATA	Parsable Character DATA
PDF	Portable Document Format

PHP	Hypertext Preprocessor
RAM	Random Access Memory
REST	Representational State Transfer
RTF	Rich Text Format
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
W3C	World Wide Web Consortium
WebDAV	Web-based Distributed Authoring and Versioning
XEDB	XML-Enabled Databases
XHTML	Extensible HyperText Markup Language
XIS	Extensible Information Server
XML	Extensible Markup Language
XML-RPC	XML Remote Procedure Call
XPath	XML Path Language
XSD	XML Schema Definition
XSL	Extensible StyleSheet Language
XSLT	Extensible StyleSheet Language Transformation
XUpdate	XML Update Language

Chapter 1

INTRODUCTION

This introductory chapter discusses the motivation for studying XML databases, including some questions that provide direction for the work. It also briefly introduces previous research, the scope of the thesis and also the research methodology. Chapter one concludes by giving the organization of the thesis.

1.1. Introduction and Motivation

Since XML [1] technology was launched on the world in 1998, it has, without doubt, made a serious impact on web publishing [2]. One of the areas where XML has made its mark is in data interchange between applications or businesses, especially across the Internet. As a result there has been an increase in the number of XML documents generated from business transactions. On the other hand, the XML revolution has also increased the need in the research community to find an effective approach for storing and processing XML documents. With considerable work being done by various research communities in an effort to address this issue, a new kind of database technology has emerged. A Native XML database system is a typical example of one of these new generations of technologies.

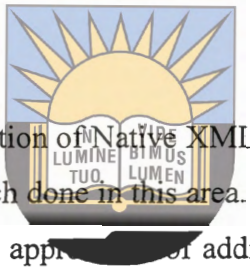
A great deal of comment, publicity, and media hype has been, and is still being, published on the Internet and also in journals about Native XML database systems. However, what has been said about these XML databases does not necessarily guarantee by any means that these products will be widely accepted. Like most other technologies, native XML databases face challenges. There are some factors one might want to consider before deciding to select a particular technology. One of the most important factors is performance: end users expect the database to handle their tasks as quickly and efficiently as possible.

The arrival of Native XML database systems generated, many questions including:

- Why another kind of database technology?
- What Native XML database systems are available on the market?
- What differentiates them from other databases?
- How do these products perform when handling different database tasks?

With these questions in mind, this research was initiated in order to explore some Native XML database systems. The project focused on the performance issues of Native XML database systems.

1.2. Related Work



The performance comparison evaluation of Native XML database systems is still an open issue. There has been a lot of research done in this area. As a result, a number of research groups have come up with different approaches of addressing this matter. For example, some research groups have proposed Application Benchmark techniques as possible tools for comparing the query processing performance of Native XML database systems. These include X007 [3, 4, 5, 6] which consists of eighteen queries, XMark [7] consists also of twenty queries, XMach-1 [8, 9] consists of eight queries, and XBench [10, 11, 12, 13] consists of twenty queries. XMach-1 [8, 9] is the only one which contains queries and update operations. It contains three simple update operations. Although, it contains queries and update operations, it is still not complete because it lacks other databases operations.

Additional to the proposed benchmarks, other research groups [14, 15, 16] concentrate their efforts on doing performance comparison studies of XML-enabled and Native XML database systems.

As indicated in the previous paragraphs, the performance comparison of XML-enabled database and Native XML database systems was the foremost focus for most of the previous studies done. As a result, there are few studies that attempt to compare the Open Source Native XML database systems against each other.

So far various studies comparing the Open Source Native XML database systems can be separated into two groups. These two groups are (1) studies comparing the features of Open Source Native XML database systems, and (2) studies comparing the capacities of some Open Source Native XML database systems for handling common database operations. The scope of this thesis will focus on the previous studies comparing the capacities of Open Source Native XML database systems in handling common database operations. Some research groups like [17] and [18] have conducted some performance comparison studies using some of the Open Source Native XML database systems that have been investigated in this project. This project, does not intend to use any of the previous studies' results (e.g. [17, 18]). It conducts its own performance comparison tests and compares its results with those obtained by the previous studies. Furthermore, this will help in examining different factors that affect the performance of Open Source Native XML databases with the view of exploring the capacities, strengths, weaknesses, and limitations of Open Source Native XML databases.

1.3. Scope of the thesis

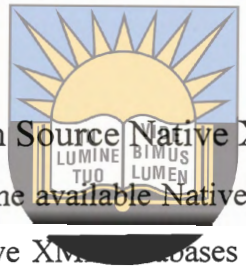
Databases are the prime storage engines for many types of data. Furthermore, there are four basic processing functions for a database application: create, read, update, and delete. These functions are referred to by the (unfortunate) acronym CRUD [19].

The goal of this project is not to build a new Native XML database system or to improve on any of the existing ones, but rather to investigate and compare the performance of some chosen Native XML database systems in handling the four basic database processing functions mentioned in the previous section. The focus will be on the time

taken by each of these chosen Native XML databases in performing a basic database function. There is also an attempt to understand how these chosen Native XML databases systems work when carrying out their different database processing functions and gain some insight into the different related technologies that they use. This will clarify and identify different factors that can influence their performance.

1.4. Research Methodology

The following factors were considered in conducting this study and in answering some of the questions listed in section 1.1:



- Selection of the Open Source Native XML databases

A look at [20], reveals a list of some available Native XML databases. A quick survey, showed that there are several Native XML databases that have been developed in both industry and academia. These products can be categorized into two groups: commercial and open source products. A choice was made to use Open Source native XML database products. The next step involved selecting the specific open source candidates to be used in this study. Comprehensive documentation, popularity, support in the form of newsgroups, and the Java programming language were chosen as the criteria for selection. Based on these criteria, the following were selected:

- Xindice [21] version 1.1b4, an open source native XML database by the Apache Software Foundation
- eXist[22] version 0.9.2, an open source native XML database founded by Wolfgang Meier at the Technical University of Darmstadt, Germany
- dbXML[23] version 2.0, an open source native XML database mainly developed by dbXML Group
- Berkeley DB XML [24] version1.1.0-Win32, an open source native XML database by SleepyCat Software

Each of the indicated versions of these selected candidates was available as a free download. At the time these products were downloaded, the versions stated in the previous section were the latest versions from their respective open source projects. At the time of completion of the project some of these products were already on a different version.

- Setting up the Native XML databases

To conduct this study, an experimental test environment was set up for each of these chosen products. They were all installed in a Windows 2000 Professional machine with 512 RAM, 40 GB hard disk and 1.80 GHz of CPU.

- Experimental data sets

A program was written that generated the data sets that were used in the experiments. The full code for this program can be found in Appendix B. In addition, test hypotheses for the purpose of these experiments were also formulated.

- Experimental results obtained

Graphs were used to display and compare the different results obtained from various conducted tests. Each of the results displayed in graphs was discussed and a short conclusion was drawn based on the results obtained with regards to the products' behavior on that specific operation.

1.5. Organization of the thesis

Chapter 2 introduces XML databases and lays the foundation for the rest of the thesis. It gives an overview of XML databases, and also explains the reason for XML databases' existence, their benefits, and the different type of XML databases available. The chapter focuses on Native XML databases, their features as well their storage techniques. It furthermore explains the various technologies and mechanisms used by Native XML databases for managing their database content and for performing various database operations.

Chapter3 discusses the evaluation methodology used in this thesis. It also introduces some criteria that were used for selecting the studied Open Source Native XML databases. Some of the previous related databases performance studies done with the selected databases are highlighted. After that, the selected Open Source Native XML databases for this project are also briefly introduced and their basic features are discussed as well. The chapter ends by comparing some of the key features and attributes of these selected Open Source databases.

Chapter 4 explains the motivation, aim, and hypotheses being tested by the experiments. It furthermore describes and motivates the test environment. Methodologies used to conduct the performance comparisons tests are discussed as well. Chapter 4 also provides detailed descriptions of each of the performance test operations conducted.

Chapter 5 introduces some findings from the previous related works. It also presents and discusses the response time results obtained from each of the experimental performance tests conducted using Open Source Native XML databases as described in chapter 4. These response times are shown in the graphs and they are analyzed and explained with respect to each database operation. Lastly, in this chapter, a comparison of findings from this project and the previous related studies will be done.

Chapter 6 is a concluding chapter. It summarizes the whole thesis, highlighting both the positive and negative findings. It also suggests future research directions.

Chapter 2

OVERVIEW OF XML DATABASES AND XML-RELATED TECHNOLOGIES

2.1. Introduction

This chapter introduces XML Databases. An XML Database is a specialized kind of database, designed to store XML documents. The chapter defines XML Databases and provides an overview of XML Databases. It discusses the reason for XML Databases' existence, the benefits of using such technology, the types of XML Database available, and focuses on Native XML Databases (NXDs). The features of NXDs, and their storage techniques, are discussed as well.

This chapter also further examines the different XML-related technologies used by Native XML Databases for performing their basic database tasks, including different techniques for reading, querying and updating the contents of XML documents stored in a Native XML Database. A comparison of the different techniques for reading the contents of an XML document will be done as well.

2.2. XML Databases

2.2.1. What is an XML Database?

Before discussing XML Databases, it will help to understand what an XML database is. There is no incontrovertible answer to what an XML database is exactly. In fact, there are different views about what constitutes an XML database.

For the purpose of this thesis, we will consider an XML database as a collection of XML documents that persist and can be manipulated [25]. This definition is provided by Mark Graves in his book entitled *Designing XML databases*.

A *collection* is similar to a table in a relational database whereas XML *documents* are equivalent to records in a relational database.

XML documents can be classified on the basis of structure and the data they contain. They tend to be either *Data Centric documents* or *Document Centric documents*. These two types of documents will be explained in the following sections.

Data Centric documents are documents that use XML for data transport. They are designed for machine consumption and the fact that XML is used at all is usually superfluous [26]. The order of XML elements and overall physical structure of Data Centric documents are often unimportant. These elements can contain other elements or data.

Data Centric documents are characterized by a highly regular structure with many repetitions of data structures. The processing of the document is usually focused on its use and exchange by applications [25].

Examples of Data Centric documents are sales orders, price lists, scientific data, and stock quotes.

Data of the kind that is found in Data Centric documents can originate both in the database and outside the database [26].

Document Centric documents are documents that are usually designed for human consumption. In contrast to Data Centric documents, the order of XML elements does generally matter in this type of document.

They are characterized by a complex or irregular structure and mixed content, and their physical structure is important. Examples are books, email, advertisements, and almost any hand-written XHTML document [26]. The processing of the document is focused on the final presentation of the information to the user [25].

Document Centric documents are usually written by hand in XML or some other format, such as RTF, PDF, or SGML, which is then converted to XML. Unlike Data Centric documents, they usually do not originate in a database [26].

The two types of XML document discussed above differ in their structure and characteristics. Also the kind of operations that are desired on these XML documents will vary depending on the structure and content of a particular document.

In a Document Centric document, desired operations include retrieving the entire document, searching for a word, finding previous words or following words, modifying a section, or reordering a section [25]. In a Data Centric document, desired operations include retrieving a specific fragment of the document, searching for a particular combination of elements and data, modifying or deleting a single element or a single piece of data, or adding a new element to the document [25].

In spite their differences, both follow the XML specification and the distinction between the two is not always clear.

These two types of XML documents can be combined to form hybrid documents that are both Data Centric and Document Centric [27].

2.2.2. Why XML Databases?

Since XML was launched on the world in 1998, one of the areas where XML has made its mark is in data interchange between applications or businesses, especially for use on the Internet. As consequence of this, the number of XML documents generated from business transactions has increased. These XML documents can be of different types, as mentioned in the previous section. Also their contents may vary. Some may contain simple elements, and others may contain complex elements. This has raised the need in the research community to find an effective way of storing and processing XML documents.

All major vendors of traditional databases, both object-oriented and object relational, felt the need to extend functionalities and capabilities of their products to handle XML documents by providing some utilities, tools or techniques for converting data between XML and Relational database formats.

Because traditional databases originally were designed to store structured data, converting data between these two formats is not easy; it always comes at a price,

especially when dealing with an XML document containing complex elements. The mapping strategy is difficult to achieve. As a result, when an XML document containing complex elements is stored in a relational table, information can be lost, such as element ordering and the distinction between attributes and elements.

In an effort to address the shortcomings of traditional databases when handling a complex XML document, the XML community initiated an effort to develop a new kind of database: the so-called XML database. In contrast to traditional databases, XML databases are designed to store semi-structured data. Semi-structured data are defined as data whose structure is not rigid, complete, or fixed.

The main motivations for developing this kind of databases [28] are

- To use the data model of an XML document, rather than the model of the data it describes.
- To have an XML document as the fundamental storage unit. XML documents come out in exactly the same way [illegible] in.
- To enhance speed. Documents are converted to some internal representation, which is highly optimized for XML documents.

The key difference between XML databases and other databases, such as relational and object-oriented, is that the internal models of XML databases are based on XML.
XML databases can handle both categories of XML documents discussed previously in section 2.2.1. This does not mean that the XML databases are perfect technologies or have come to replace the existing databases whose technology is proven and mature.
"I do not view XML and relational storage as competitors; each has its strengths and may be better suited to particular purposes. There is room for coexistence in a well-planned environment [29]". This thesis by no means aims to provide a discussion on the topic "XML database Vs traditional database systems".

The following section describes the different types of XML databases.

2.2.3. Types of XML Databases

As discussed earlier, different kinds of XML documents exist and also different approaches to storing and managing XML documents have emerged. These approaches can roughly be classified into two types of XML databases: *XML-enabled databases (XEDB),* and *Native XML databases (NXD).*

This next section differentiates between these two types of XML databases.

As discussed by Ronald Bourret in his introduction to XML databases [20], Native XML databases differ from XML-enabled databases in three ways:

- Native XML databases can preserve physical structure (entity usage, CDATA sections, etc.) as well as comments, PIs, DTDs, etc. While XML-enabled databases can do this in theory, it is generally not (never?) done in practice.
- Native XML databases can store XML documents without knowing their schema (DTD), assuming one even exists. Although XML-enabled databases could generate schemas on the fly, this is impractical in practice, especially when dealing with schema-less documents.
- The only interface to the data in native XML databases is XML and related technologies, such as XPath, the DOM, or an XML-specific API, such as the XML:DB API. XML-enabled databases, on the other hand, offer direct access to the data, for example through Open Database Connectivity (ODBC).

While XML-enabled databases are not used in this thesis, some of their features are briefly discussed.

XML-enabled databases (XEDB) are traditional databases with added features for supporting XML. They have an added XML mapping layer provided either by a third

party or by the database vendor. This layer is used for storage and retrieval of XML data. For example the XML solutions provided by vendors like Oracle, IBM, and Microsoft fall in this category.

The table-based and the object-relational mappings are the two mapping techniques commonly used to map an XML document schema to a relational (database) schema.

When transferring data between relational databases, the table-based mapping is useful. The table-based mapping technique cannot easily be used, however, for XML data that has a highly irregular structure.

The object-relational mapping is used by most XML-enabled database systems to model an XML document as a tree of objects, which are specific to the data in the documents, although the way this technique is supported varies from product to product.

A relational query language like SQL, alone, has some shortcomings when querying XML. As a result, for querying the XML data stored in XML-enabled databases, an XML query language must be implemented separately.

Section 2.3 describes in detail Native XML databases (NXD).

2.3. Introduction to Native XML Databases

2.3.1. Introduction

Native XML databases (NXD) are databases designed especially to store XML documents [26]. They store XML documents in their original form. In other words Native XML databases store XML documents in their native form without decomposing them or without performing any conversion on them.

Like other databases, they support features like transactions, multi-user access, security, query languages, programmatic APIs, and so on [26]. There are other features, such as document storage techniques, which differ from product to product.

Before continuing the discussion further about NXDs, it will be better to understand what a NXD is.

In order to answer this question, let us examine one possible definition of NXD proposed by XML:DB Initiative, which is a consortium developing and implementing XML database standards.

According to XML:DB Initiative, a native XML database is one that [26, 30]:

- Defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX.
- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a record (row) in a table as its fundamental unit of (logical) storage.
- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed or compressed files.

One important point that we can learn from this definition is that NXDs provide robust storage and manipulation of XML documents.

2.3.2. Why do Native XML Databases exist?

As indicated earlier, a significant amount of research has already been done to adapt traditional databases for handling XML documents. But there are still certain cases where these XML-enabled databases have disadvantages, especially when dealing with complex XML documents. A few examples of these shortcomings are:

- There is a lack of perfect mapping between XML data and the relational schema.
- XML-enabled databases do not easily handle documents without a schema
- Retrieval of an XML document from an XML-enabled database may not reproduce exactly the same document as the original.

Native XML databases were created to resolve these problems.

Native XML databases are also useful for storing documents whose "natural format" is XML, regardless of what those documents contain [26]. They store XML documents without decomposing them. This gives the Native XML databases the capability to retrieve XML documents in exactly the same form in which they were stored.

Several other uses for Native XML databases are to store semi-structured data, to increase retrieval speed in certain situations, and to store documents that do not have Data Definition Types (DTD) or another type of XML Schema (i.e. Schema-less documents) [26].

Native XML databases support XML query languages, allowing for questions like "Get me all documents in which the third paragraph after the start of section contains a bold word". Such queries are clearly difficult to ask in a language like SQL.

2.3.3. Storage in Native XML Databases

From Ronald Bourret's point of view [26], the architecture of a Native XML databases is categorized as one of two types: *text-based* and *model-based.*

To get a clear idea about the architecture of a Native XML database, let us investigate further to discover how each of these two types of Native XML database architectures store XML documents.

Naturally, these two approaches differ in their ways of storing XML documents within databases. The two approaches are discussed in the next sections.

- A text – based NXD approach

In this approach, the entire XML document is stored as text. This might be a file system, a BLOB in a relational database, or a proprietary text format [26].

All text-based NXDs use indexing techniques. This allows the query engine to easily point to any part of the XML document. As a result such databases obviously have an enormous performance advantage when retrieving entire documents or document fragments.

In this sense, a text-based Native XML database is similar to a hierarchical database, in that both can outperform a relational database when retrieving and returning data according to a predefined hierarchy [26].

As with hierarchical databases, performance can be degraded when returning data in a form other than the entire document, such as inverting the hierarchy or portions of it.

- A model-based NXD approach

Unlike the text-based approach that stores the entire XML as text, this approach represents the contents of the XML document in a selected model.

Before storing any data (contents of XML document), the database builds an internal object model from the document and stores this model [26]. The storage of this model is database dependent. Whatever model (relational, object-oriented, or proprietary database) is selected, it needs to support the mixed content and semi-structured nature of a complex XML document.

This approach has the advantage of combining fragments from different documents.

Like a text-based NXD, model-based NXDs are likely to encounter performance problems when retrieving and returning data in any form other than that in which it is stored, such as when inverting the hierarchy or portions of it. Whether they will be faster or slower than text-based databases is not clear [26].

The following section explains the features of Native XML databases (NXD).

2.3.4. Features of Native XML Databases

Another important question that needs to be answered is, "what features are available in these Native XML databases?"
Not all Native XML databases are exactly the same. In spite of that, there is a considerable similarity amongst them.

The following items are the most common features of Native XML databases [26, 30]:

- Document Collections

The notion of a collection is supported by many Native XML databases. A collection is very similar to the concept of a table in relational databases. It plays a role similar to that of a table in a relational database or a di file system.
Native XML databases diverge from the table concept in that not all Native XML databases require a schema to be associated with a collection. This means that you can store any XML document in the collection, regardless of schema. Native XML databases that support this functionality are termed schema-independent [30].

Usually, a document collection contains documents of the same type. Its aim is to be able to focus on a certain kind of document when querying the database.
For example, suppose you are using a Native XML database to store sales orders, you might want to define a sales order collection so that queries over sales orders could be limited to documents in that collection [26].

- Query Languages

Almost all Native XML databases support one or more query languages. The most popular of these are Xpath (with extensions for queries over multiple documents) and Xquery, although numerous proprietary query languages are supported as well [26].

- Updates and Deletes

There are a variety of strategies for updating and deleting documents in Native XML databases.

These strategies are database dependent. Some databases simply replace or delete the existing documents, others use modification through a DOM tree, whereas yet other databases use special query languages such as Xupdate or Xquery, which specify how to modify a fragment of a document.

- Transactions, Locking, and Concurrency

All Native XML databases support transaction and rollback. However, locking in most NXDs takes place at the document level, rather at the document fragment level. This makes the multi-user concurrency relatively limited in such databases.

- Application Programming Interface (API)

Almost all Native XML databases offer APIs. They usually come in the form of an Open Database Connectivity (ODBC) Interface, with methods for connecting to the database, exploring metadata repository, executing queries, retrieving results and communicating over HTTP.

- Round-Tripping

This is one important feature of Native XML databases. It refers to the ability to store a document and get the same document back again. This feature is very important for document centric applications because they need exact ordering of a document.
All Native XML databases can round-trip documents at the level of elements, attributes, PCDATA, and document order [26].

- Indexes

All Native XML databases use indexes to speed up the queries.
There are three types of indexes [26]. These are value indexes, structure indexes, and full-text indexes. Value indexes index text and attribute values. Structural indexes index the location of elements and attributes. Full-text indexes index the individual tokens in text and attribute values.
Most Native XML databases support both value and structural indexes. Some Native XML databases support full-text indexes [26].

2.4. XML-related technologies for Native XML Databases

2.4.1. Introduction

As has been indicated, Native XML databases (NXD) are databases designed especially to store XML documents. Once XML documents are stored in an NXD, the need arises to manipulate the contents of the XML document collections.

The following sections discuss the different technologies that will be used throughout this thesis for performing different tasks with an NXD. These tasks include: accessing, manipulating, querying, and updating the contents of the XML document collections.

2.4.2. XML:DB API

Currently, there is a proliferation of Native XML databases on the market. This makes developing applications rather more difficult. Each Native XML database defines its own API. Therefore, interoperability is difficult. In other words it is hard for a programmer to develop software, which can work with more than one database without having to make any changes to the software.

In an effort to overcome the aforementioned problem, a decision was made by dbXML Group L.L.C, SMB GmbH, and the Open Healthcare Group to start the XML:DB API project.

The XML:DB API [31] is designed to enable a common access mechanism to XML databases. The API enables the construction of applications to store, retrieve, modify and query data that is stored in an XML database. These facilities are intended to enable the construction of applications for any XML database that claims conformity with the XML:DB API.

The goal of the XML:DB API is to bring similar functionalities like an Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) style access API to native XML databases. The API is intended to be language independent. It is targeted towards all object oriented programming languages, but most of the early work was done in Java. Implementations in other languages are also of great interest. It is also platform and vendor-independent. This allows it to provide as much as flexibility as possible and to be implemented by several XML databases

The API is built around some core concepts. These are: *drivers, collections, services, and resources*. We will only introduce the most important aspects of these core concepts here. For interested readers, we recommend the API specification [31].

- Drivers

Each database that supports the XML:DB must provide a specific driver that encapsulates all the database access logic. Drivers are implementations of the Database interface and are managed by the DatabaseManager [32]. The driver concept in XML:DB API does not differ much from JDBC, or ODBC.

- Collections

Collections are roughly equivalent to tables in a relational database. In native XML databases XML documents are stored in collections. The XML:DB API makes intensive use of the collection concept and assumes that any implementing database has at least one collection where documents are stored. Collections are represented in the API by the Collection interface [32].

- Services

The flexibility and extensibility of the XML:DB API is achieved through services. Services allow one to do a whole lot of useful work with the API. XPathQueryService is an example of a very widely used service; it enables execution of XPath queries against the database.

The API specification defines several services including XPathQueryService, XUpdatequeryService, and CollectionManagementService [32].

- Resources

Since there are numerous common ways of working with XML data, the XML:DB API defines an abstraction for the content stored in the database. This abstraction is encapsulated in the generic Resource interface. By specializing Resource it is possible to

support types of data beyond XML, for example, binary data. For XML the XMLResource specialization is provided and allows one to easily access and update the underlying XML data as textual XML, a W3C DOM, or a SAX event stream [32].

The XML:DB API is still currently evolving. Most of the core framework is stable, and it has been already implemented by most database products. There is also a reference implementation in Java available, and there are several other implementations in progress including some commercial databases [32].

All Native XML databases that will be discussed in this thesis have completely implemented this standard for their products.

2.4.3. XML Parser

An XML document consists mainly of objects. These objects can be elements, attributes, comments, processing instructions, entity definitions, entity references, or CDATA sections. Elements are the most common form of markup. Delimited by angle brackets, some elements may be empty, in which case they have no content. Attributes are name-value pairs that occur inside start tags after an element name.

In order to access data contained in an XML document, one needs a program than can read and understand the syntax and XML content model. Fortunately, the XML recommendation makes certain assumptions about the way in which an XML document will be processed. Rather than an application just treating the document as a piece of text, the model indicates the use of an XML processor that passes the content and structure of the document to an application [33]. An XML processor is also termed an XML parser.

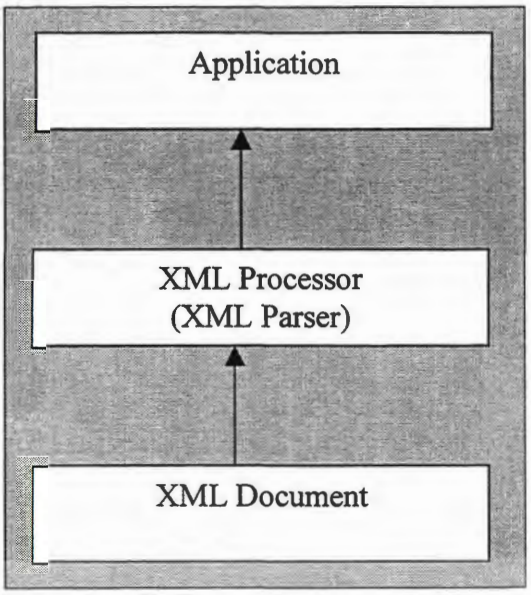


Figure 2.1 XML parsing process diagram

Figure 2.1 above shows an overview of XML parsing. In the middle of the figure one finds the XML processor. It acts as an interpreter between the application and the XML document. In other words, the XML processor takes the XML document as a data source and puts it in a format that the application can understand.

There are two kinds of XML parser [34, 35]. These are *non-validating* and *validating* parsers.

A non-validating parser simply checks that the document is "well formed", i.e. that it conforms to the basics of XML specification; a *validating* parser ensures that the document conforms to the DTDs or XML schemas specified within it. DTDs and XML schemas are different mechanisms that are used to specify valid elements that can occur in a document, the order in which they occur and constrain certain aspects of these elements.

There already exist a number of XML parsers, which are of quite professional quality today, from different vendors and in different programming languages in addition to Java. In addition, there are different types of Application Programming Interface (APIs) that can be used by XML parsers.

This thesis will limit the discussion to the two types of APIs that are commonly used by XML parsers. These are Document Object Model (DOM) and Simple API for XML (SAX).

DOM is discussed in section 2.4.3.1, SAX in 2.4.3.2 and these two distinctly different ways of working with XML will be compared in section 2.4.3.3.

2.4.3.1. Document Object Model (DOM) API

DOM [36] is a W3C recommendation that uses a *tree-based model* to represent the structure and content of the XML document being parsed. The DOM tree built up by the DOM parser usually resides in the memory. It consists of different types of nodes that can represent elements, attributes, CDATA sections and so on, defined in the XML document. These nodes are arranged according to the tree structure of the XML document. DOM uses the concept of a family tree to represent the relationship amongst nodes in a DOM tree whereby you have parent, child, sibling, ancestor and descendant just as in a family tree.

Consider the following simple XML document [37] containing employers' information:

```
<? xml version="1.0"?>
 <EMPLIST>
  <EMP>
    <ENAME> MARY</ENAME>
  </EMP>
  <EMP>
     <ENAME> SCOTT</ENAME>
  </EMP>
 </EMPLIST>
```

Figure 2.2 simple XML document

The DOM tree for the above XML document would be as follows:

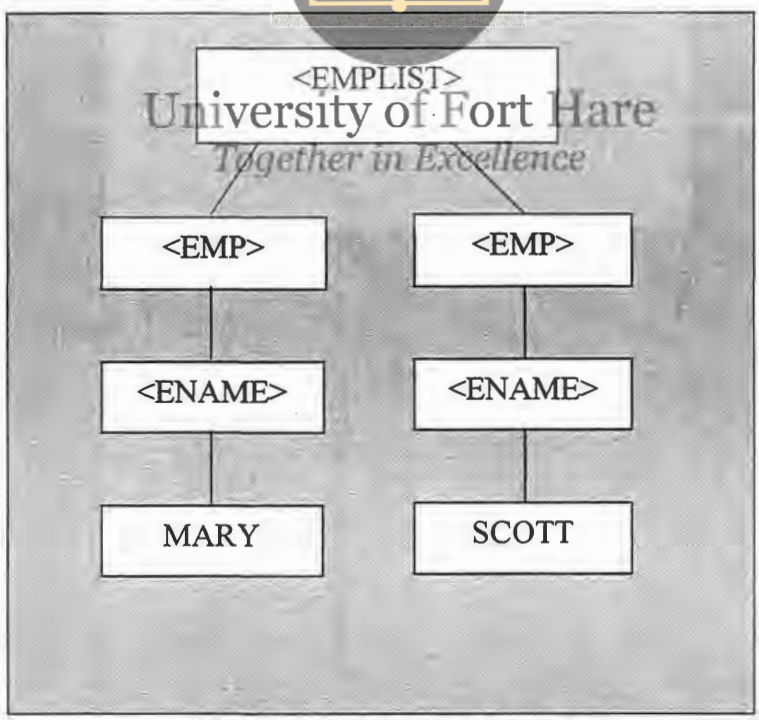


Figure2.3 DOM tree of simple XML document

Figure 2.3 shows the visualization of the XML document in Figure 2.2 Both <EMP> elements are children of <EMPLIST> and siblings of each other, while both these elements and all those below them are descendants of <EMPLIST>.

DOM API is essentially platform-independent and is language neutral for XML documents and HTML. Its purpose is to provide a standard set of interfaces for accessing and manipulating the content of an XML document. The API consists of a number of interfaces, each representing part of an XML document.

Table 2.1 [38] gives a summary of several important DOM classes and interfaces and their descriptions.

Interface	Description
Document interface	Represents the XML document's top-level node, which provides access to all the document's nodes-including the root element.
Node interface	Represents an XML document node.
NodeList interface	Represents a read-only list of Node objects
Element interface	Represents an element node. Derives from Node.
Attr interface	Represents an attribute node. Derives from Node.
CharacterData interface	Represents character data. Derives from Node
Text interface	Represents a text node. Derives from CharacterData.
Comment interface	Represents a comment node. Derives from CharacterData
ProcessingInstruction interface	Represents a processing instruction node. Derives from Node.
CDATASection interface	Represents a CDATA section. Derives from Text.

Table 2.1 DOM classes and interfaces

The interfaces listed in Table 2.1 provide convenient operations for accessing and manipulating documents.

DOM- based parsers are written in a variety of programming languages and are usually available for download at no charge.

2.4.3.2. Simple API for XML (SAX) API

Shortcomings of DOM, particularly problems with processing arbitrarily large XML documents, motivated application programmers to use other technologies. One of these is SAX [39].

SAX was developed by the members of the XML-DEV mailing list. It is an alternative approach for parsing XML documents that uses an *event-based model* to represent the structure and content of the XML document being parsed.

Unlike the tree-based model, the event-based model is stream based and does not build an internal representation of the document. In other words, SAX does not load the entire document into memory at once. This makes SAX better for accessing large XML documents. The SAX parsing process is [illegible]raightforward. While parsing the XML document it reports events (notifications) as it reaches important parts of the document such as startElement, endElement, characters, and so on by calling functions (methods) of event handlers. These events provide information such as the element name, its value, its attributes, and so on as defined in the XML document. Events are also referred to as callbacks.

Methods that SAX parsers invoke when events occur are listed in Table 2.2 [38]. A short explanation for each of these methods is also provided.

Method Name	Description
SetDocumentLocator	Invoked at the beginning of parsing
StartDocument	Invoked when the parser encounters the start of an XML document
EndDocument	Invoked when the parser encounters the end of an XML document
StartElement	Invoked when the start tag of an element is encountered.
EndElement	Invoked when the end tag of an element is encountered.
Characters	Invoked when text characters are encountered.
ignorableWhitespace	Invoked when whitespace that can be safely ignored is encountered.
processingInstruction	Invoked when a processing instruction is encountered.

Table 2.2 Methods invoked by the SAX parser

Figure 2.4 illustrates the way SAX parsing will break down the structure of the sample XML document in Figure 2.2 into a series of linear events.


```
start document
start element: EMPLIST
start element: EMP
start element: ENAME
characters: MARY
end element: EMP
start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP
end element: EMPLIST
end document
```

Figure 2.4 SAX events generated in simple XML document

There are three basic kinds of event handlers within SAX. These are:

DTD handler - for accessing contents of XML Data Type Definition

Error handler - for low-level access to parsing error

Content handler - for accessing the contents of the documents.

These handlers are bits of code that do something suitable when the occurrence of an event triggers them [33]. It is the duty of application programmers to write handlers for each event.

Like DOM- based parsers, SAX- based parsers also are available for a variety of programming languages and there are several available for free downloading.

2.4.3.3. DOM vs. SAX

DOM and SAX are created to serve the [redacted]pose, which is to provide access to the information stored in an XML document and also they have both been implemented in a number of programming languages in addition to Java.

The two aforementioned technologies differ in their parsing models. DOM uses a tree-based model and SAX uses the event-based model. Each of these models has its advantages and disadvantages.

DOM provides easy access to any nodes in the DOM tree, and also provides facilities for manipulating nodes such as adding, removing, or updating a node in the tree. A disadvantage of using DOM is that it consumes more memory especially when processing large XML data sets because it loads a whole document into memory before processing can occur. It also takes time to build the tree.

DOM is useful for applications that include changes. For example reordering, adding, or deleting elements.

SAX, on the other hand, does not allow random access to the document, does not provide any means of updating XML documents, and also SAX events are not permanent.

An advantage of using SAX is that it does not build an internal representation of the XML document into the memory. This makes SAX better for accessing large documents. Also its approach of presenting the document as a sequence of events makes it a lightweight API that is good for fast reading.

SAX is useful for applications such as search and retrieval that do not change the XML tree.

The differences between DOM and SAX are summarized in the table below.

	DOM API	SAX API
Model	Tree data structure	Event based model
Information Access	Random access (in-memory data structure)	Serial access (flow of events)
Memory cost	High memory usage (document is loaded in memory)	Low memory usage (only events are generated)
Startup time	Slower because every element is parsed	Faster, especially if the elements of interest are easy to locate
Repeated Search time	Faster because everything is in memory	Slower because every search involves a new parsing run
Modification capability	Very flexible	Limited to writing a new XML document with every pass.

Table 2.3 SAX and DOM comparison

Despite their difference as shown in the Table 2.3 above, these two technologies can be complementary. In fact, many DOM parsers use a SAX parser to retrieve data from a document for building the DOM tree [38].

- Other techniques

Additional to DOM and SAX, there are other techniques that can be used to parse XML documents. These will not be discussed in detail here. For the sake of completeness, the following list has been provided. For the interested readers, see the literature references in order to read more about alternative ways besides DOM and SAX.

- JAXB – Java Architecture for XML Binding [40]
- JAXP – The Java API for XML Processing [41]
- JDOM – The Java Document Object Model [42]
- DOM4J – Document Object Model for Java [43]

2.4.4. XPath

XPath [44] is a selection language defined by the W3C that provides syntax for extracting a node or a set of nodes from an XML document effectively and efficiently.
Xpath is a string-based language of expressions that may be used in many contexts with other XML technologies such as XML Pointer, which is a technique for building pointers into XML documents and also XSL for specifying patterns in style sheets.

Like DOM, the data model in Xpath treats a document as a tree of nodes of which there are seven types: root, element, attribute, text, namespace, comment, and processing instruction.

Xpath supports the *location path* method. Location path defines the context of the nodes to be found, and the condition that the selected nodes must satisfy. It also allows for navigating an Xpath tree from one node to another. A location path consists of one or more location steps. A location step is made up of *axis, node test, and optional predicate*.

XPath path expressions are broadly similar to a set of street directions. You have a defined starting point (in XPath jargon, the context node), a direction (called in XPath an axis), and some description of the final destination, which distinguishes it (in Xpath, a predicate) [45].

The *Axis* is the direction of movement. It indicates which nodes, relative to the context node, should be included in the search. The axis also dictates the ordering of the nodes in the set [38].

The *Node test* refines the set of selected nodes. Node tests rely upon the principal node type of an axis for selecting nodes in a location path.

Predicates are expressions used as criteria for inclusion in the set of nodes selected. They are placed in square brackets.

XPath expressions also use some abbreviated syntax, as in table 2. 4, to shorten their path expressions for the location path.

Expression	Meaning
/	Root node, or a separator between steps in a path
//	Descendants of the current node
.	Current node
..	Current node's parent
@	Attributes of the current node
*	" Any " (node with unrestricted name)
[ ]	A predicate for a given step

Table 2.4 some location-path abbreviations

Referring to our sample XML example figure 2.2, the syntax in table 2.4 can be used to create the following example queries in XPath, as in table 2.5.

Queries	Description
/ EMPLIST/ EMP or // EMP	Find all EMP elements that are inside EMPLIST elements. In our case it would select both <EMP> elements in the XML documents
/ EMPLIST/ EMP/ *	Find all elements that are children of <EMP> element. In our case it would select both <ENAME> in the XML document
/ EMPLIST/ EMP[ENAME="MARY"]	Find an element that contains an element <ENAME> with the value "MARY". In our case only the first <ENAME> element in the XML document would be selected. Using the XPath query example, we describe the *three parts* of a location step namely *axis, node test, and predicate*. • *Axis* The axis (or direction) in this case is from the root node to the ENAME node element, location steps are separated by slashes (/) • *node test* A node test is an evaluation to determine which nodes along the axis are selected for the next step which in our case is */EMPLIST/EMP*. The first step selects the

	EMPLIST node and the second step selects the <EMP> element that is child of that node • *predicate* A predicate is an expression which is enclosed in brackets, which in this case is *[ENAME="MARY"]*

Table 2.5 XPath example queries

These are just some simple queries. Xpath provides several functions and special elements that allow the creation of quite complex selection expressions. An XPath expression is a union of one or more location paths. Its aim is to map a node called the *context node*, which is the node that the XPath expression has reached in the XML document, into a set of nodes.

In addition to being able to select pa[illegible] XML document, XPath also supports functions and node-set operators.

XPath provides a number of functions for the manipulation of strings, booleans, and numbers.

Node-set operators allow us to manipulate these node sets to form other node sets. Xpath also provides node-set functions that perform an action on a node-set returned by a location path.

It is recommended that readers, who would like to know more about Xpath, see W3Schools Tutorial [46] and Xpath tutorial [47]. They have a comprehensive set of practical examples explaining the various ways of selecting data with Xpath.

2.4.5. XUpdate

Unlike XPath, XUpdate [48] is not a W3C recommendation. It is an XML update language developed by the XML:DB initiative. Xupdate is designed to work with regular XML documents as well as XML database collections and even virtual XML data models. The main purpose of this update language is to provide open and flexible update facilities for modifying data in XML documents. To achieve this, Xupdate provides functionality for creation, insertion, deletion, value updating and renaming of XML nodes. It uses XML syntax for the query. In order to select the nodes to be updated, Xupdate makes heavy use of the expression language defined by Xpath. This also applies for conditional processing.

XUpdate has specialized elements that define output operations. In other words, Xupdate specialized elements allow one to specify what changes should be made in the XML document that needs to be modified.

In the next section, some specialized elements are listed, and also, a short explanation for each of the listed elements is provided. For a detailed description of these functions, please refer to [48].

An update in the XUpdate language is basically a well-formed XML document by itself, because it only contains one top-element, often called the root element, tags are properly balanced, and attribute names are unique and their values quoted.

An update is represented by an *Xupdate: modifications* element in an XML document. Every update needs to have the following root element:

<xupdate:modifications version ="1.0"
xmlns:xupdate ="http://www.xmldb.org/Xupdate"> </Xupdate:modifications>

An *Xupdate: modifications* element must have a version (e.g. "1.0") attribute which specifies the version of the XUpdate as shown above.

As the root element (top-element), the *Xupdate:modifications* can contain one or more of the following types of elements:

Xupdate:insert-before	To insert data before the selected element
Xupdate:insert-after	To insert data after the selected element
Xupdate:append	To append data to the selected element
Xupdate:update	To update the content of the selected element
Xupdate:remove	To remove the selected element
Xupdate: rename	To rename the selected element
Xupdate:variable	To define a variable that can be used by other Xupdate elements
Xupdate:value-of	To get the value assigned to a previously defined xupdate:variable element
Xupdate:if	To perform conditional processing

The insert and append elements, in their turn, contain the following types of elements:

Xupdate:element	To create a new element
Xupdate:attribute	To add an attribute to an element
Xupdate:text	To add a text node in the result tree
Xupdate:processing-instruction	To create a processing instruction node
Xupdate:comment	To create a comment node in the result tree

In order to illustrate the basics of the XUpdate language, a simple example in which Xupdate renames an element is provided. A simple XML document in Figure 2.2 is used to illustrate this example.

```
<xupdate:modifications version ="1.0" xmlns:xupdate ="http://www.xmldb.org/Xupdate">
< Xupdate: rename select =" /EMPLIST/EMP/ENAME">
 ESURNAME
</ Xupdate: rename>
</Xupdate:modifications>
```

Figure 2.5 An Xupdate rename

The piece of code shown in Figure 2.5 would rename the element "ENAME" to "ESURNAME"

As we can see, the Xupdate rename in Figure 2.5 is a well-formed XML document, because it obeys all the syntax rules for XML, in that every start tag has a corresponding end tag and it has exactly one root element namely *Xupdate: modifications*. The root element *Xupdate: modifications* also has two attributes whose names are unique (e.g. version, and xmlns:xupdate) and their values ("1.0" and "http://www.xmldb.org/Xupdate") are quoted.

For a detailed description of the elements and functionality of Xupdate, please refer to [48] and [49]. They have more information and very good online tutorials.

The following Table 2.6 gives a summary of the XML technologies that will be used throughout this thesis.

Types	Technologies
Accessing technologies	XML:DB API
Program Manipulation Technologies	DOM, SAX
Querying Technologies	XPATH
Updating Technologies	XUPDATE

Table 2.6 Various XML Technologies used

The XML technologies shown in Table 2.6 are not the only XML technologies available. There are others that are not discussed here. For the sake of completeness, the following

list has been compiled, which contains some of these technologies. See literature references in order to read more about these technologies.

- XSLT – Extensible Style Sheet Language [50]
- XSD - XML Schema Definition Language [51]
- DTD – Data Type Definition [52]

2.5. Open Source Native XML Databases

There are a number of Native XML databases competing for this expanding market. These come in two groups: *Commercial* and *Open Source* Native XML databases.

For this thesis only Open Source, free Native XML databases are considered. There are number of reasons for choosing Open Source products such as the licensing costs issues, and access to source code.

Before continuing further, let us try to understand what *Open Source* means.

In general, *Open Source* refers to any program whose source code is made available for use or modification as users or other developers see fit [53].

For the rest of this thesis only Open Source Native XML databases will be considered. Some of these Open Source Native XML database products are discussed in detail in Chapter 3.

2.6. Summary

This chapter gave an overview of XML databases and XML-related technologies. The XML document is the storage unit for XML databases. XML documents can be classified according to the structure and data they contain. Based on that, an XML document can be a *Data Centric document* or a *Document Centric document.* There are two different types of XML databases, *XML-enabled databases (XEDB),* and *Native XML databases (NXD).* These two are related but have different approaches for handling XML documents. For the scope of this thesis only NXD was considered for discussion. The features, and storage strategies used by Native XML databases (NXDs) were examined.

NXD are specialized databases for storing XML documents. On the other hand, data contained in XML documents can have regular or irregular structure. In order to access data contained in an XML document, one needs a program than can read and understand the syntax and XML content model. [illegible] XML-related technologies used for accessing, parsing, querying, and updating XML documents contents were also discussed.

Chapter 3

EVALUATION CRITERIA AND SELECTION OF PRODUCTS

3.1. Introduction

In the previous chapter, we defined the XML database, gave overview of it, the reason for its existence, its benefits and its features. Also the types of XML databases available were discussed, focusing more on Open Source Native databases. Furthermore technologies used to access the XML databases and also different techniques for manipulating data contained inside the database were examined.

This chapter will discuss the evaluation methodology. It will start by identifying attributes that are felt to be important for carrying out our evaluation. Each of these attributes will be explained.

Chapter 3 will further talk about the selection criteria that will be used for choosing the Open Source Native XML Database products to be studied for this thesis, and then some of the previous related studies done [illegible]y of the selected databases will be highlighted. The chapter will also briefly introduce each of the selected products by giving a high level overview, and discussing some of the basic features for each of them. Features that will be considered for discussion here are only those that are judged as significant for the scope of this thesis. In addition to this, each of the studied products will be evaluated using a drawn up list of attributes that are considered important.

Lastly, this chapter will briefly compare some of the key features and attributes of all the selected Open Source Native XML Database products.

3.2. Evaluation Criteria

Before using or implementing a particular product, it is necessary to know the benefits, drawbacks, and risks of using that particular product. One of the ways to accomplish this is by evaluating that product.

The evaluation of product or software is an entire field in the Information Systems profession. A number of approaches are available for assessing the quality of a particular product; most of the time approaches taken during evaluation will depend on the evaluator's needs.

This thesis is not intended to conduct an evaluation of the Open Source NXD products in depth, but rather to examine some of these products' attributes that are considered relevant for the thesis. The idea behind this is to determine the benefits, drawbacks and risks of using such products.

A list of attributes that are considered important are drawn up. The following attributes will be used for the evaluation of the Open Source NXD products: functionality, market share, support, maintenance, performance, scalability, usability, security, and flexibility.

The next sections will discuss each of these attributes.

3.2.1. Functionality

This refers to the capabilities of a particular product to do what the user wants it to do. It is difficult to find programs that provide all the functionality that one would like. For that reason, functionality is dependent on the particular user's need. Therefore its evaluation will vary from one user to another. When assessing functionality of a particular product, it is often useful to write down at least a brief list of the functions that are important to the evaluator.

The scope of this thesis is limited to the discussion of the general functionalities of Native XML databases. The term general functionalities refers to common functions that need to be performed when working with a database. These include: storing, querying, updating, and deleting data. The platform or operating systems requirements for Open Source Native XML database products will also be investigated.

3.2.2. Market Share

This section refers to the popularity of a particular product. When planning to use a product, it is also important to know about its popularity, because most of the time products with high popularity are often easier to support.

Market share is extremely hard to measure for most Open Source products, because they can be downloaded and installed without registering with anyone [54].

Judging a product only according to its popularity can be misleading, because the popularity of a product can just be an indication that a particular product is more widely admired than others. This means that popularity may show widespread interest (e.g. it is an interesting product), rather than that the product is widely used or ready for use.

Another factor that one needs to consider when evaluating the market share of a product may be the mailing list activities. The activities in the mailing list can give one an idea about the popularity of a particular Open Source product.

3.2.3. Support

In this context, the word support will refer to the assistance that users of a particular product can obtain when they encounter specific problems during product usage. Support

is a very important issue for the success of an open source project. The level of support provided to the users can determine the future of a particular project. Users or developers lose interest in using products that offer little assistance because no one would bother working with a project that leaves his/her questions unanswered. As result, Open Source projects that provide good support to their users seem to have many users and developers. On the other hand, Open Source projects that do not provide adequate support to their users tend to collapse. This is an indication that product support is the key to or the foundation for the survival of a particular open source project.

When working with OSS products, there are different approaches to how support can be handled. For the purpose of this thesis, the term "support" covers several areas: installing the product, documentation available, and mailing list activities.

3.2.3.1. Documentation

This will refer to the amount of technical information available for a given OSS/FS product. Documentation is sometimes a major problem when working with an OSS/FS product. Most of the Open Source projects seem to have a serious problem with supplying decent documentation to their users, or, if they provide any documentation at all, it is not updated regularly or maybe it does not say everything that a user would need to know about the product. It is usually intended to be a general guide rather than a complete manual that could be handed to a beginner.

The lack of adequate documentation for OSS makes the task difficult for users of that particular product. One of possible reasons for the lack of adequate documentation for OSS products might be the fact that many times, OSS developers consider their involvement in a particular OSS project as a voluntary contribution. Many OSS projects are viewed as voluntary work; OSS developers prefer spending their time writing the software and do not really care about documenting it since they do not really have a contractual responsibility to provide the documentation.

Documentation is imperative for a given product that is to be used by someone other than the developer of that particular product. On the other hand, when writing documentation for a particular product, one needs to know that it is not the amount of information available in the documentation that counts but rather the quality of the documentation. The better the quality of the documentation, the easier it will be for the user of that particular product. Most of OSS/FS products face the problem of quantity Vs quality.

3.2.3.2. Installation

The complexity of installing a particular OSS/FS product is evaluated here. In other words, there is an investigation of the clarity of information available that will guide the user of a particular product to follow the product installation process. Product installation steps themselves will not be covered here.

As was acknowledged in the previous section, quality documentation should be a complete manual that could be handed to a novice. In other words, the documentation should always cater to the lowest common denominator. Information contained in documentation must give the user clear installation instructions about that specific product so that, after a user has followed all those instructions, he/she would be able to use the product and also get responses to what, why, and how questions about a particular product.

In reality, installation and documentation are interconnected; they go hand in hand and are not completely separate.

3.2.3.3. Mailing List

This is an interactive way commonly used by the OSS/FS community to provide support for a given product. It also facilitates the interaction between developers and users of a given product. Through the mailing list, users who have specific problems trying to use a working product can ask their questions. Mailing list participants can also discuss other issues such as the products' features and can report bugs, and so on. However, this does not mean that all OSS/FS products are well supported in this way, or that all users should choose this route.

The activities in the mailing list can determine the future or success of a particular product. OSS/FS projects are easier to sustain once they have many users; many developers are originally users, so if a small percentage of the users become developers, having more users often translates into having more developers. Also developers do not want their work wasted, so they will want to work with projects perceived to be successful [54].

3.2.4. Maintenance

Product maintenance is very important because from time-to-time, needs change; new uses are continuously created, and so on. This is the challenge faced by most OSS/FS projects to keep their products alive, and also determine their future. If a product is being actively maintained, it is far more likely that the product will continue to be useful.

OSS/FS maintenance options are essentially the same as those for support. In reality, maintenance and support go hand in hand and they are not completely separate.
A general way to measure maintenance is to examine developers, or users' mailing list activities or archives to check if there is evidence that they are actively discussing improvements to the product. More often, the OSS/FS developers that have experience

in providing support, and maintaining their products, will be able to make changes or fixes as necessary.

3.2.5. Performance

This refers to the ability of a product to perform under different workloads. Here the emphasis is on the speed at which a given OSS/FS product performs under different conditions. The best way to obtain more detailed information about performance of a given OSS/FS product is through that particular project's mailing list. Performance will not be referred to further here; rather we will discuss it in more detail in chapter 5. It is the aspect of XML databases that we are investigating in this thesis.

3.2.6. Scalability

This refers to the capability of an OSS/FS product to continue to function well when it is changed in size or volume to meet the user needs. In other words, scalability refers to the size of data or problem a given OSS/FS product can handle.

3.2.7. Usability

Usability measures the quality of the human-machine interface for its intended user. A highly useable product is easier to learn and easier to use [54]. There are some elements that should be taken into consideration when assessing usability of a product such as visual consistency, user interface (i.e. Command line interface or Graphical User Interface), and so on.

3.2.8. Security

In general, evaluating a product's security is complicated, in part because different uses and different environments often impose different security requirements on the same type of product [54].
In this context, the discussion will be limited only to the basic security issues of a given product (e.g. user authentication). Advanced security features are beyond the scope of this thesis. Therefore they are not considered for discussion here.

3.2.9. Flexibility

This measures how well a given product can be used to handle unusual circumstances that it wasn't originally designed for. In this attribute, OSS/FS products have a significant advantage: since the user or developer has access to the code, any OSS/FS products can be modified to handle any circumstance not previously considered [54].

3.3. Open Source Native XML Database products

3.3.1. Introduction

As pointed out earlier, there are several Open Source Native XML Database systems that have been and are still being developed both in industry and academia.
Ronald Bourret in his article entitled *XML Database Products: Native XML Databases* [20] gives a list of some of Open Source XML Database products that are available. A detailed discussion of all these Open Source XML Database products is beyond the scope

of this thesis. Interested readers are referred to [20] in order to read more about these products.

Taking into account what has been revealed in the previous paragraph, the first step of this work was to select appropriate candidates for this thesis. Section 3.3.1.1 explains the selection criteria that were applied in choosing the Open Source Native XML Database products that have been used.

3.3.1.1. Selection Criteria

Specific criteria were applied for selecting the Open Source Native XML Database products used for this study. Some of these are Open Source Native XML Database products, which are written in the Java programming language, as this is the language which is most familiar. Documentation issues were also considered. A particular product should have comprehensive documentation and support in the form of newsgroups.

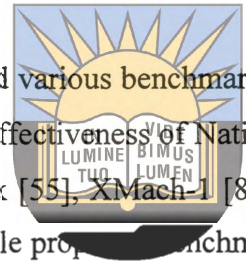
After reviewing the available candidates, not all the Open Source solutions met the requirements. The choice was quickly narrowed down to the following Open Source Native XML Database products:

- Xindice [21], an Open Source Native XML database by the Apache Software Foundation
- eXist [22], an Open Source Native XML database founded by Wolfgang Meier at the Technical University Darmstadt, Germany
- dbXML [23], an Open Source Native XML database mainly developed by dbXML Group
- Berkeley DB XML [24], an Open Source Native XML database by SleepyCat Software.

3.3.2. Previous studies using the selected Open Source Native XML databases

The various studies done in the field of Native XML database performance have shown that performance comparison evaluations of Open source native XML database systems is ongoing research and still remains an open issue. Furthermore, there is no unique approach or tools available for tackling this entire performance problem.

In an effort to address this problem, diverse research groups have come up with different approaches for solving this performance comparison dilemma.



Some research groups have proposed various benchmark techniques as possible tools that could be used for investigating the effectiveness of Native XML databases. X007 [3, 4, 5, 6], XMark [7], Michigan benchmark [55], XMach-1 [8, 9], and XBench [10, 11, 12, 13] are examples of a number of available pro[illegible]nchmark techniques. These benchmark techniques can be divided into two categories; the Micro benchmark and the Application benchmark. The Micro benchmark is designed to test individual system components to isolate problems, measure, and, thus, improve a particular component of an XML system. Amongst the above named benchmark techniques, [55] is the only one that belongs in this category. The Micro benchmark category is outside the scope of this thesis. On the other hand, application benchmarks measure the overall performance of a Database management system. The other four benchmarks, namely [3], [7], [8], and [10] fit in the application benchmark category. All of the application benchmarks listed above center more on testing the query processing abilities of XML databases. For example, [7] and [10] each consist of twenty queries, [3] consists of eighteen queries, and [8] consists of eight queries. They differ in terms of their query functionalities, in the sense that each benchmark is superior to the others in one or more features; but their similarity is that they do not include all the databases operations. [8] is the only one which contains queries and update operations. It contains three simple update operations. Although it contains queries and update operations, like those other listed benchmarks, it is still not

complete because it lacks other database operations. So far, there is not yet an official benchmark. Most of these proposed benchmarks are just individuals' benchmarks.

Other research groups have concentrated their effort in doing performance comparison studies of XML-enabled and Native XML database systems. For example, [14] compared two different approaches to XML data management. For the first approach, they use Oracle, a widely used commercial XML-enabled database DBMS. For the second approach, they used eXist, an open source native XML database. These two databases were compared in terms of their response times for some common database operations. The database query operation was the only operation that they investigated in their study.

The study in [15] compared the time taken and space consumed by five different database systems when storing and extracting XML documents. This comparison was done with the aim of determining the suitability of these alternatives for storing XML documents. The five database systems investigated in the performance comparison study are the following: relational, object-oriented, object-relational, directory servers, and native XML database systems.

[16] also conducted a performance comparison of an XML-enabled database and a native XML database (for legal reasons they said that the products cannot be named)

As explained in earlier sections, many of these performance comparison studies that have been done by various research groups focused on comparing the XML-enabled and Native XML databases capabilities. In particular, their capacities for handling database query operations were investigated. There are few studies that attempt to compare Open Source Native XML database products against each other. As results of this, the performance difference between various existing Open Source native XML database products is still an open issue. This results in hesitation among the database community members to approve this new technology.

The next section, will introduce some of these studies that attempt to compare Open Source Native XML database products against each other. So far, the various studies done for comparing the Open Source Native XML database systems can be divided into two categories. The first category consists of studies done in comparing the features of Open Source Native XML database systems. Researchers in the second category compared the capabilities of Open Source Native XML database systems in performing common database operations.

In [56] the authors discussed and compared the features of five native XML databases products. These include three leading commercial products and two Open Source Native XML databases. The three commercial products investigated were: excelon's extensible information server (XIS), Ipedo XML database, and Tamino XML server. Xindice and eXist were the two open source native XML databases that were considered in their comparison study. The performance comparison study in [69] was done with respect to the technologies used by these previously named five databases to access, query, and update stored documents.

Similar to [56], [57] also evaluated and compared features of Xindice, eXist, and Tamino. The comparison of these products was done according to the products' features that the evaluators were interested in. With their project objective in mind, the aim of the comparison was to show the strengths and weaknesses of each of the three named Native XML databases in order to have a basis to select one for their project's task.

Unlike the first category as discussed above, in the second category researchers focused on comparing the ability of Open Source Native XML database systems in performing some of the common database operations.

[17] reports the performance comparison study done on some Open Source Native XML database systems. The compared databases products include eXist, Xindice, and Berkeley DB XML. The aims of their comparison evaluation were to analyze and compare the

space required to store the XML documents, the time taken to insert XML documents, and time taken to perform different kinds of queries by each of these named databases.

Like [17], [18] also compares the performance of eXist and Xindice in carrying out some common database operations. The database operations investigated in [18] include four common database operations i.e. insertion, query, modification, and delete.

The aims of [18], and this work are essentially identical; both compare the performance capabilities of Open source Native XML databases in carrying out common database operations. This study extends [18] in several ways. First, eXist and Xindice are the only open Source XML databases considered in [18]. In our performance comparison, we considered four open Source XML databases. These are eXist, Xindice, dbXML, and Berkeley DB XML. Secondly, these open Source XML databases are explored in depth (i.e. the benefits of using such technology, and their architectures). Thirdly, different XML technologies used by the open source XML databases for achieving the investigated common database operations [illegible]ed.

In the following sections, each [illegible] will be introduced and evaluated using the attributes discussed in section 3.2. The aim of this evaluation is to show the strengths and weaknesses of each of these Open Source products. Furthermore, a comparison of their features and attributes will be done as well, in order to examine the products' similarities and differences.

For the sake of completeness, it is necessary to point out that not all the features of the aforementioned Open Source XML database products will be discussed here. The discussion is limited to some of the key features that fit in with the scope of this thesis. Readers, who would like a detailed technical explanation about all the features of these Open Source products, are referred to [21, 22, 23, and 24] which contain more information.

3.3.3. Xindice

3.3.3.1. Overview and Features

Xindice is a proprietary database developed by volunteers under the license of the Apache Software Foundation, which uses a model-based storage strategy for storing XML documents in the database system. Documents are not required to have a DTD or conform to a prescribed schema. XML documents are stored in a collection and are arranged into a hierarchy of collections like a traditional file system. In Xindice, one collection can contain sub-collections in a hierarchical manner.
By default each document is associated with a unique id when it is added into the database.

Xindice can run on a Windows, UNIX, and Linux platform. For the scope of this thesis, we will only consider the Windows platform.

Xindice supports XPath as a query language. This language is used for querying XML documents stored in a collection. In Xindice, XPath queries are executed at the collection level. Xindice does allow the manual creation of indexes through the command line interface. An index can be created based on element or attribute values of some elements or all elements in order to improve or speed up queries.

For updates, Xindice supports the XUpdate language from the XML:DB Initiative. The update can be applied to either a single XML document or an entire collection of documents.

Xindice provides excellent standards support for the vendor independent XML:DB API which comes with DOM and SAX support. It also supports others APIs such as CORBA API, and an XML-RPC plugin which supports access from languages such as PHP, Perl, and AppleScript.

XML:DB API is the only API that will be considered for the scope of this thesis.

3.3.3.2. Functionality

Xindice provides good support for basic functionalities. It stores data in XML format. Apart from its storage capability, Xindice also offers a means of manipulating data contained in the database. It utilizes Xpath for querying data stored inside the database, and supports XUpdate for updating its data.

3.3.3.3. Market Share

As stated earlier, most Open Source products' down side is that they lack a proper mechanism for registering their users. This lack of information about people, who use a particular product, makes it difficult to k[illegible]pularity of a particular product.

There are many approaches that one can use to get information about a product's market share. In this case, two different approaches were used.

In the first approach, the web search engine from Freshmeat [58] was used to get information about "vitality" and "popularity" of these products. Freshmeat is the first stop for Linux users hunting for the software they need for work or play [59]. It offers a variety of information about thousands of applications, which are preferably released under an open source license including open source projects. The percentage figures that we used for evaluating the "popularity" and "vitality" of the different open source native XML databases used in this thesis is one example of the kind of information that is provided by Freshmeat.

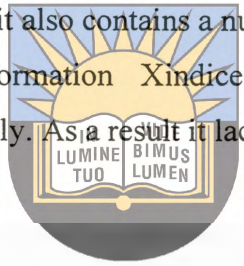
The second approach used the products' mailing lists to get opinions from users and developers about the products.

According to Freshmeat [58], Xindice has 1.62 % of popularity and its vitality is 0.01%. In [60, 61], Xindice is classified amongst popular native XML databases. From feedback that was got from the mailing list, a considerable number of people seem to be happy working with Xindice.

3.3.3.4. Support

3.3.3.4.1. Documentation

Xindice has good documentation available, which supplies enough information about the product and its related technologies; it also contains a number of examples.
In spite of the amount of information Xindice documentation provides, this documentation is not updated regularly. As a result it lacks other details.



3.3.3.4.2. Installation

Xindice documentation has a considerable amount of information that will guide the user on how to set up the build environment and how to install the server.
As highlighted in the previous section, there are a few gaps in the Xindice documentation. For example, the Xindice source distribution was used. There was no information about where to put the file *xindice.jar* after building it. It took several hours to find out how to get the system running. After we found out where to put *xindice.jar* file, the rest of installation process was very easy to do.

The database sever of Xindice is delivered as a servlet. It is not a standalone server anymore.

3.3.3.4.3. Mailing List

Xindice support seems to be slow in providing help, responding to queries and suggestions on its mailing list. This is not a good sign for the future of this product.

3.3.3.5. Maintenance

Most Open Source products depend on users' feedback and comments in their mailing lists for making improvements or fixing bugs. Mailing list content can give an indication of whether a product is being maintained or not. Users tend to lose interest in a product when they do not get enough technical support.

Xindice products' maintenance appears to be very slow. For some time, there have been numerous debates and questions on the Xindice mailing list about the Xindice development status. There have been a few attempts to explain this phenomenon and many reasons have been given to justify it. For example, somebody said this"Xindice seemed to have disappeared around the same time the XML version 2.0 was released".

3.3.3.6. Scalability

According to [62], Xindice was not designed to store and manage single monster sized documents, where one document is treated as a set of mini documents. It was specifically designed for managing many small to medium sized documents.

Xindice's scalability will be further scrutinized in chapter 5.

3.3.3.7. Usability

Xindice lacks a Graphical User Interface (GUI). It comes with a command line interface to perform the administration of the database. In order to use this interface, one must learn about the Xindice command line syntax and also remember the commands every time one needs to perform a task. Sometimes this can be annoying.

3.3.3.8. Security

Xindice lacks transaction control. It also has no administrator or user authentication method and no security management.

3.3.3.9. Flexibility

Like any other Open Source Software, Xindice provides great flexibility by making the code available to the user or developer. This means that the code can be modified to allow the product to handle any situation not previously considered.

3.3.4. eXist

3.3.4.1. Overview and Features

eXist is a proprietary database developed by Wolfgang Meier [63] at the Technical University in Darmstadt, Germany which uses a model-based storage strategy for storing XML documents in the database system. Like Xindice, eXist XML documents are stored in a collection. They are managed in hierarchical collections, comparable to storing files in a file system. Inside the database, collections can contain child collections. In another words, collections may be arbitrary nested.

eXist provides schema or data definition as an option. Documents are not required to have an associated schema or document type definition (DTD). eXist automatically builds indices on the incoming XML documents to be stored.

eXist also treats an XML document as a DOM tree. To speed up query processing, it uses a numerical indexing scheme, which supports quick identification of structural relationships between nodes, such as parent- child, ancestor- descendant, and so on. Furthermore, the indexing is applied to all nodes in the document, including elements, attributes, text, and comments. By default, eXist creates full text indexing over all text and attributes values.

Unlike Xindice, which allows for the manual creation of indexes, eXist indexes are created automatically for all elements and attributes and are managed by the database engine.

Like Xindice, eXist also can run on a Windows, UNIX, and Linux platform. Windows platform is the only one that will be considered in this thesis.

XPath is supported for querying XML documents stored in a collection. In addition to this, eXist extends the standard XPath to efficiently process full text queries, including keyword searches. This allows users to query a distinct part of the collection hierarchy or even all the documents contained in the database. In eXist, queries are executed at the collection level, as for Xindice.

For updating XML documents, eXist uses Xupdate, which is a standard proposed by the XML:DB initiative. In eXist, documents may only be updated as a whole.

Like Xindice, eXist also provides support for the vendor independent XML:DB API which comes with DOM and SAX support.

Apart from the previously mentioned API, eXist also supports other APIs including: XML-RPC, a REST-style Web Services API, SOAP, and WebDAV. These APIs are beyond the scope of this thesis.

3.3.4.2. Functionality

As in Xindice, basic functionalities are also well supported by eXist. In eXist, information is stored in XML format. For working with its database contents, eXist supports Xpath as its query language. This language is used to query XML data stored inside the database, and it also supports XUpdate as its update language.

3.3.4.3. Market Share

eXist also experiences the shortcoming discussed in 3.3.3.3.

eXist is fairly popular. Referring to Freshmeat [58] statistics, eXist has a popularity of 1.94% and its vitality is 0.07%. eXist is classified amongst popular native XML databases by [60, 61]. Considering the feedback from the mailing list, many people seem to be happy working with eXist.

3.3.4.4. Support

3.3.4.4.1. Documentation

eXist's documentation size is small compared to Xindice. Nevertheless, it is still informative, accurate and its quality is good as well. Unlike Xindice, eXist's documentation is updated frequently.

3.3.4.4.2. Installation

eXist documentation provides guidance to the user on how to set up and build the environment and how to install the server. Referring to our experience with Xindice, we find the eXist installation process less challenging than Xindice's one.

The eXist database server offers three modes of operation: standalone, embedded, and servlet modes. The embedded mode is used here. The embedded mode allows the database to run in the same virtual machine as the client application.

3.3.4.4.3. Mailing List

Unlike Xindice, eXist support seems more proactive in providing help, responding to queries and suggestions on its mailing list. This is a positive sign for the potential of this product.

3.3.4.5. Maintenance

Maintenance for eXist products appears to be superior when compared with Xindice. As a result, eXist appears to be more actively developed than Xindice. Because of its development, there are more activities in the eXist mailing list and also it has more users. Users tend to have interest in a product that offers adequate help.

3.3.4.6. Scalability

eXist database is currently best suited for applications dealing with small to large collections of XML documents.

eXist's scalability will be further looked at in chapter 5.

3.3.4.7. Usability

eXist comes with both a Graphical User Interface (GUI) and a command line interface. These two are designed to serve the same purpose, which is to perform the administration of the database. It is the users' responsibility to choose the tool that suits them from these alternatives. Most of the time, the choice is based on one's ability to work with the technology. Obviously, those who want to remember the eXist command line syntax will choose the command line interface and those who do not want to remember it will go for the Graphical User Interface.

3.3.4.8. Security

In eXist, the database engine supports basic user authentication and access control (read, write and update permission). As with Xindice, eXist also lacks transaction control.

3.3.4.9. Flexibility

As discussed in section 3.3.3.9, eXist also supports great flexibility.

3.3.5. dbXML

3.3.5.1. Overview and Features

Like Xindice, dbXML is also a proprietary database developed by the dbXML Group that uses a model-based storage strategy for storing XML documents in the database system. dbXML manages XML documents in collections. Collections can be laid in a hierarchical fashion. This is comparable to an operating system directory structure. Collections can be nested as well. The XML documents contained in a collection are schema or DTD independent. In other words, they do not need to be bound by a common schema or DTD.

Although dbXML supports schema-less XML documents, it is also possible to restrict the stored documents so that they match a given schema.

In a database, indexes can be used to optimize retrieval of documents in a collection. Like Xindice, dbXML allows indexes to be created manually by the programmers for enhancing the performance of dbXML queries. It currently provides three different types of indexes. These are: value indexes, name indexes, and full-text indexes.

Indexes are created based on element or attribute patterns as well as full-text.

Similar to Xindice and eXist, dbXML also uses XPath as its database query language. In dbXML, queries are executed against specific collections or documents within a collection. It also supports full text searches.

Like the two previous Open Source Native XML databases products, dbXML can run on a Windows, UNIX, and Linux platform. Additional to these platforms, dbXML can run on Mac OS X, and Solaris.

Resembling the other Open Source databases, dbXML too supports Xupdate technology for modifying the content of XML documents stored inside the database. It uses Xupdate to update the whole document.

Lastly, dbXML does have support for the vendor independent XML:DB API as well. For completeness sake, we would like to indicate that beside XML:DB API, dbXML also supports three others APIs. These are: the direct API, client API, and Web Services. Although these are mentioned here, they will not be considered for further discussion.

3.3.5.2. Functionality

Similar to Xindice, and eXist, dbXML also provides excellent support for basic functionalities. Data are stored in their native format inside the database. dbXML also uses different techniques for managing its database contents. Xpath is used for querying XML data stored inside the database, and for updating its data, dbXML supports XUpdate.

3.3.5.3. Market Share

dbXML faces the same problem as highlighted in section 3.3.2.3, similar to the two previous Open Source databases already discussed.

Looking at the figures provided by Freshmeat [58], dbXML has 1.01% popularity and its vitality is 0.01%. In [60, 61], dbXML is not classified amongst popular native XML databases. Referring to the feedback from the mailing lists, not much has been said about dbXML.

3.3.5.4. Support

3.3.5.4.1. Documentation

dbXML has the smallest documentation set available of the four databases. It contains sufficient information regarding the product and its related technologies; but it does not provide many examples.

In spite of the small amount of information available, dbXML documentation is not updated on a regular basis. Like Xindice, it is short of a comprehensive installation guide.

3.3.5.4.2. Installation

dbXML has the smallest documentation, it has a quick installation guide which is not very clear. It explains to the user step by step about how to set up the environment and how to install the dbXML server. The [illegible]on contained in that guide makes the installation process easier for the user.

The dbXML database server supports both embedded and client/server scenarios. Like eXist, for dbXML also we chose to use the embedded mode.

3.3.5.4.3. Mailing List

Resembling Xindice, dbXML support seems to be incredibly slow in offering help, replying to queries and suggestions on its mailing list. This is not a promising sign for this product.

3.3.5.5. Maintenance

dbXML products' maintenance appears to be very slow compared to Xindice. Since the dbXML main developer, Tom Bradford, got a new job, this has affected the dbXML development; it seems to have stopped. There have been debates on the dbXML's mailing list about who's willing to take on the main developer's responsibility to continue this project.

3.3.5.6. Scalability

dbXML provides a simple way to store and manage large numbers of XML documents.

dbXML's scalability will be further analyzed in chapter 5

3.3.5.7. Usability

Comparable to eXist, dbXML also provides users with two options: Graphical User Interface (GUI) and command line interface. These two tools allow the users to perform all the basic database administration functions. It is up to users to decide which of these two alternatives to use. Most of the time, the choice will be based on the user's familiarity with the technology. Naturally, the command line interface will be the best choice for users who have learned by heart the dbXML command line syntax whereas users who are not confident with the command line will definitely go for Graphical User Interface option.

The dbXML command line includes a help system that can guide its user through using these commands.

3.3.5.8. Security

dbXML also supports basic user authentication and security management. Unlike Xindice and eXist, it supports transactions control mechanisms.

3.3.5.9. Flexibility

Just like previously mentioned databases, dbXML as well supports flexibility.

3.3.6. Berkeley DB XML

3.3.6.1. Overview and Features

Berkeley DBXML is a key-value database developed by SleepyCat Software, which uses a text-based storage technique for storing XML documents in the database system. It relies on the Berkeley DB database library for basic functionalities (e.g. update, and concurrency control). XML documents are directly stored in the database in their native formats. A *container* is the logical storage unit of the XML document. It can contain one or more XML documents. Each document stored in Berkeley DB XML can have metadata attributes associated with it. XML documents are not required to have a DTD or conform to a prescribed schema.

Unlike the three previously discussed Open Source database products, Berkeley DB XML has no concept of collection hierarchy.

Like Xindice, and dbXML, Berkeley DB XML also supports the manual creation of indexes. In order to enhance the performance of Berkeley DB XML queries, programmers can create indices for all documents in a container. Each container supports multiple indexes. An index can be built following four different types of information.

These are: path types, node types, key types, and syntax types. The idea behind this is to enable fast look up. An index can be added, deleted or replaced.

Similar to those Open Source Native XML databases previously discussed, Berkeley DB XML also can run on Windows, and on the Linux platform. Additional to these platforms, like dbXML, Berkeley DB XML can run on Solaris platform as well.

Berkeley DB XML queries are performed using the XPath language. To retrieve XML documents stored inside the database, XPath queries are issued against the DB XML containers in which the data resides.

For updating XML documents contained in the database, Berkeley DB XML supports XmlModify. XmlModify is a programmatic API, which has a similar functionality to XUpdate.

Like those databases previously discussed, Berkeley DB XML also provides support for the vendor independent XML:DB API. Besides the XML:DB API, Berkeley DB XML also supports XML-RPC.

As previously mentioned, XML:DB API is the only API that will be considered for the scope of this thesis.

3.3.6.2. Functionality

As with the three Open Source Native databases discussed earlier, Berkeley DB XML provides good support for basic functionalities. It stores data in the repository in XML format. It makes use of Xpath technique for querying XML data stored inside the database, like the other three databases do. Unlike them, Berkeley DB XML does not provide support for XUpdate. Rather it uses XmlModify for updating data stored inside the database. The readers' attention is drawn to the fact that this thesis does not aim to compare these two update techniques.

3.3.6.3. Market Share

Like the three aforementioned Open Source products, Berkeley DB XML faces the same difficulty talked about in section 3.3.3.3.

The figures provided by Freshmeat [58], indicate that Berkeley DB XML has 1.14% popularity and 0.09 % vitality. Like dbXML, [60, 61] do not classify Berkeley DB XML amongst popular native XML databases. Looking at the mailing lists' feedbacks, positive comments have been received from people who have worked with Berkeley DB XML.

3.3.6.4. Support

3.3.6.4.1. Documentation

Berkeley DB XML has very good documentation available in contrast to the earlier mentioned databases. It is clear and contains an adequate amount of information about the product and its related technologies. It also contains many examples.

3.3.6.4.2. Installation

The documentation supplied with Berkeley DB XML gives clear instructions for installing the database. These are straightforward and easy to understand. This makes the Berkeley DB XML installation process very easy.

As with the previous databases, the Berkeley DB XML database server supports the embedded mode.

3.3.6.4.3. Mailing List

Berkeley DB XML support gives the impression of being more proactive in providing help, responding to queries and suggestions on its mailing list. This is a positive indication for the potential of this product.

3.3.6.5. Maintenance

Berkeley DB XML products' maintenance has proved to be very regular. As a result, Berkeley DB XML appears to have more ongoing development than the other three Open Source databases.

3.3.6.6. Scalability

Kimbro Staken [64] claims that Berkeley DB XML is designed around storing large document collections. It can easily hold gigabytes of XML data.

Berkeley DB XML's scalability will be further inspected in chapter 5

3.3.6.7. Usability

Database administration of Berkeley DB XML is accomplished from the command line using Berkeley DB XML commands. This command line allows the performance of all the basic administration functions that would be expected.

Before starting to use the Berkeley DB XML command interface, one must understand the Berkeley DB XML command line syntax and also remember the commands every

time there is the need to perform a task. This is a challenge that Berkeley DB XML users need to face.

Like Xindice, Berkeley DB XML does not have a Graphical User Interface (GUI).

3.3.6.8. Security

Berkeley DB XML does not have basic user authentication. It uses Berkeley DB security and transaction features. Berkeley DB XML and dbXML are the only ones that have transaction control.

3.3.6.9. Flexibility

Comparable to all others Open Source products, Berkeley DB XML too supports flexibility.

3.3.7. Comparison

While the chosen Open Source Native XML databases products discussed earlier are slightly different, they share almost the same key features, although maybe the way in which these features are supported varies from one database to another.

Table 3.1 gives an overview of some key features and platforms supported by the chosen databases.

Developer	Product	Main Features	Platforms
Apache Software Foundation	Xindice	XPath, XUpdate, XML:DB API, Indexes	Windows, Unix, and Linux
Wolfgang Meier	eXist	XPath, XUpdate, XML:DB API, Indexes	Windows, Unix, and Linux
dbXML Group	dbXML	XPath, XUpdate, XML:DB API, Indexes	Windows, Mac OS X, Solaris, Unix, and Linux
Sleepycat Software	Berkeley DB XML	XPath, XmlModify, XML:DB API, Indexes	Windows, Solaris, and Linux

Table 3.1 key features and platforms comparison

Table 3.1 above gives a brief review of the main standard technologies and platforms. As one can see, all the NXDs products in the Table 3.1 can run on the Windows and Linux platforms. Additional to that, Xindice, eXist dbXML can run also on UNIX platform but Berkeley DB XML is the only one that does not run on UNIX. Furthermore, dbXML and Berkeley DB XML can also run on Solaris platform but both Xindice, eXist can't. Lastly, dbXML is the only one that can run on Mac OS X.

All the NXD products that are listed in the Table 3.1 support XPath. This technology is used by NXDs to query XML data contained inside the database. It is a query language standard proposed by the W3C.

Indexes are also used by all the NXD products shown in Table 3.1. Although indexing is supported by all these databases, the indexing techniques used vary from one product to another. For example, eXist uses a full index technique, which means that everything inside the database is indexed. Also it supports an automatic indexing technique. This is done by the database itself. On the other hand, the other three databases: Xindice, dbXML, and Berkeley DB XML support a manual indexing technique where it is the

responsibility of the developer to specify which elements need to be indexed. Also, unlike eXist, they use a partial index technique. No matter what indexing technique is used, all the databases use indexes for a common purpose, which is to improve the speed of queries.

All these databases also provide support for XML: DB API in which various methods are supplied for connecting and executing operations on the database.
As shown in Table 3.1, not all these database products support Xupdate as their update language. For updating its data, Berkeley DB XML uses XmlModify which is a proprietary update language developed by Berkeley DB XML. The other three (Xindice, eXist, and dbXML) use Xupdate which is a standard update language developed by XML:DB project initiative. The two mentioned update techniques differ in their way of doing the update operations. Some force the entire document to be retrieved in order for changes to be made, as for example in the case of Berkeley DB XML, while the XUpdate standard allows users to modify document fragments.

For a summary comparison of attributes used for evaluating the chosen Open Source Native XML database products, see Table 3.2 in Appendix A- attributes comparison.

All the Open Source Native XML Database products listed in Table 3.2 provide good support for basic functionalities that are of interest here (e.g. insert, update, and query). They also provide very good flexibility.

As far as market share goes, according to the statistics provided by Freshmeat, eXist seems to be the one which is most popular and also has a good vitality. Xindice comes in second place in terms of its popularity but it does not have a good vitality. Berkeley DB XML occupies the third place in terms of popularity but it has a very good vitality compared to the other two products. Looking at dbXML's popularity, it seems to be the least popular compared to the other three products but it has the same vitality as Xindice.

In terms of support and maintenance, Berkeley DB XML seems to provide very good support and maintenance for its product compared to the others databases. eXist also provides good support but not as good as Berkeley DB XML. eXist support and maintenance seems to be much better than that provided by Xindice and dbXML. Xindice and dbXML support and maintenance seem to be slow. In comparing these two, dbXML seems to be the slowest. In fact, it seems even to have stopped.

Looking at the chosen Open source XML database products' usability, some products present the users / developers with the ability to choose between a Graphical User Interface and Command line Interface. Examples of these products are eXist and dbXML. Others products, like Xindice and Berkeley DB XML provide only the Command line Interface. This means that the users/ developers need to remember the commands every time they need to use any of these two.

Regarding security issues, eXist and dbXML provide basic security features. They support basic user authentication and sect[illegible]agement. At the same time, they lack a transaction control mechanism. Xindice and Berkeley DB XML lack basic user authentication. Xindice further, lacks, in short, a transaction control mechanism. On the other hand, Berkeley DB XML and dbXML support a transaction control mechanism. Furthermore Berkeley DB XML uses Berkeley DB security and transaction features.

Lastly, looking at the storage issues, all these Open source XML database products can provide a schema-less storage for XML documents. Some manage XML documents in collections fashion, whereby collections can contain subcollections or child collections. Also these collections can be hierarchical. These kinds of databases include: Xindice, eXist, and dbXML. On the other hand, Berkeley DB XML stores XML documents in a container and also it lacks the concept of a collection hierarchy.

3.4. Summary

This chapter discussed the evaluation methodology. The following attributes were found necessary and important: functionality, market share, support, mailing list, maintenance, performance, scalability, usability, security, and flexibility. An explanation of each of these attributes was provided.

The chapter also introduced the criteria that were applied in selecting the Open Source XML database products that were used for this thesis. Based on these criteria, the following products were selected: Xindice, eXist, dbXML, and Berkeley DB XML. The previous performance studies done using any of the studied Open Source XML databases were highlighted, and features of each of the mentioned products were briefly discussed as well. Also each of these products was evaluated using the evaluation attributes mentioned previously.

A comparison of some key features, platforms and attributes of the chosen Open Source Native XML database products was done. As shown in Table 3.1, which provided a summary comparison of the key features, including platforms supported by the chosen products, these products have similarities. The chosen Open Source native XML databases can run on the Windows platform and all support XPath, XML:DB API, indexes. The support or techniques used to implement these features might vary from one product to another, because each product is based on different architectures. Despite this, these Open Source products use the previously mentioned features for the same purposes. For example, they all use XPath as a query language, XML:DB API is used for connecting to the database, and indexes are used for speeding query processing. For updating data contained inside the database, these products do not support the same technology. For example Xindice, eXist, and dbXML support XUpdate, but Berkeley DB XML does not support this, and instead it uses XmlModify.

Table 3.2 in Appendix A, compares attributes of the products that were discussed in this chapter. All these products provide good support for our basic functionalities, and can also provide good flexibility. The products like eXist and Xindice are more popular than others like Berkeley DB XML and dbXML. Furthermore, Berkeley DB XML has a greater vitality than eXist and Xindice. Berkeley DB XML also seems to provide very good product maintenance, followed by eXist. Xindice and dbXML seem to provide poor maintenance for their products. A number of questions are being asked about the future of these products. At least, Xindice is still struggling to remain alive but dbXML seems as to have stopped.

For the products' usability, eXist and dbXML provide both Graphical User Interface and command line interface for their user and therefore it is up to the user to choose the one he / she is comfortable with. On the other hand, Berkeley DB XML and Xindice only have a command line interface.

Concerning security issues, products such as eXist and dbXML provide user authentication and security management whereas Berkeley DB XML and Xindice do not. Also Xindice, eXist, and dbXML lack support for transaction control mechanisms. In turn Berkeley DB XML uses Berkeley DB security and transaction features.

Chapter 4

EXPERIMENTAL EVALUATION OF THE CHOSEN OPEN SOURCE NATIVE XML DATABASE PRODUCTS

4.1. Introduction

Chapter 3 briefly introduced each of the four selected open source native XML database products. Additional to this, some basic features and attributes of the selected products that were judged to be relevant to the scope of this thesis were identified and discussed. Finally, a comparison of some of the key features and attributes of all these products was also done.

This chapter explains the motivation, aim, and hypothesis tested by the experiments. These tests carry out a set of experiments designed to evaluate the metric performance of the chosen open source database products in executing the database basic operations. The chapter also describes the methodology and test suite setup used for testing performance of these open source products.

Chapter 4 will also elucidate the motivation for choosing the present testing environment and it furthermore provides a detailed description of each performance test operation that will be conducted.

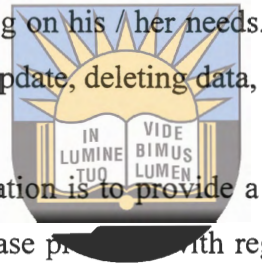
4.2. Motivation

As previously pointed out, there are currently a variety of Open Source Native XML database products available on the market. At the same time, as was seen in the preceding chapter, these chosen Open Source products have a low percentage in terms of popularity and vitality and this gives a clear indication that the acceptance of this technology in the database market is still an uncertain issue. There is a struggle for the survival of this technology and also this raises questions about this technology's maturity. Maybe one of the reasons that might explain this could be the fact that many people are still unaware of these Open Source products' availability, capabilities, functionalities, and performances.

The focus here will be on the performance issues of the chosen Open Source Native XML database products introduced in chapter 3.

4.3. Aim

Databases have proved to be an extremely valuable technology for handling large amounts of data. Looking at it from the viewpoint of the database user, when working with a database there are always some kind of basic operations that the user has in mind and would like to perform depending on his / her needs. These operations can be one of the following: data insertions, data update, deleting data, or querying data.



The aim of this experimental evaluation is to provide a metric performance comparison of Open Source Native XML database products with regards to handling basic database operations like storing, querying, updating, and deleting XML documents. The metric used for the performance comparison is the response time for executing instructions to carry out a particular database task, that is the amount of time taken to store, query, update, or delete XML data. At the same time, it is necessary to understand how a particular operation is completed because this will help in explaining the performance test results.

4.4. Hypothesis

The hypotheses tested by these experiments is that (1) it is possible to perform the following operations: store, query, update, and delete data when working with Open Source Native XML database products, (2) sometimes the performance of these primitive operations can be affected by the size of the database, (3) using database indexing

techniques on data stored inside the database can be beneficial to the improvement of the performance of a particular operation, and (4) some XML databases perform efficiently on certain operations but not on others.

4.5. Methodology and Test suite set-up

4.5.1. Timing Methodology

There are a number of methodologies available that can be used to measure the performance of a database and these come in two groups: external performance measurement and internal performance measurement. The external performance measurement is used when measuring the response times of various requests on different databases. On the other hand, the internal performance measurement is useful when measuring the time spent in the various components of the database system.

The two performance methodologies explained in the previous paragraph have the same objective, which is to collect timing data. This timing collection is done by placing check points within the database system. In [65], a check point is defined as a procedural invocation inserted into the database system's flow of control in order to call the performance measurement routines for the collection of timing data.

Techniques used when placing check points within the database system may be different for these two approaches. The comparison of these two approaches is outside the scope of this thesis.

The methodology used for measuring the response time is similar to the external performance measurement approach. As pointed out earlier, the aim of the experiment is to compare the metric performance of Open Source Native XML database products in handling basic database operations. In other words, in these experiments, our major

concern is the response time needed by an Open Source Native XML database product to execute an instruction for performing a particular database basic task such as insertion, deletion, update, and query. The response time captured calculates the full instruction execution process of a particular task, from the starting of a task's execution instruction, until the end of that particular instruction.

In order to capture the response time for each of these basic tasks, two check points were placed in purpose-written benchmark programs to collect timing data. The first check point is placed to record the time before the task's execution begins and the second one is placed to record the time after the task's execution is completed. The response time is calculated by subtracting the timing recorded at the second check point from the timing recorded at the first check point. The task's execution instruction is repeated n times, and the average is taken.

The code segment below in Figure 4.1 illustrates how the response time is captured, in milliseconds.

```
Date date = new Date();
t1 = date.getTime();

   for(int k = 0; k < n; k++){
        [...]
                              }
date = new Date();
t2 = date.getTime();

total = t2 - t1;

p.println(Size + "," + ((double)(total/1000)));
```

Figure 4.1: Response Time

In Figure 4.1 above, t1 represents the first check point which is placed before the task's instruction begins to execute, the for loop indicates the reiteration of the instruction that

must be executed, where n indicates the total number of times that a particular task's instruction needs to be repeated. [...] inside the for loop denotes the task's (e.g. insertion, deletion, modification, and delete) instructions that need to be executed. This repetition is done because the time for only one execution is extremely small and may lead to misleading results. The second check point that collects the timing after the task's instruction has been executed n times is represented by t2 in Figure 4.1. The response time which is the difference between the timing recorded at the second check point (t2) and the timing recorded at the first check point (t1) is represented by total in Figure 4.1 above. In this context, this total indicates the time required to repeat the instruction n times. Furthermore, in order to get the response time in seconds, the total is divided by a thousand. The different response times recorded are printed in an output file. Those response times contained inside the output file will be used for plotting the database performance comparison graphs.

4.5.2. Experimental set-up

In the subsections 4.5.2.1 and 4.5.2.2 below, we provide more details about the hardware and software environments that were utilized to carry out the various kinds of performance tests that will be described in section 4.5.3.

4.5.2.1. Hardware Environment

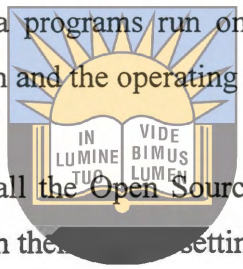
The hardware used to carry out all these experiments consists of a single machine. This single machine was running as a client / server. The following are its specification; it includes an Intel Pentium 4 processor with 1.80GHz CPU, 512 MB RAM and 40 GB hard disk.

4.5.2.2. Software Environment

The machine used in the test environment runs Microsoft Windows 2000 professional as Operating System. The service pack currently installed on this machine is Microsoft Windows 2000 Service Pack 4.

The Open Source Native XML databases for which the metric performance is being measured in these experiments include Berkeley DB XML version 1.1.0-Win32, dbXML version 2.0, Apache Xindice version 1.1b4, and eXist version 0.9.2.

The Java virtual machine (JVM) used in the test environment is Java 2 SDK, Standard Edition Version 1.4.0. All the Java programs run on a JVM. The JVM serves as an intermediary between the application and the operating system (OS).



The different memory settings for all the Open Source Native XML database products tested including the JVM were left in the[illegible]settings.

The XML data set used to study the metric performance of the Open Source Native XML databases is a fragment of an XML document containing inventory information about an automobile's parts that was taken from a [66] example. Its contents can be seen in Listing 4.1.

```
<?xml version="1.0"?>
<parts>
  <part sku="101">
    <desc>Ball Bearing</desc>
    <maker>S.K.F.</maker>
    <instock>Yes</instock>
    <price>$20.00</price>
  </part>
</parts>
```

Listing 4.1: Fragment of XML document used

As shown in Listing 4.1, the root element of this XML's fragment document is <parts>. Furthermore <parts> element contains part <part> which contains different information about a variety of automobile products.

A <part> element has one attribute <sku> (product number) that describes the <part> element. Additional to the attribute <sku>, the <part> element also has four other elements (children). For example, the element <desc> which specifies the part's description, the element <maker> for specifying the part's maker, the element <instock> for indicating the part's availability in stock, and finally the element <price> that indicates the part's price.

On the basis of the information shown in Listing 4.1, the Java program (CreateXML.java) was written that takes in an integer number as a parameter and generates an XML document. The integer number that has been passed to the program as input represents the total number of the <part> elements that should be generated inside the created XML document. Furthermore, each Sku attribute of the created part element will have a randomly generated number. Each of these numbers represents the product number or sku for every part element contained inside the newly generated XML document. The full code for generating nodes inside the XML document can be found in Appendix B.

4.5.2.3. Rationale for choosing to use a single machine

For the context of this set of experiments, a performance comparison of Open Source Native XML databases with regard to handling some simple database operations is being provided.

In order to accomplish this, the performance tests were conducted in a closed environment. This approach was taken to isolate others factors that might influence the final timing results like network overhead, communication costs (e.g. HTTP, CORBA, XML RPC, etc.) or transformations (e.g. XSL) of the output.
For this rationalization, it was decided to carry out all the experiments in a single-user machine. Even though using a single machine seems to be the best option for carrying out all the experiments in order to avoid the timing being corrupted, it should not be overlooked that the load on the system (machine) can still slightly influence the results. This factor is very small and has very litt[illegible]on the final result.

4.5.3. Description of the tests

The set of test performance operations that will be used to evaluate the metric performance comparison of the studied Open Source Native XML databases has been defined.

XPath and XUpdate, which were introduced and discussed in sections 2.2.4 and 2.2.5, are respectively the two technologies that were used for formulating these metric performance tests operations. XPath was used for querying data whereas XUpdate was used for inserting, modifying, and deleting data.

The following subsections give details for each of the different types of basic database operation that will be considered for testing the selected Open Source XML database

products' performance. A list is provided of all test performance operations that will be considered.

4.5.3.1. Storage

This involves inserting information or XML data into a XML database. The type of information that has been stored into the database will be different depending on the database user or the database administrator's needs. This information can be a whole XML document, a block of XML data, an element, or an attribute.
In this context, the focus is on whole XML document storage operations, sometimes also referred to as bulk insertion.

The bulk insertion operations that are included in this test performance are not only to study how well an Open Source XML database can handle data storage but also to look at the impact the different XML documents that are being stored in the database can have on the database performance. Therefore, for achieving the target the storage performance tests will be carried out on a certain number of generated XML documents.

In order to get a reasonable comparison, three different sized XML documents are generated using CreateXML.java as explained in section 4.5.2.2. The three generated XML documents include a small XML document with a size of 6.37 KB, a medium XML document with a size of 19.00 KB, and a large XML document with a size of 126 KB. Furthermore, these XML documents differ in the number of part <*part*> elements that have been generated inside each. The small XML document contains 50 generated part elements, the medium XML document contains 150 generated part elements, and the large document XML contains 1000 generated part elements.

Each of these three generated XML documents will be stored in different databases. We will further study and compare the performance of the studied XML databases when storing each of these documents.

4.5.3.2. Queries

Queries allow the database users or administrators to select or extract XML data contained in a database so these can be viewed, analyzed, or manipulated. Depending on one's needs, the type of XML data that are to be selected or extracted will differ. These can be simple element, attribute, portion of a document, or the whole document.

There are a number of issues that also need to be considered when formulating or designing XML queries. The XML queries that were formulated for use in the performance tests, involved the following aspects: *query target, query path, conditional path, and operators.* Beside this it was also decided to include the logical operators such as "AND", "OR", and "NOT" because they are very common in querying databases.

In the previous section 4.5.3.1 XML documents were stored in the database. These stored XML documents will also provide a set of documents for all queries performance tests. In other words, the XML data to be queried or carry out the queries performance tests are the XML documents that were inserted in the database in the previous subsection.

The following subsections introduce and describe the syntax for each of the XML queries that were formulated.

Queries by query target:

The targets of XML queries can vary according to a particular query's desired goal or aim. Whatever the desired goal might be, the query target will usually be one of the following: the whole XML document or specific XML data contained inside an XML document. This specific data can be encoded in either elements, or attributes. For the purpose of these performance tests, the queries that were designed are intended to measure the performance of databases in processing XML queries.

Q(1a) Querying a whole XML document : To query all parts contained inside an XML document, the XPath syntax is */parts*.

Q(1b) Querying elements: To query all part elements contained inside an XML document that are children of parts, the XPath syntax is */parts/part*

Additional to querying elements, it has been decided to include queries for attributes in the query performance tests. The reason for doing so is to compare the attribute query with other queries (e.g. element). This comparison will assist in determining whether or not to use either elements or attributes to encode XML data for a particular database.

Q(1c) Querying attributes of an element: To query all sku (product number) attributes that are children of a part, the Xpath expression is */parts/part/@sku*

Conditional Queries

Unlike in the previous case, in some other cases, the formulation of an XML query can also depend on several conditions such as the position inside the XML document for the element or attribute that needs to be selected or extracted, the type of XML data that is encoded in either elements or attributes, etc.

The main reason that query path and conditional path have been included in the performance tests is to measure how the depths of both affect the performance of XML queries.

Q(2a) Querying a whole XML document: To query all the <sku> (product numbers) attributes that are children of <part> and having value greater than 2414, the Xpath expression is */parts/part[@sku > '2414']*

Queries using the position occupied by an element

Q(3a) Querying elements: To query the description of the part which occupies position 1 within the hierarchy of node elements contained inside the generated XML document, and whose parent is parts, the Xpath expression is */parts/part[1]/desc*

Q(3b) Querying elements: To query the description of the part which occupies position 75 within the hierarchy of node elements, and whose parent is parts, the Xpath expression is */parts/part[75]/desc*

Q(3c) Querying elements: To query the description of the part which occupies the last position within the hierarchy of node elements, and whose parent is parts, the Xpath expression is */parts/part[last()]/desc*

Queries using value of an attribute

In the queries below, the numbers 3119, 7199 and 8834 are the values of attributes <sku> (product number) contained inside the document. The first (3119) is near the beginning of the document, the next (7199) in the middle and the last (8834) at the end.

Q(3d) Querying attribute: To query the description for the part whose attribute is equal to 3119, the Xpath syntax is */parts/part[@sku='3119']/desc*

Q(3e) Querying attribute: To query the description for the part whose attribute is equal to 7199, the Xpath syntax is */parts/part[@sku='7199']/desc*

Q(3f) Querying attribute: To query the description for the part whose attribute is equal to 8834, the Xpath syntax is */parts/part[@sku='8834']/desc*

Queries by operators

Q(4a) AND: To query the part which is a child of parts and whose sku (product number) is equal to 7064 and maker equal to S.K.F, the Xpath expression is */parts/part[@sku='7064' and maker ='S.K.F']*

Q(4b) OR: To query the part which is a child of parts and whose sku (product number) is equal to 7199 or sku is equal to 3191, the Xpath expression is */parts/part[@sku='7199' or @sku ='3191']*

Q(4c) NOT: To query the parts that are children of parts and whose sku (product number) is not equal to 5604, the Xpath expression is */parts/part[@sku!='5604']*

Missing information

Missing information refers to the informa[illegible]oes not appear in the document.

Q(5a) Missing element: We try to query the price of a part element whose hierarchical position is equal to 178, the Xpath expression is */parts/part[178]/price*

Q(5b) Missing attribute: Here, we are trying to query the description of a part whose sku (product number) is equal to 107, the Xpath expression is */parts/part[@sku='107']/desc*

4.5.3.3. Modifications

Modification operation refers to different kinds of amendments that can occur to information contained in XML documents. This includes altering, modifying stored data, or adding new XML data into the existing XML document. The new information being added can be a new element, attribute, or block of XML data.

In the context the performance tests, diverse reasons that can influence the performance of modification operations will be examined; for example the number of elements or attributes being modified in the database.

Investigating the impact that altering XML data at different locations within XML documents can have on the database performance when working with databases that contain different XML documents sizes will also be considered. For the modification test performances, the following locations within the XML document will be considered *beginning, middle,* and *end of the XML document.*

It is to be noted that the type of modification operations considered in these performance tests are only those that are regularly done when working with a database. An example of this kind of common modification operation is altering the values of attributes or elements.

The XML documents that were stored in the database in subsection 4.5.3.1 will be used to test the performance of our modification operations.

Below, the XUpdate syntax that is used to alter existing nodes (element or attribute) is provided, with all test descriptions.

Modify the value of an element

M(1a) To modify the value of the price (e.g. from its current value to $500.00) which is a child of the part occupying position 1 within the hierarchy of node elements contained inside the generated XML document, the Xupate syntax is

"<xu:update select=\"/parts/part[1]/price\">" + "$500.00" + "</xu:update>"

M(1b) To modify the value of the price (e.g. from its current value to $700.00) which is a child of the part occupying position 75 within the hierarchy of node elements contained inside the generated XML document, the Xupate syntax is

"<xu:update select=\"/parts/part[75]/price\">" + "$700.00" + "</xu:update>"

(M1c) To modify the value of the price (e.g. from its current value to $540.00) which is a child of the part occupying the last position within the hierarchy of node elements contained inside the generated XML document, the Xupate syntax is

"<xu:update select=\"/parts/part[last()]/price\">" + "$540.00" + "</xu:update>"

Modifying an attribute value

(M1d) To modify the value of the attribute <sku> (or product number) which is a child of the <part> element from 3119 to 65, the Xupdate syntax is

"<xu:update select=\"/parts/part/@sku[.='3119']\"> + "65" + "</xu:update>"

(M1e) To modify the value of the attribute <sku> (or product number) which is a child of the <part> element from 7199 to 509, the Xupdate syntax is

"<xu:update select=\"/parts/part/@sku[. > + "509" + "</xu:update>"

(M1f) To modify the value of the attribute <sku> (or product number) which is a child of the <part> element from 8834 to 2623, the Xupdate syntax is

"<xu:update select=\"/parts/part/@sku[.='8834']\"> + "2623" + "</xu:update>"

4.5.3.4. Deletions

Like the aforementioned, deletion is also one of the important operations when using a database. It consists of removing unwanted data contained inside the database collection. The deleted data can be an element, an attribute, or a block of XML data.

As in the case of modifications, an investigation of the most important factors that can affect the overall performance of the XML database when removing particular data contained in a database will be conducted. These include the number of elements or

attributes to be deleted and also the position of the to-be-deleted elements or attributes in files (XML documents).

In the interest of readers, it is emphasized that, as in the case of those previous discussed operations, only the delete operations that are frequently done when working with a database will be considered in these performance tests.

In the next subsections, a list is provided of the XUpdate syntax that is formulated in order to carry out these deletion performance tests and a short explanation for each one of them is also provided.

Deleting an element

D(1a) To delete the price of the part element, which occupies position 1 within the hierarchy of node elements, the Xupdate syntax is

"<xu:remove select=\"/parts/part[1]/pric

D(1b) To delete the price of the part element, which occupies position 75 within the hierarchy of node elements, the Xupdate syntax is

"<xu:remove select=\"/parts/part[75]/price\"/>"

D(1c) To delete the price of the part element, which occupies the last position within the hierarchy of node elements, the Xupdate syntax is

"<xu:remove select=\"/parts/part[last()]/price\"/>"

Deleting an attribute value

D(2a) To remove the attribute <sku> with value equal to 65 which is the child of the <part> element, the Xupdate syntax is

"<xu:remove select=\"/parts/part/@sku[.='65']\"/>"

D(2b) To remove the attribute <sku> with value equal to 509 which is the child of the <part> element, the Xupdate syntax is

"<xu:remove select=\"/parts/part/@sku[.='509']\"/>"

D(2c) To remove the attribute <sku> with value equal to 2636 which is the child of the <part> element, the Xupdate syntax is

"<xu:remove select=\"/parts/part/@sku[.='2636']\"/>"

4.6. Summary

This chapter explained the motivation, aim, and hypothesis tested by the experiments. It further provided a detailed description of the test environment and timing methodologies used to conduct the performance comparisons. Lastly, this chapter provided a detailed list and explanation of each of the database operations (e.g. storage, query, modification, and delete) that will be carried out for comparing the metric performance of the chosen databases.

Chapter 5

OUTCOMES AND DISCUSSION OF THE EXPERIMENTS

5.1. Introduction

Chapter five starts briefly by discussing some of the previous results obtained by different research groups that conducted similar database performance comparison works. The chapter furthermore presents and discusses the results from the experiments as described in chapter 4. Lastly, a comparison of the various results will be done with the intention of contrasting our research findings with the previous research results, and a conclusion will be made.

5.2. Previous Research Findings

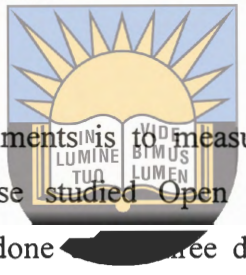
As pointed out in chapter three, some research groups have already done performance comparisons using some of the Open Source native XML databases that we have been investigating in our work, and they obtained results about the performance of these databases. For example, some findings were made by authors [14, 15, 16, 56, and 57] who conducted a performance comparison study of XML-enabled and Native XML databases systems. Other researchers [56, 57] also made some findings when they compared features of some selected Native XML database products.

In the interest of readers, mention is made of the fact that we limit the survey to the findings of the studies that are analogous to the scope of this thesis. In other words, only the previous findings for studies that involved comparing performance of the Open Source native XML databases will be considered.
In what follows, a discussion of some of these previous research findings will be done.

The results obtained from the performance tests conducted by [17] indicate that eXist database outperforms Xindice when querying XML documents stored in the database. On the other hand, the results that they obtained from the insertion performance tests

demonstrate that Xindice database carried out the insertion operation much more quickly than eXist. Similar to the [17] findings, the test results obtained by [18] as well indicate that the performance of Xindice when inserting XML documents in the database was faster than eXist database's performance and, in addition to this, they also claimed that their test findings show that Xindice presents satisfactory results as well for modification, and delete operations. Nonetheless, for the database query operation, their findings demonstrate that eXist surpassed Xindice. This result once more is similar to the query operation findings in [17].

5.3. Experimental Results



The main concern in these experiments is to measure the response time needed to complete a task by each of these studied Open Source Native XML databases. Performance test runs have been done on three different sizes of generated XML documents. The sizes of generated XML documents are 6.37 KB, containing 50 generated part elements, 19.00 KB containing 150 generated part elements, and 126 KB containing 1000 generated part elements. The focus of these performance tests is to investigate the ability of each of the studied Open Source XML databases to handle the basic database operations such as storing, querying, updating, and deleting.

The response time of each of the Open Source native XML databases when executing each of these basic database operations was measured. The instruction part for executing a primitive basic database operation (e.g. store, query, modify, and delete) was repeated n times, and the average time was taken. In the experiment, the time required for connecting to the database, or the time required for creating a collection was not considered as only the basic database performance was of interest. Consequently only the time used by these XML databases to complete a particular operation was the focus.

The experimental results of these metric comparison performance tests are presented in a graphical format in the next subsections. A complete listing of the source code used for the different performance tests (i.e. storing, querying, updating, and deleting), and the tables containing results obtained from the experiments can be found in Appendices B, and C.

For the rest of this chapter, each of the basic operations considered in these performance tests is introduced. The experimental results obtained for each of these basic operations are presented and discussed.

5.3.1. Storage

The first performance test was to store the generated XML documents into the database. Before discussing the storage performance results, there is a need to examine the different internal storage mechanisms available that can be used by Open Source native XML databases to store XML data. This will help in the understanding of how Open Source native XML databases internally organize XML data and will also assist in interpretation or explanation of the storage tests performance results.

There are two approaches that Open Source native XML databases use to physically store collections of XML documents. These are *text-based* and *model-based* storage approaches. These two techniques have already been discussed in section 2.3.3 in chapter 2. To recapitulate what we said earlier concerning the two above-mentioned storage techniques, the *text-based* storage approach stores the entire XML document in text form as flat files, relational BLOBs with processing ability, or proprietary storage (text) format. It also provides some database transaction support in accessing the document. The *model-based* storage on the other hand first models the XML document using a model like the Document Object Model (DOM) to represent an XML document as a tree structure and then maps objects of this representation to a database. How the model is

stored depends on the database. Some databases store the model in a relational or object-oriented database whereas others use a DOM persistent storage model, or a proprietary storage format suitable for their model.

Each of these storage methods uses some kind of a mapping. The tasks of a mapping are to parse an XML document's representation, identify the relevant objects, and issue appropriate database calls to store objects.

Referring back to the experiments, the data sets that were used to test the storage performance are the three XML documents that were generated. These documents contain the same kind of information but they differ in the number of items that were generated and also the randomly generated attribute number of each part item. In Figure 5.1 we show a sample section of the generated XML document


```
<?xml version="1.0"?>
<parts>
 <part sku="7541">
<desc> Ball Bearing </desc>
<maker> S.K.F. </maker>
<instock> Yes </instock>
<price> $20.00 </price>
</part>
 <part sku="30">
  [...]
</part>
  [...]
</parts>
```

Figure 5.1: Sample section of generated XML document

The manner in which the generated XML document shown in Figure 5.1 will be stored into the studied Open Source native XML databases will depend on the storage

mechanisms employed by that particular XML database. This can be either of the two methods introduced previously, namely text-based storage or model-based storage. For this reason it will be significant to have an idea about how these two formats represent XML data. In Figures 5.2 and 5.3 the testing data sets (generated XML document) illustrate the storage representation of these two approaches.

```
<?xml version="1.0"?>
<parts>
 <part sku="7641">
<desc> Ball Bearing </desc>
<maker> S.K.F. </maker>
<instock> Yes </instock>
<price> $20.00 </price>
</part>
<part sku="30">
[...]
</part>
[...]
</parts>
```


Figure 5.2: Text-based storage representation of generated XML documents

The Figure 5.2 above is similar to Figure 5.1. This shows that the text-based representation typically stores whole documents as a unit, without changing any information. In other words, in the text-based storage approach our generated XML documents are stored "as-is".

The next Figure 5.3 shows the technique that the model-based format will use to represent our generated XML documents.

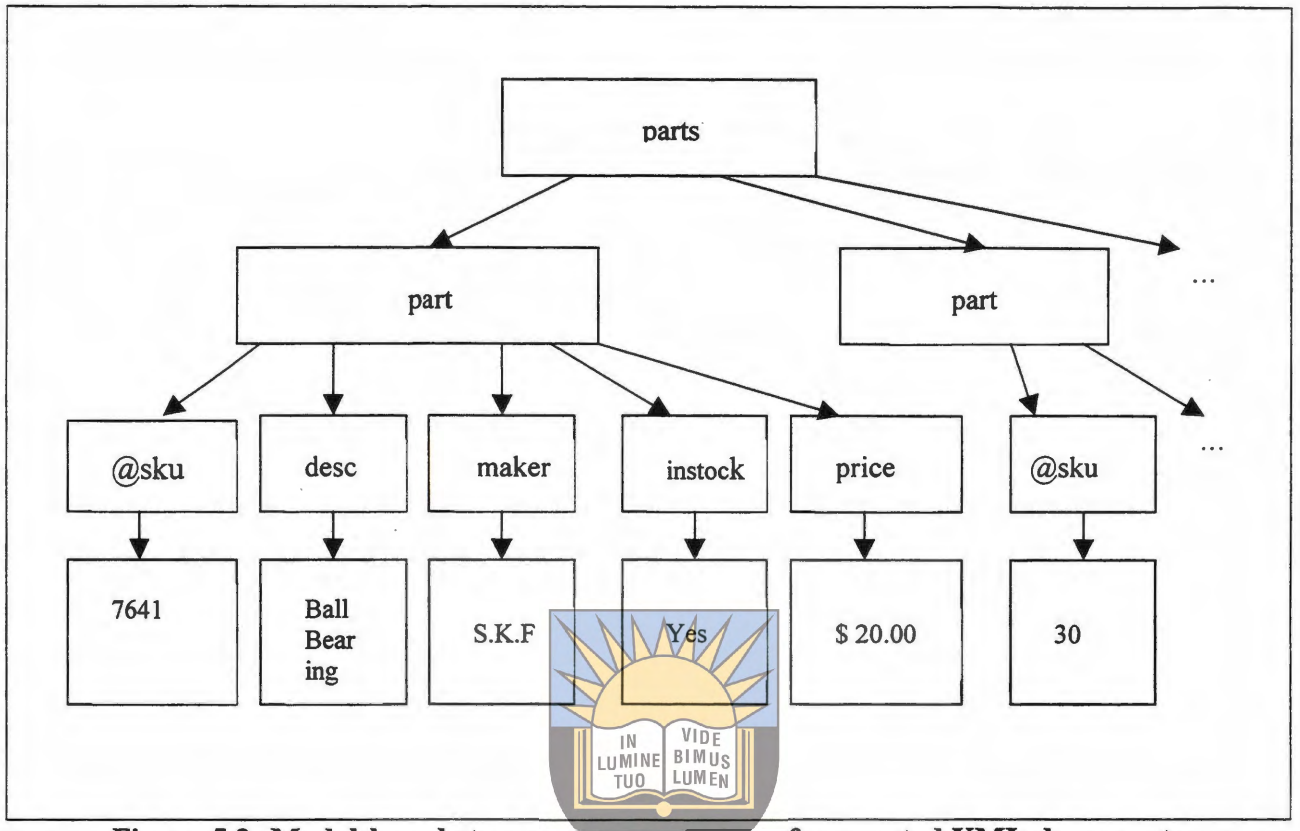


Figure 5.3: Model-based storage re[illegible]n of generated XML documents

Figure 5.3 above illustrates how the model-based storage builds an internal object model of the generated XML documents before storing it into the database. This method views the XML document as a hierarchical structure of nodes as shown in figure 5.3. These nodes can be of several types, such as element, attribute, character data, and so on. An element has a name and may have children. An attribute has a name and carries text. Lastly, character data carries text but has no name. For example, looking at the figure 5.3, the elements are: parts, part, desc, maker, instock, and price. An example of attribute is sku. 7641, Ball bearing, S.K.F, Yes, $20.00, and 30 are character data.

When these two XML data representation approaches as illustrated in Figures 5.2 and 5.3 are compared, it can be seen that text-based representations classically store the whole document, while model-based approaches store trees based on the data's hierarchical arrangement.

After getting insights into the two main storage approaches used by Open Source Native XML databases it is now pertinent to examine the impact that these two techniques can have on the storage capability of the studied Open Source Native XML databases.

5.3.2. Test results for storage

A set of experimental tests were run for comparing the storage capability of the studied Open Source Native XML databases. In all these experiments, the major concern is the response time needed for executing an instruction to carry out a task. A task is defined as a basic database operation that has to be achieved, which in this case is storing XML documents into a database. In other words, in these experiments the focus is on timing the execution of the database storage operation only. The time needed to store an XML document into a collection is measured. The time required for connecting to the database, or time required to create a collection are not included here.

In the next subsections, the results of each of the three generated XML documents that are used for testing the storage performance of the studied Open Source native XML databases are presented.

5.3.2.1. The storage for fifty part elements

Figure 5.4 presents a performance comparison graph of the time taken by the four studied Open Source native XML databases for storing a generated XML document that contains fifty part elements (small.xml) in the database.

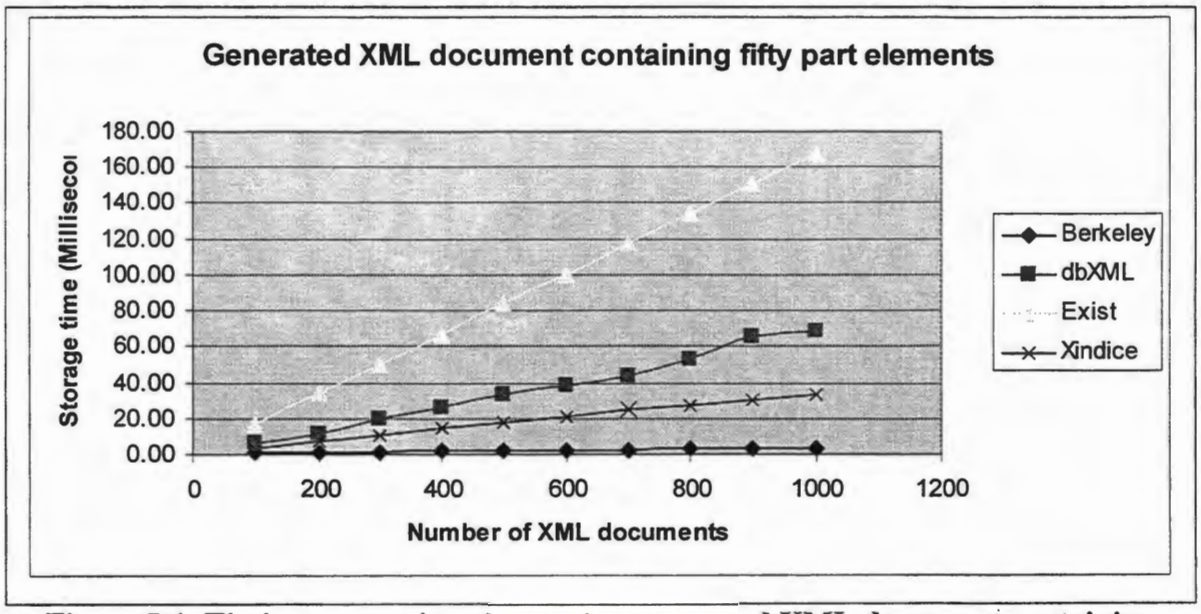


Figure 5.4: Timing comparison for storing generated XML document containing fifty part elements

Prior to interpreting the data shown in the comparison graph above, a short explanation of the information shown on the vertical [illegible] horizontal axis is necessary. Numbers displayed on the horizontal-axis represent the number of generated XML documents that are stored inside that particular database collection. For example, 100 represents hundred generated XML documents, 200 for two hundred generated XML documents, and so on. The numbers on vertical-axis are the various times obtained in milliseconds when storing different numbers of generated XML documents into diverse collections inside the database. For example, the time taken for storing a hundred generated XML documents, the time taken for storing two hundred generated XML documents, and so forth. Lastly, the four lines represent the four studied Open Source native XML databases that are being compared in this storage performance test, namely Berkeley DB XML, Xindice, dbXML, and eXist as shown in the legend.

The results on the comparison graph above in Figure 5.4 reveal that the performance of Berkeley DB XML is much better than the other three, in the sense that its performance shows minor difference along the different generated XML documents. This is attributed to the fact that Berkeley DB XML uses text-based storage. As already mentioned, in the

text-based storage approach, the XML document is stored "as-is" without any conversion. Since Berkeley DB XML uses this text-based storage technique, when storing an XML document it does not need to spend time building a tree representation of that particular XML document. That is why Berkeley has an advantage over the other three when storing XML documents. For example, looking at the comparison graph when inserting from 100 to 1000 xml documents, Berkeley DB XML does show only a very slight change compared to the other three XML databases.

eXist, dbXML, and Xindice make use of the model-based storage approach. Contrary to Berkeley, prior to storing an XML document, they must define an XML document representation model of the document being stored using (Document Object Model) DOM technique, and after this map that XML document's representation model into the database. The mapping itself also depends on the target database model which can be any database model. For example eXist stores the model as persistent DOM objects whereas dbXML uses a relational mapping model, and Xindice uses a proprietary storage format.

Looking at the comparison graph in Figure 5.4, it is obvious that as the number of XML documents increases from 100 to 1000, the storage times for eXist, dbXML, and Xindice also increase. This shows the disadvantage of model-based storage in the sense that when the XML document's size increases it also needs more time for building an XML document representation model or the DOM tree structure for such document.

Like Xindice and dbXML, eXist uses the model-based storage techniques but the results in the comparison graph in Figure 5.4 demonstrate that eXist performs worse than the other three databases. This poor performance is caused by the mapping technique and indexing scheme used by eXist. As mentioned, eXist makes extensive use of the DOM technology. It stores the XML document as persistent nodes objects. Consequently when storing an XML document, eXist builds a node index for the top n level element nodes in the DOM tree. This process can be time consuming, especially when dealing with large XML documents. For example looking at the comparison graph in Figure 5.4, as the number of XML documents increases from 100 to 1000 XML, we can see that the storage

time is also highly increased. This can be explained by the fact that as the number of XML documents that are being stored in the collection increases, creating an index for every single element contained in the XML document in the collection leads to decreasing performance as well.

5.3.2.2. The storage for hundred and fifty part elements

In this subsection, the results of the second dataset that was used in the storage experiments will be presented. This dataset contains more elements than the previous one.

The performance comparison graph in Figure 5.5 shows the time taken by the four studied Open Source native XML databases for storing a generated XML document containing hundred and fifty part elements.

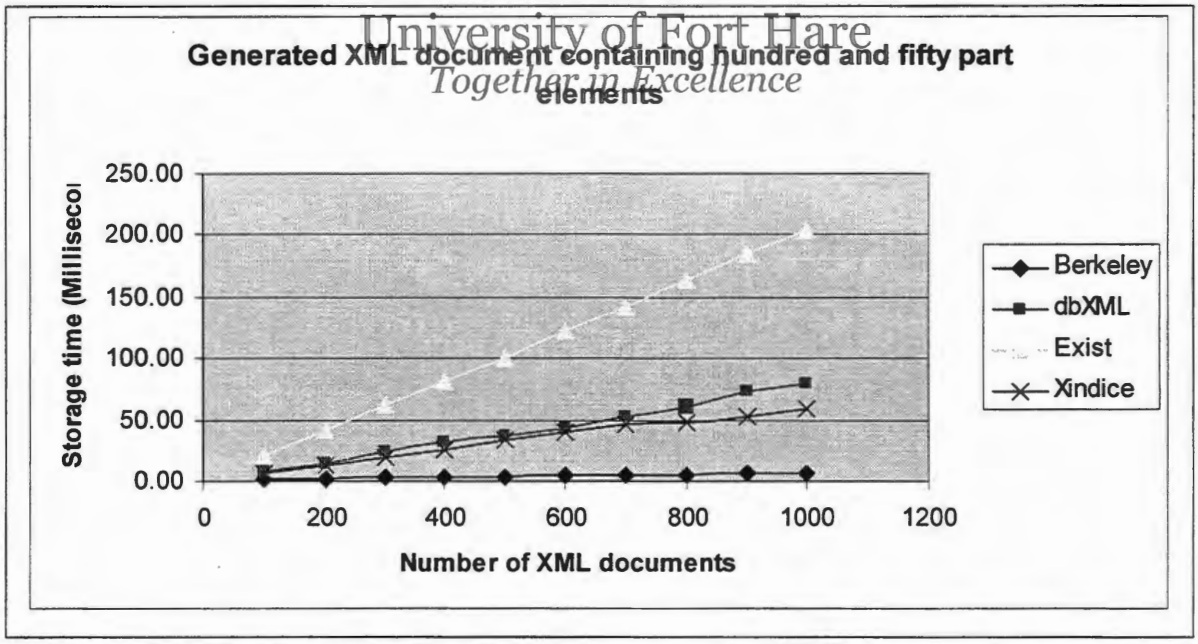


Figure 5.5: Timing comparison for storing generated XML document containing hundred and fifty part elements

Although the number of part items inside the generated XML document has increased from fifty to hundred and fifty part items, as shown in the comparison graph in Figure 5.5, the performance results seem to be similar to the one in Figure 5.4. Here the performance of Xindice is slightly higher compared to the previous Figure 5.4. This shows that the Xindice's proprietary format storage indexing techniques has a disadvantage when storing big size XML documents. The Berkeley DB XML still shows a better performance than the other three. It is followed by Xindice and dbXML. eXist still performs worse than the other three databases

5.3.2.3. The storage for thousand part elements

The experimental results for the third dataset used in this experiment will be discussed in this subsection.

Figure 5.6 presents the comparison graph showing the time taken by the four studied Open Source native XML databases for storing a generated XML document containing thousand part elements.

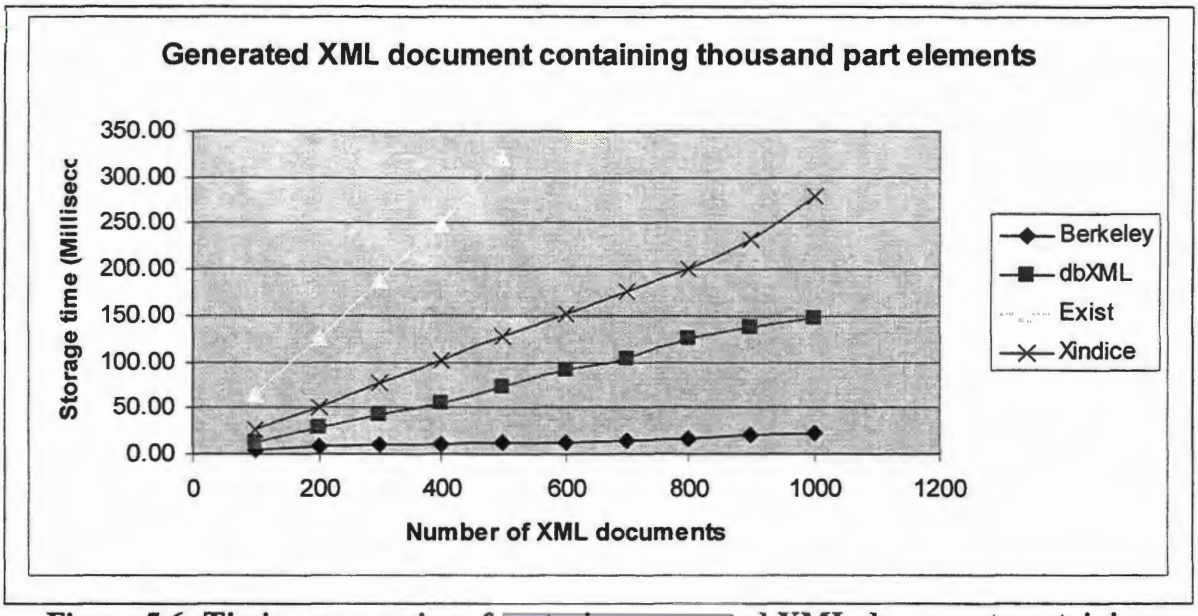


Figure 5.6: Timing comparison for storing generated XML document containing thousand part elements

As in the two previous graphs in Figure 5.4 and 5.5, the Berkeley DB XML is still showing a better performance than the [illegible]. eXist still performs worse than the other three databases. In this case it has even run out of memory when storing from 100 to 1000 XML documents. This is because the DOM mapping technique used by eXist can be very difficult to handle especially when storing big XML documents in the database. Looking at the graph in Figure 5.6, Xindice's performance seems to be problematic for handling large size documents. For example, in inserting XML documents from 100 to 1000 documents, its storage time has highly increased. This seems to be a common problem that might occur with databases that use the DOM mapping technique. As the XML document increases in size, building an XML document representation model for such document becomes time consuming and problematic.

As a conclusion, the results displayed in these three comparison graphs 5.4, 5.5, and 5.6 above have shown that the Berkeley DB XML is the fastest of the three Open Source native XML databases for storing XML documents and the slowest is eXist.

5.3.3. Queries

After storing the generated XML documents in the database, the next performance test was to query those data contained in the stored XML documents. The XPath was used as the preferred query language for querying XML documents. The reason for choosing XPath is that it is a W3C Recommendation and also it is a popular language for querying XML data. Most of the Native XML databases support XPath, including the ones that are being studied here.

Before examining the results of the query performance tests, a discussion of the different techniques used by XPath when querying or manipulating XML documents stored in the database will be done. The XPath query techniques were already introduced in section 2.4.4 in chapter two. Here, an attempt will be made to give an explanation of how these XPath query techniques work, referring to the generated XML documents that were stored in the Native XML database as an example. The reason for doing this is to get some insight into the manner in which XPath query methods will manipulate the stored generated XML documents. This, in addition, will assist in explaining our query test performance results.

As revealed in the previous section 5.3.1, the storage model used by the studied Open Source XML databases varies. This can be text-based storage, as shown in Figure 5.2, in the case of Berkeley DB XML, or model-based storage as displayed in Figure 5.3, for the other three, namely Xindice, eXist, and dbXML.

Regardless of the Open Source XML database storage models, XPath views an XML document as a tree of nodes. The nodes in the XPath tree are similar to those in the DOM tree. There are seven different types of nodes. These are: root, element, attribute, text, namespace, comment, and processing instruction. Each of these node instances is distinct from any other node instance.

In Figure 5.7, we show a graphical representation of the XPath tree for the generated XML documents.

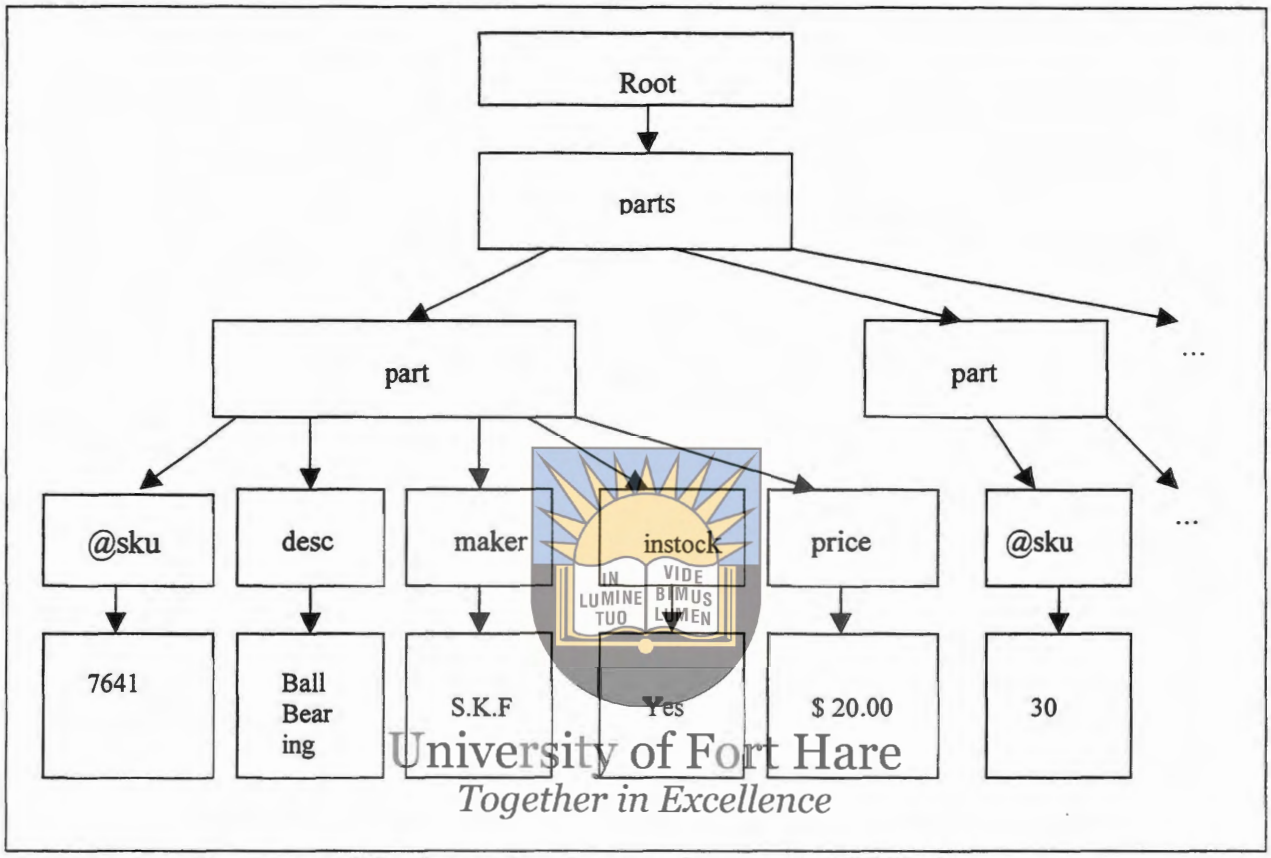


Figure 5.7: XPath tree for the stored generated XML documents

As shown in Figure 5.7 above, XPath's representation of the generated XML documents is similar to the model-based data representation in Figure 5.3. In other words, both model-based storage and XPath model the generated XML document as a tree of nodes. In the XPath tree, there is a single root node, which is different from the root element node of the document. The root node is the very first (and essentially a virtual) element of the XML tree, whereas the root element node of the document is the root element of the XML document which, in this example in Figure 5.7, is the *parts* element node. The root node in the XPath tree is the parent of the root element node of the document. The root node in the XPath tree also contains all other nodes in the tree. In addition, every node except the root node has a parent node, and parent nodes may have any number of child, or descendant, nodes. For example, looking at Figure 5.7, the *root* node contains one

child node (i.e. *parts*). Element node *parts* contains child element node *part*. The element node *part* is the parent of the attribute node *sku,* and element node *part* also contains child element nodes *desc, maker, instock,* and *price*. The element nodes *desc, maker, instock, price*, and also the attribute node *sku,* contain text nodes.

The order of child element nodes is significant, since all information accessible in the XPath tree is in the form of sequences. This order is established by the order in which the nodes appear in the original XML document (i.e. the generated XML document shown in Figure 5.1).

Each node in an XPath tree has a string value that XPath uses for comparing nodes. For this reason XPath defines a way for computing a string value for each node type in the tree. The string value of a node is computed by concatenating all text node descendants in the document order. For example, referring back to Figure 5.7, the string value for the *maker* element node is *S.K.F.* For the *part* element node that has four descendent element nodes (*desc, maker, instock, and price*), the string value consists of the concatenation of the string values for all its four descendent element nodes in document order. Thus the resulting string value is *Ball Bearings S.K.F. Yes, and $2.70*.

So far we have seen the how XPath models our stored generated XML as a tree structure. Now let us further examine how we can use this structure to situate or choose a particular fragment of our generated XML documents.

An XPath query is evaluated by a top-down traversal of the XPath expression tree. The input XML tree is traversed according to each sub-expression and the context state is updated along the way [67].

XPath query uses an expression referred to as *location path* for specifying how to move from one node to another in an XPath tree. In section 2.4.4, it was explained that a location path consists of one or more location steps separated by slashes (/). Every location step is made up of three parts: axis, node test, and optional predicate.

Searching through an XML document begins at a defined starting point (the context node) in the XPath tree. Searches through the XPath tree are made relative to this defined starting point (the context node).

Referring back to the example in Figure 5.7, suppose that one wanted to know which parts have S.K.F makers available; using the root node of the XPath tree as the context node (starting point), one could use the location path */parts/part[maker ='S.K.F']* to select the maker element node for each part that has a S.K.F maker. In this location path, the three parts of the location step mentioned in the previous paragraph are illustrated, namely axis, node test, and predicate. Here, the axis defines the direction to move in the document tree. In the example, the direction is from the root node to the maker node element. Location steps are separated by slashes (/). A node test is an evaluation to determine which nodes along the axis are selected for the next step which in our example is */parts/part.* The first step selects the parts node and the second step selects the <part> elements that are children of that node. Finally, a predicate is an expression which is enclosed in brackets *[maker='S.K.F']* to filter nodes selected by the node test.

After getting some insight into the XPath data model and techniques used for traversing an XPath tree, the performance of the studied Open Source Native XML databases in handling XPath queries can now be studied.

5.3.4. Test results for queries

A set of experimental tests for comparing the query performance of the studied Open Source Native XML databases were run. As in the previous experiments, the major concern for these experimental tests is the response time needed to execute a task. In the context of the experimental tests, the only interest is in timing the database query execution part. Timing is started from the beginning of the query execution, until the query execution is completed. This query execution is repeated five times and the average time from all these repetitions is calculated. The time required for connecting to the

database and time for getting a particular database collection that is being queried is not included in the timing results.

The text-based Open Source Native XML database was not able to query the generated XML document containing a thousand part elements because the machine ran out of memory when creating the DOM tree. Due to this problem, it was decided to use the 19.00 KB generated XML document containing a hundred and fifty part elements for the rest of the experiments.

The queries are executed on a number of collections, containing 100, 200,..., 1000 XML documents respectively. Each of these documents has a size of 19.00 KB.

The rest of this section presents and discusses the various query results that were obtained in the query performance tests.

In the interest of space, and also due to [redacted] arities of certain tests results obtained, not all the results will be considered for discussion here. Rather, some samples of these results have been selected for discussion. For the complete test results obtained in the various query tests conducted, please refer to Appendix C which contains all the test results.

5.3.4.1. The query /parts/part

In query (Q1b), *"/parts/part"*, interest was in displaying all the <part> elements that are children of parts. The objective of the test was to measure the query performance for displaying all the appearances of the <part> element within all the XML documents that are stored in the database. In Figure 5.8, the different times taken by the four studied open source native XML databases for accomplishing this operation are displayed.

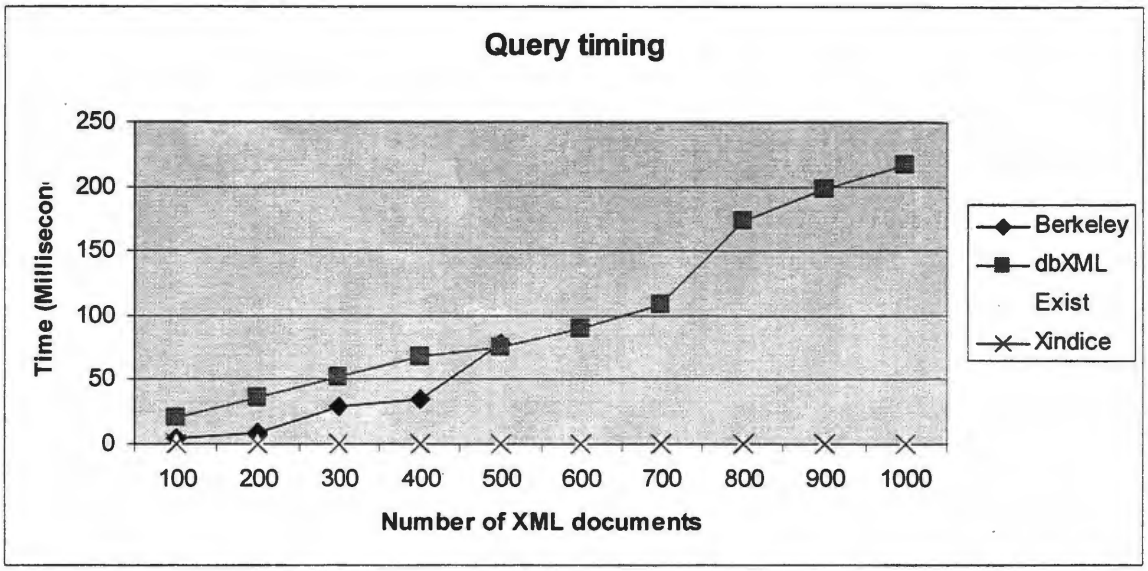


Figure 5.8: Comparison of time taken for executing query Q1b

As in Figure 5.4, the numbers displayed on the horizontal axis represent the number of generated XML documents that are stored inside that particular collection. The numbers on the vertical-axis represent various times obtained in milliseconds when querying collections of different sizes. Lastly, the [illegible] graphs represent the four Open Source native XML databases that are being compared in this query performance test, namely eXist (lower along the horizontal axis), Xindice (lower as well along the horizontal axis), Berkeley DB XML(middle) and dbXML(upper).

The results obtained from the query performance tests, as shown in Figure 5.8, reflect the advantages and disadvantages of the two main storage techniques used by the studied XML databases namely model-based and text-based. To recap, in the text-based model a document is stored "as-is" without being parsed, whereas the model-based approach parses an XML document before storing it according to a specific model (e.g. relational model, DOM persistent, and so on.). The XPath technology uses DOM parser technique for querying a document. As with model-based Native XML databases, every query operation also requires parsing the XML document that has been processed. Additional to this, the parsed document needs to be loaded into memory.

- The test results obtained reveal that the query performance of Xindice and eXist remains steady for all the documents' sizes. In other words, they show excellent

results when querying different document sizes contained in various collections. For example, looking at the comparison graph when querying from 100 to 1000 XML documents, the time taken to execute queries in Xindice and eXist does not change.This is a very good performance. Such behavior can be explained by the fact the two databases use a model-based storage technique. When storing their data, they spend a lot of time building indices on the data being stored. As a consequence of this, their performances are poor for the insertion operation, especially for eXist database. But when querying their stored data they take advantages of the indices built during insertion to quickly get information about data being queried. As result, this makes it easy for them to efficiently retrieve the data.

- Additional to this, the aforementioned Native XML databases use XPath as their query language for querying their stored data. As pointed out, XPath also uses model-based storage, namely DOM to represent data that is being stored. This fact also gives Xindice and eXist an advantage when using such techniques to query data. Because of this similarity, [redacted] model-based storage XML databases allow querying without parsing the entire document into memory.
- Referring back to the results in Figure 5.8, unlike Xindice and eXist, the query performance of dbXML is poor compared to the two model-based XML databases. The difference is caused by the mapping model used by dbXML. During its storage process, after parsing XML data that needs to be stored, dbXML uses a relational mapping model for storing the parsed data. The drawback of the relational model mapping approach is that the stored XML documents need to be reconstructed before being queried. As result of that, sometimes it can consume a lot of time when reconstructing the documents, especially when dealing with big documents or a huge collection of smaller documents. For example, Figure 5.8, shows that as the number of XML documents in the collection increases (e.g. from 100 to 1000) the performance of dbXML also starts dropping. On the other hand, since dbXML also uses XPath as its query language. XPath has to build a DOM tree of the document being

processed in memory before processing a query. One of the main problems with this technology (XPath) is its inability to process big size documents.

- The results in the comparison graph in Figure5.8 demonstrate that Berkeley DB XML performs worse than the others three databases. This very poor performance is attributed to the type of storage techniques (text-based) used by Berkeley DB XML. Unlike the model based approach, in text-based storage the entire document has to be parsed to perform any queries on it. This is always a problem when dealing with big XML documents or huge collections of XML documents. Furthermore, Berkeley DB XML supports XPath as its query language. As mentioned, XPath uses the DOM parser technique for accessing information stored in an XML document. This always parses the entire document being processed and builds a DOM tree representation of the XML document as illustrated in Figure 5.7. In addition to this, the DOM tree representation of the XML document is loaded into memory before executing the query. This requires a high memory usage when processing huge collection of small documents or big documents. The shortcomings of the DOM parser which is the main technique used by XPath were investigated in chapter 2 (i.e. it takes time to build the tree, high memory usage). For example, looking at Figure 5.8, when the number of XML documents stored in the collection increases, the time required for querying also increases. This is probably because the DOM parser required more time to build the tree. As a result, the Berkeley DB XML could not process all these data. It was just able to execute query from 100 to 500. After 500 it was impossible because the DOM parser required too much memory to load the whole dataset in the main memory before executing the query. As a result the machine runs out of memory when creating the DOM tree. Lastly, the other probable reason for the Berkeley DB XML's poor performance (i.e. machine runs out of memory) may be the fact that we used the default settings (i.e. default value of the duration of a single transaction for all Open Source NXDs tested) during the tests, as explained in section 4.5.2.2. Therefore if a single transaction takes a long time (i.e. when creating a DOM tree for big collection of XML documents), the machine may well run out of memory.

Based on our test results, we can conclude that Xindice and eXist have shown an excellent performance, followed by dbXML, and Berkeley DB XML has shown very poor performance. Furthermore, as mentioned, query (Q1b) involves displaying all the <part> elements that are children of parts. This was a general query without any precise specification. The results shown in Figure 5.8 confirm that the execution of this kind of query is affected by the size of the database in the case of some databases, namely Berkeley DB XML and dbXML as well.

5.3.4.2. The query /parts/part[75]/desc

In this section, the results obtained from the second type of query will be presented. Unlike the previous queries, which are general queries involving a full scan of each node in the DOM tree, here some specific queries are formulated which depend on specific factors. For example, in query (Q3b) */parts/part[75]/desc*, the part description *<desc>* element of the *<part>* element which occupies a specific position (i.e. 75 for this example which is the middle element position) within the hierarchy of node elements contained inside the generated XML document was queried.

As explained in the previous section, the aim of the query (Q3b) is to investigate the performance the studied databases for retrieving an XML document fragment depending on the hierarchical position occupies by the element being queried within the XML document.

The different times taken by the four native XML databases for executing the query (Q3b) are shown in Figure 5.9 below.

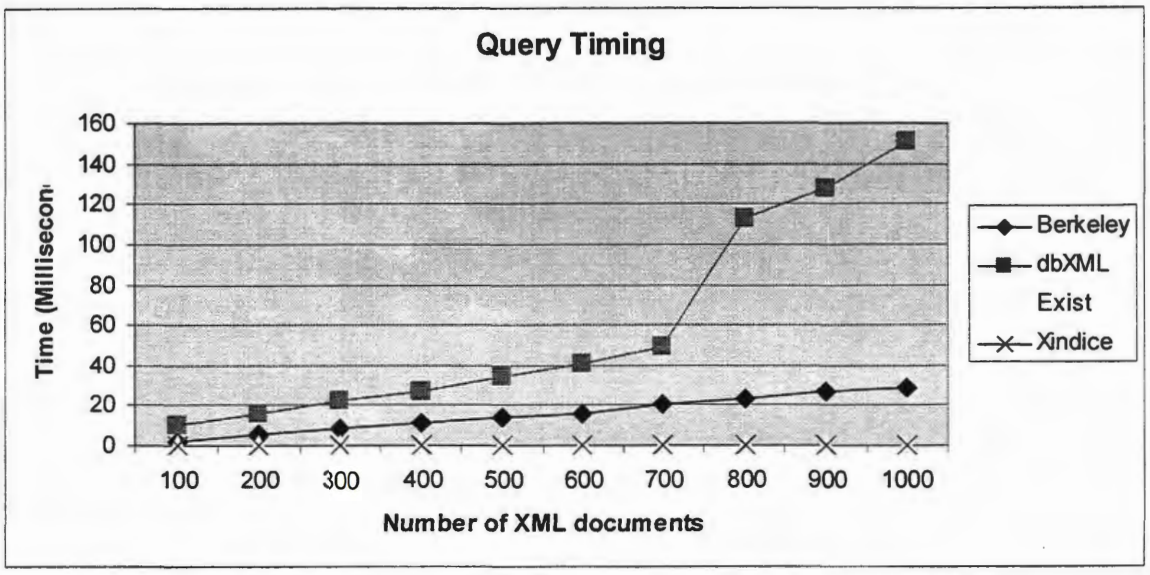


Figure 5.9: Comparison of time taken for executing query Q(3b)

Unlike Figure 5.8, Figure 5.9 shows that all the four studied open source native XML databases are able to perform this query operation for all document sizes. Also the time taken by Berkeley and dbXML for executing this query is faster than in the previous query. Note that for Xindice and eXist, the execution times appear analogous as in the previous query. The results obtained from the experiment have shown that the query execution time for Berkeley DB XML and dbXML was half reduced compared to the previous experiment. For example in dbXML for executing query Q1b, it took 20.0 milliseconds but for executing Q3b it took 10.0 milliseconds. Likewise for Berkeley DB XML for executing Q1b the timing was 5.00 milliseconds but for executing Q3 b it took 2.00 milliseconds. The author in [73] says that every element and tag has a unique XPath address. For example in the query (Q3b) */parts/part[75]/desc*, number *[75]* represents a unique XPath address for element <part>. Therefore when querying an XML document using a unique XPath address for an element the search should be faster than in the previous query Q1b. The previous query Q1b */parts/part* is a general query. The query (Q1b) requires a full scan of each node. It can take a long time and a large amount of memory to process the request especially when dealing with big XML documents. This is a clear disadvantage for text-based storage database (e.g. Berkeley DB XML) when processing this kind of query (full scan of each node).

In section 2.3.3 in chapter 2, it was mentioned that the performance of Native XML databases is good when retrieving and returning data according to the predefined hierarchy (i.e. order in which elements nodes are stored inside the XML document). This query is trying to query an element according to the position that it occupies inside the stored XML document.

On the other hand, as shown in Figure 5.7, the XPath DOM tree is built according to the hierarchy of node elements contained inside the XML document. Therefore, when executing this kind of query (i.e. query (3b)), it will be easy for the DOM parser to locate a particular element node in the DOM tree, knowing its XPath address within the DOM tree representation. Therefore this query (Q3b) will be executed faster by XPath as well.

Comparing the performance of each of these Native XML databases as shown in Figure 5.9, Xindice and eXist show excellent results which is the same as in the previous Figure 5.8. There is also a change in the performance of Berkeley DB XML which has performed better than in the previous test. This improvement should surely be attributed to the indexing scheme used by Berkeley. Also since the position of the element node is known, this can reduce the DOM parsing time and reduce search time. Also there might be no need for parsing and loading the whole XML documents into the memory. On the other hand, dbXML has still shown poor performance. This can be attributed to the time consumed for reconstructing the documents before querying.

From the test results, it can be concluded that, with query Q3b, Xindice and eXist have a steady performance, followed by Berkeley, and dbXML has shown poor performance. Furthermore, the execution of query Q3b is not affected by the size of the database.

5.3.4.3. The query /parts/part[@sku='7199']/desc

The other factor that was considered when formulating the query was the attribute of element. The reason for considering an attribute in this query performance test is that an

attribute node can also contain information about an element node. In other words, an attribute can be used to describe an element node.

This section, will present the results that were got from querying an attribute of an element node.

Each <sku> attribute of <part> element contained inside our generated XML document that is stored inside the database collection has a random generated value.
In query (Q3e) */parts/part[@sku='7199']/desc*, we queried the part description <*desc*> of the <*part*> element node that has an attribute <sku> value equal to 7199.
Retrieve an Xml document fragment depending on the value of an attribute node contained within the XML document.

Figure 5.10 displays the various times taken by the four studied open source native XML databases for executing the query (Q3e).

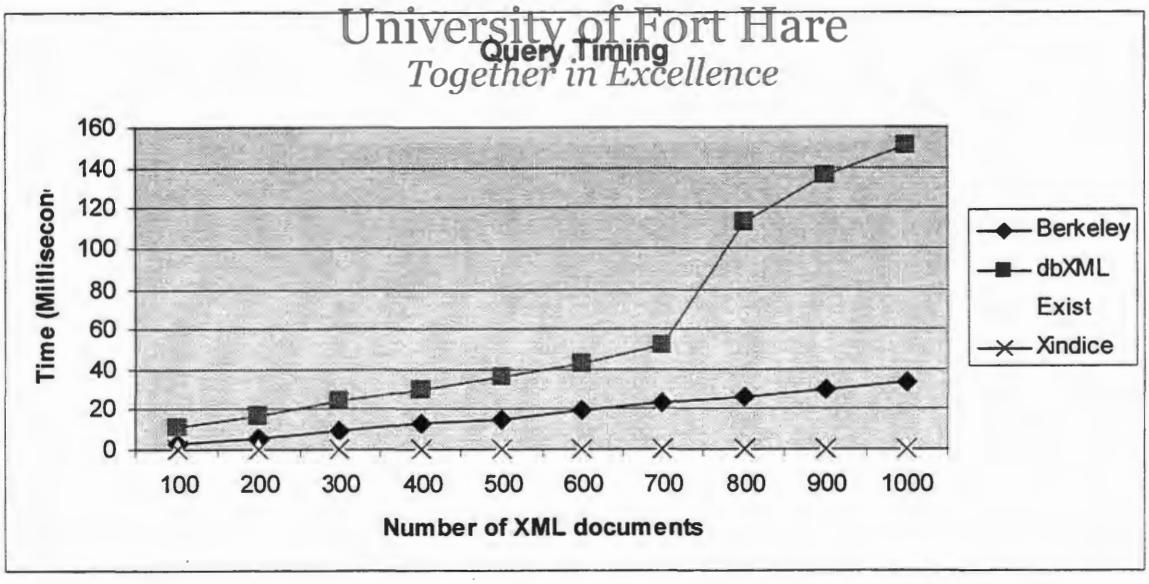


Figure 5.10: Comparison of time taken for executing query Q3e

The results obtained from this query performance test, as shown in Figure 5.10, are also similar to the previous ones in Figure 5.9. This shows that the queries Q3e and Q3b have almost the same performance.

As discussed above, the interpretation of the results shown in Figure 5.10 is similar to the ones previously given for 5.9. Therefore no further explanation is needed but to simply refer our readers to the explanation given for query Q 3b.

5.3.4.4. Other queries

Some other queries in the query performance tests were considered in additional to the queries discussed in subsections 5.3.4.1, 5.3.4.2, and 5.3.4.3. The following is the list of those queries:

Q(1a): /parts : display all (documents) t[illegible]nents

Q(1c): /parts/part/@sku: display all the <sku> attributes that are children of <part> element

Q(2a): /parts/part[@sku >'2414']: display all the <sku> attributes that are children of <part> and having value greater than 2414

Q(3a): /parts/part[1]/desc: display part description <desc> element that is the child of the <part> element which occupies position 1 within the hierarchy of node elements contained inside the generated XML document.

Q(3c): /parts/part[last( )]/desc: similar to Q(3a) and Q(3b) but here we want to display part description <desc> for the <part> element which occupies the last position within the hierarchy of node elements.

Q(3d): /parts/part[@sku='3119']/desc: display part description <desc> element that is the child of the <part> element which has product number <sku> attribute equal to 3119.

Q(3f): /parts/part[@sku='8834']/desc: like in Q(3d) and 3(e) but here we are interested in displaying part description <desc> for the <part> element whose product number <sku> attribute is equal to 8834.

Q(4a): /parts/part[@sku='7064' and maker ='S.K.F']: display <part > element that is the child of the <parts> element which has product number <sku> attribute equal to 7064 and maker's name S.K.F.

Q(4b): /parts/part[@sku='7199' or @sku ='3191']: display <part > element that is the child of the <parts> element which has product number <sku> attribute equal to 7199 or which has product number <sku> attribute equal to 3191.

Q(4c): /parts/part[@sku!='5604']: display all <part > elements that are children of the <parts> element which have product number <sku> attribute not equal to 5604.

Q(5a): /parts/part[178]/price: similar to Q(3a),Q(3b), and Q(3c) but here we are trying to display price value <price> element of the <part> element whose position is equal to 178.

Q(5b): /parts/part[@sku='107']/desc: like in Q(3d), Q3(e), and Q(3f) this query also displays part description <desc> of the <part> element which has <sku> attribute equal to 107.

In the next sections, some remarks are made concerning what was observed when the above listed queries were executed. Furthermore, all the different test results obtained from the above queries can be found in queries results table in Appendix C.

As for query (Q1b), when processing queries (Q1a), (Q1c), (Q2a), and (Q4c) that were also general queries, the candidate databases also demonstrated the same performance behavior. Due to this we didn't consider representing them graphically.

Apart from query (Q3e), which had a similar performance to query (Q3b), there are also other queries which show the same behavior and these include queries (Q3a), (Q3c), (Q3d), (Q3f), (Q4a), and (Q4b). When these listed queries were executed, the performance of the studied Open Source XML databases was the same as with queries (Q3e) and (Q3b). Therefore, we didn't see a need for representing them graphically here.

Lastly in the query performance tests, missing information or information that is not contained within the XML documents stored in the database was also tested. The reason we did this kind of query is to check how Native XML database handle this kind of operation.

When the query for missing data was executed, the response time that was got for querying a missing element in query (Q5a) was similar to the one got in query (Q3c), whereas the one got when a missing attribute was queried in query (Q5b), it was similar to the one got in query (Q3d). As with other queries that are not shown graphically here, the test results obtained for these two queries ((Q5a) and (Q5b)) can be found in queries results' table in the Appendix C.

The queries test results obtained for eXist and Xindice, as shown in Figures 5.8, 5.9, and 5.10, were very low timings for these two XML databases. For the sake of completeness, it was decided to re-run the experiment for these two named XML databases by increasing the number of query repetitions. The results that were got from the re-run prove that eXist's performance was superior to that of Xindice. For example for executing query (Q1c), the timings for both databases look dissimilar (over 100 to 1000 XML documents), the difference was eXist took 2.0 milliseconds whereas Xindice took 7.0 milliseconds. Although there is a difference between the two of them, they both still show good performance for all these te[illegible] The full test results from the re-run tests can be found in the query results' table in Appendix C.

In conclusion, the results got in the various queries as shown in the three comparison graphs 5.8, 5.9, and 5.10 above have shown that eXist and Xindice are the fastest of the four Open Source native XML databases for querying XML documents and the slowest is dbXML.

5.3.5. Modifications

There are many proprietary languages that have been developed for use in modifying the content of an XML document stored in an XML database. In the context of the modifications performance tests, XUpdate language was chosen. We are aware that XUpdate is not a W3C recommendation and also that not all the XML databases support XUpdate natively [69]. The reason for choosing to use XUpdate language for modifying

the content of XML documents is that it is an XML:DB initiative, which makes it an accepted update language, and most XML databases utilize the XUpdate language as their modification language.

The presentation of the results of the modification performance tests will start by explaining the different techniques used by XUpdate for performing alterations to the contents of an XML document. The generated XML documents, which were stored in section 5.3.2, are the basis for this discussion. The idea behind this is to gain some insights on the way XUpdate will manipulate the generated XML documents. This will also assist in illuminating the modifications test performance results.

In Figure 5.11, we show a section of sample XUpdate code that will update the value of <price> element contained inside our generated XML documents.


```
String xupdate = "<xu:modifications version=\"1.0\""
                 + " mlns:xu=\"http://www.xmldb.org/xupdate\">"
                 + "<xu:update select=\"/parts/part[@sku ='102']/price\">"
                 + "$350.00"
                 + "</xu:update>"
                + "</xu:modifications>";
```


Figure 5.11: an XUpdate update

The piece of code shown in Figure 5.11 above will update or change the content of the <price> element to $350.00 where sku (part number) is equal to 102. The basic idea behind XUpdate is to provide a way to make changes to an XML document using meaningful syntax [69]. Looking at the third line of code in Figure 5.11, it is evident that for selecting the part of the XML document that needs to be changed, XUpdate uses XPath syntax for specifying the path "*/parts/part[@sku ='102']/price*". This fact allows for the conclusion that XUpdate uses XPath technology for navigating between nodes contained inside an XML document that is to be modified. It can further be confirmed

that XUpdate's representation of the generated documents is similar to XPath's one. In other words, both XUpdate and XPath represent the stored generated XML documents as a tree of nodes as shown in Figure 5.7. Like XPath, an XUpdate statement would also be evaluated by a top-down traversal of the XPath expression tree.

After getting a brief background about several techniques used by XUpdate for modifying the contents of XML documents an investigation of the performance of the studied Open Source Native XML databases in handling XUpate technology can now be done.

In the modification performance tests, a modification will be considered of some of the existing data contained inside the collection that was queried in the previous subsection 5.3.4. The two main kinds of modification operation that will be performed in the modification tests data are:

- Modification of the value of element
- Modification of the value of an attribute

Berkeley DB XML does not support XUpdate as its update language. Because of this, Berkeley DB XML is not tested in the experiments for modification and delete.

5.3.6. Tests results for modification

Similar to the previous experiments, the major concern for the modification performance experimental tests is the response time needed to execute a task. Only the timing of the execution of the database modification operation part will be done. Timing is started from the beginning of the modification execution until its end. This execution process will be repeated five times and the average time taken will be observed. The time required for connecting to the database and time for getting a particular database collection that is being modified are not included in the timing results.

In the next subsections, the tests results obtained from the various experimental tests that were conducted for comparing modification abilities of the studied native XML databases will be presented and discussed.

5.3.6.1. The modification

```
+ "<xu:update select=\"/parts/part[75]/price\">" + "$700.00" + "</xu:update>"
```

The modification declaration (M1b) above is based on the query in 5.3.4.2 but here an attempt is being made to change the value of the <price> element from its actual value to $700.00. This value modification is to be done for the <price> element that is a child of the <part> element which currently occupies position 75 within the hierarchy of node elements. The aim of the modification (M1b) is to test the ability of each of investigated databases to handle such an operation. As mentioned earlier in this section, XUpdate uses the XPath technique for walking through the constructed DOM tree and for searching for the node that needs to be updated within the DOM tree. For example, the statement */parts/part[75]/price* is similar to the one for query */parts/part[75]/desc* in 5.3.4.2. The difference between the two statements is the target node element. Here we are interested in the <price> element whereas in the 5.3.4.2, we were interested in <desc> element.

In brief, we can say that for modifying the contents of any XML document, XUpdate constructs first the DOM representation of that document using XPath procedure and after that locates the actual node within the DOM tree, before it can carry out any changes and update the database.

In Figure 5.12, the various times taken by the investigated native XML databases for executing the modification (M1b) instruction are shown.

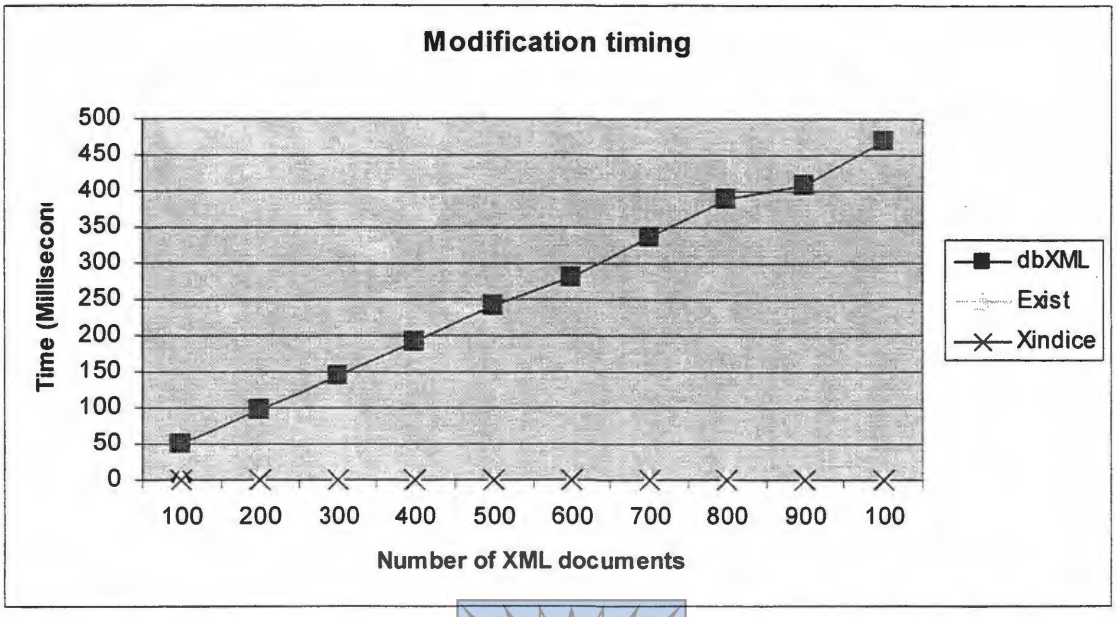


Figure 5.12: Comparison of time taken for executing modification (M1b)

The modification test results, as displayed in the Figure 5.12, clearly reveal that Xindice and eXist (both lower along the horizontal axis) outperform dbXML (upper). This difference in the performance of these databases demonstrates the advantages and disadvantages of the mapping model used by each of these databases when storing the XML data. Referring to the mapping model used by dbXML, it was already mentioned in subsection 5.3.4.1, that the relational mapping model used by dbXML for storing the parsed XML data has a major drawback, which is the reconstruction of the stored XML document every time that a given task (e.g. query, update, and so on) has to be performed. Therefore, when performing an update operation, additional to the time required by dbXML for reconstructing the XML document, there will be need also to take into account the time that XUpdate will spend in searching for the target node that needs to be modified within the constructed DOM tree. As a result the cost of these two operations (reconstruction, and update) can be time consuming. For example, looking at the comparison graph in Figure 5.12, it is obvious that, as the number of XML documents in the collection increases (from 100 to 1000), the performance of dbXML also starts worsening.

The results obtained from the conducted tests as illustrated in Figure 5.12 have shown that the update performance of Xindice and eXist are better than dbXML. This can be credited to the mapping model used by these two databases for storing their parsed XML documents and also probably to the indexing techniques that they use, as explained in subsection 5.3.4.1.

Based on the test results as shown in Figure 5.12, it can be concluded that Xindice and eXist have shown a very good performance, and dbXML has shown poor performance.
In chapter two, it was mentioned that Berkeley DB XML does not support XUpdate. For that reason, it was not considered in the modification test.

5.3.6.2. The modification

+"<xu:update select=\"/parts/part/@sku[. =7199]\"> + "509" + "</xu:update>"

Similar to query performance tests, the second category of modification involved modifying the value of an attribute of an element.
This modification aims to update the value of the attribute <sku> from 7199 to 509. This value modification is to be done for the <sku> attribute whose value (or product number) is equal to 7199 which is a child of the <part> element.

As mentioned in an earlier chapter, every <part> element contained inside our generated XML document was assigned a randomly generated sku (or part number) when the XML file was created. The value 7199 that we are trying to modify is an example of one of those generated sku (or product numbers).

In Figure 5.13, the test results that were got when modifying the aforementioned attribute of an element are presented.

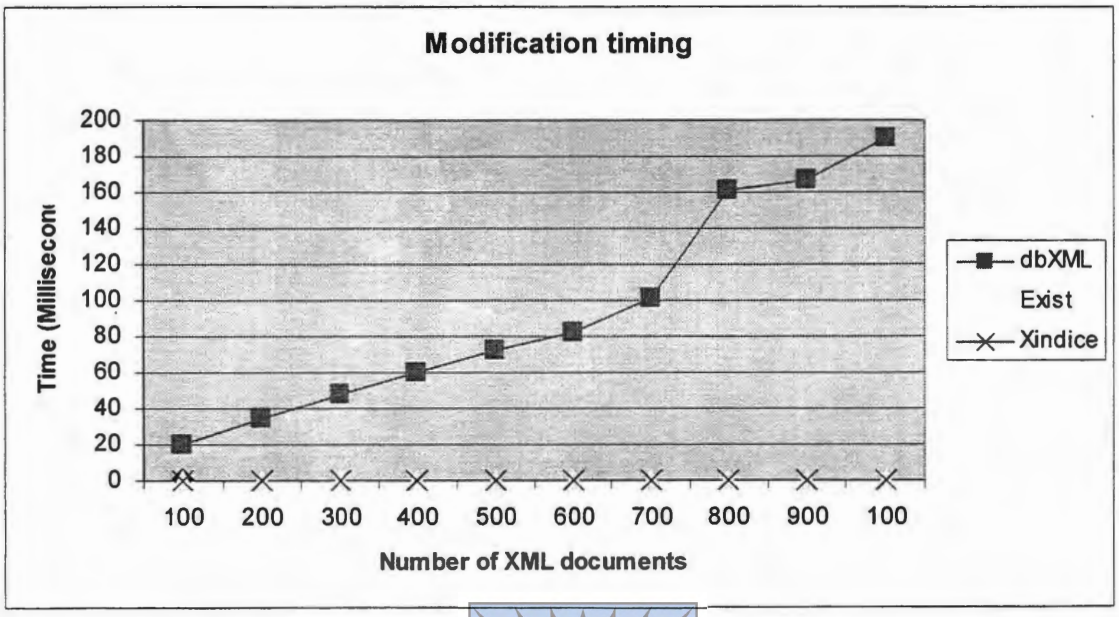


Figure 5.13: Comparison of time taken for executing modification (M1e)

The general performance results obtained from this update test, as shown in Figure 5.13, are similar to the previous one in Figure 5.12. As in the previous results, here also, Xindice and eXist (both lower along the horizontal axis) do better than dbXML (upper). As discussed above, the interpretation of the results displayed in Figure 5.13 is analogous to the ones previously given for Figure 5.12. Therefore no further details will be given, however readers are referred to the justification given in the previous section.

There are a few remarks that can be made concerning the timing results that were obtained in this update test. The execution times for Xindice and eXist remain very similar as in the previous performance modification test (M1b). On the other hand, the test results have shown that the update execution time for dbXML has greatly reduced compared to the previous experiment (5.3.6.1). For example for executing the update in the case of 700 XML documents it took 335 milliseconds for (M1b) but it took 101 milliseconds for executing (M1e). As previously explained, (M1b) involves modifying the value of element whereas (M1e) involves modifying the value of attribute. This allows for the observation that updating a value of an attribute is faster than updating a value of element. The reason for this performance difference surely is due to the indexing schemes used by dbXML. In section 3.3.5.1, it was pointed out that dbXML provides

value indexes. This kind of index is created based on attributes which may include the name and location of attributes, and their values. This can explain why (M1e) is executed faster than (M1b).

5.3.6.3. Other modifications

Apart from the modifications (M1b) and (M1e) that have been discussed in subsections 5.3.6.1 and 5.3.6.2, some other modification tests have also been investigated which show the same behavior as (M1b) and (M1e). The following is the list of these modification tests:

(M1a): + "<xu:update select=\"/parts/part[1]/price\">" + "$500.00" + "</xu:update>"
In (M1a), we are trying to change the value of the <price> element from its current value to $500.00. This change is to be done for the <price> element that is a child of the <part> element which occupies position 1 within the hierarchy of node elements contained inside the generated XML document.

(M1c): + "<xu:update select=\"/parts/part[last()]/price\">" + "$540.00" + "</xu:update>"
Like in (M1a), (M1c) also involves changing the value of the <price> element to $540.00. This change is to be done for the <price> element that is a child of the <part> element which occupies the last position within the hierarchy of node elements.

When the aforementioned modification ((M1a) and M(1c)) were executed, similar results to those shown in Figure 5.12 for modification (M1b) were achieved. For this reason, they were not graphically represented here.

(M1d): +"<xu:update select=\"/parts/part/@sku[.='3119']\"> + "65" + "</xu:update>"
In modification (M1d), we are trying to modify the value of the attribute <sku> from 3119 to 65. This value change is to be done for the <sku> attribute whose value (or product number) is equal to 3119 which is a child of the <part> element.

(M1f): +"<xu:update select=\"/parts/part/@sku[.='8834']\"> + "2636" + "</xu:update>"
Similar to (M1d), in (M1f) as well, we are trying to modify the value of the attribute <sku> from 8834 to 2636 for the <sku> attribute which is a child of the <part> element.

When the above modifications namely ((M1d) and M(1f)) were executed, the timing responses that resulted were similar to the ones shown in Figure 5.13 for modification (M1e). Because of their similarities, they are not discussed here.

Nevertheless, the update test results obtained for these update tests conducted ((M1a), (M1c), (M1d) and M(1f)), can be found in the modification results' table of Appendix C.

In conclusion, the results that were got in different modification tests as shown in Figure 5.12 and Figure 5.13 reveal that eXist and Xindice are the fastest, and the slowest is dbXML.

For the sake of completeness, also taking into consideration the very low timings that were got for eXist and Xindice, it was decided to re-run the experiment for the two databases (eXist and Xindice) by increasing the number of update repetitions. The results obtained from the re-run tests confirm that eXist performs better than Xindice. For example for executing modification (M1b), the timings for both databases seem to be distinct (from 100 to 1000 XML documents). The difference was that Xindice took 8.0 milliseconds whereas eXist took 2.0 milliseconds. All the re-run test results can also be found in the update results' Table in Appendix C.

5.3.7. Deletions

XUpdate, from the XML:DB Initiative, is an XML-based update language that uses XPath to identify a set of nodes, then specifies whether to modify or delete these nodes,

or insert new nodes before or after them [70]. This implies that, apart from XUpdate's capability to perform the modification operation, it can also be used to carry out some other database basic operations. Therefore XUpdate was the preferred language of choice for the deletion performance tests.

XUpdate and its related technology have already been discussed in the previous section. The discussion here will talk about the XUpdate deletion code.

Figure 5.14 displays a section of sample XUpdate deletion code that will remove the selected element, which this case is the <price> element.

```
String xupdate = "<xu:modifications version=\"1.0\""
                 + " mlns:xu=\"http://www.xmldb.org/xupdate\">"
                 + "<xu:remove select=\"/parts/part[@sku ='102']/price\">"
                 + "</xu:remove>"
                 + "</xu:modifications>";
```

Figure 5.14: an XUpdate delete

This piece of code deletes the <price> element that is a child of <part> element where sku (part number) is equal to 102.

If the XUpdate code in Figure 5.14 is compared to the code in Figure 5.11, it can be seen that both have similar syntax. The only difference is in the select part of the code, where for deletion XUpdate uses *<xu:remove select>* which is the case in Figure 5.14, and for updating it uses *<xu:update select>* as shown in Figure 5.11. These two are children (or sub elements) of the *<xu:modifications>* which is the root element of the XUpdate document. *<xu:modifications>* can contain one or more elements. In the XUpdate statement the sub element (child) of *<xu:modifications>* is used to specify the type of modification that needs to be done inside an XML document.

Readers who would like to know more about other types of sub elements (children) that *<xu:modifications>* element can have, or different kinds of modification that can be done with XUpdate, are referred to section 2.4.5 of Chapter 2.

The kind of delete operations that were considered for these performance tests involve attributes and elements contained inside the generated XML documents that were stored in section 5.3.1.

The tests results obtained from each of the studied Open Source native XML databases will be analyzed in the next sections.

5.3.8. Tests results for deletion

Similar to all the previous experiments, [illegible] performance tests, the major concern is the response time needed to execute a task. Only the execution part of the delete operation is timed. The timing begins from the start of the execution of the delete instruction until the execution of the delete operation is completed. This delete execution process is repeated five times from which the average time is decided. The time required for connecting to the database and time for getting a particular database collection which contains the node to be deleted are not included in the timing results.

This part of the performance evaluation, involved two types of deletion performance tests. These two include deleting the element, and deleting an attribute of an element from the database.

5.3.8.1. The deletion "<xu:remove select=\"/parts/part[75]/price\"/>"

The deletion (D1b) code shown in above, involves removing the <price> element which is a child of <part> which currently occupies position 75 within the hierarchy of node elements. If this deletion is compared with the modification in 5.3.6.1, they seem to be the same. The only difference is in the XUpdate instruction. In 5.3.8.1, a *remove* instruction is used whereas in 5.3.6.1 a *modification* instruction is used. Furthermore, both use the XUpdate. This means that there will be some kind of similarities in the manner in which these two operations will be carried out. For example, for deleting a node from the database collection XUpdate uses the same technique (XPath) (e.g. /parts/part[75]/price) for walking through the built DOM tree and for searching for the node that needs to de deleted from the database collection.

In brief, it can be said that modification and deletion operation functionalities have the same logic. There is not much difference between the two.

The different times taken by the studied native XML database for carrying out the deletion (D1b) instruction are shown in Figure 5.15.

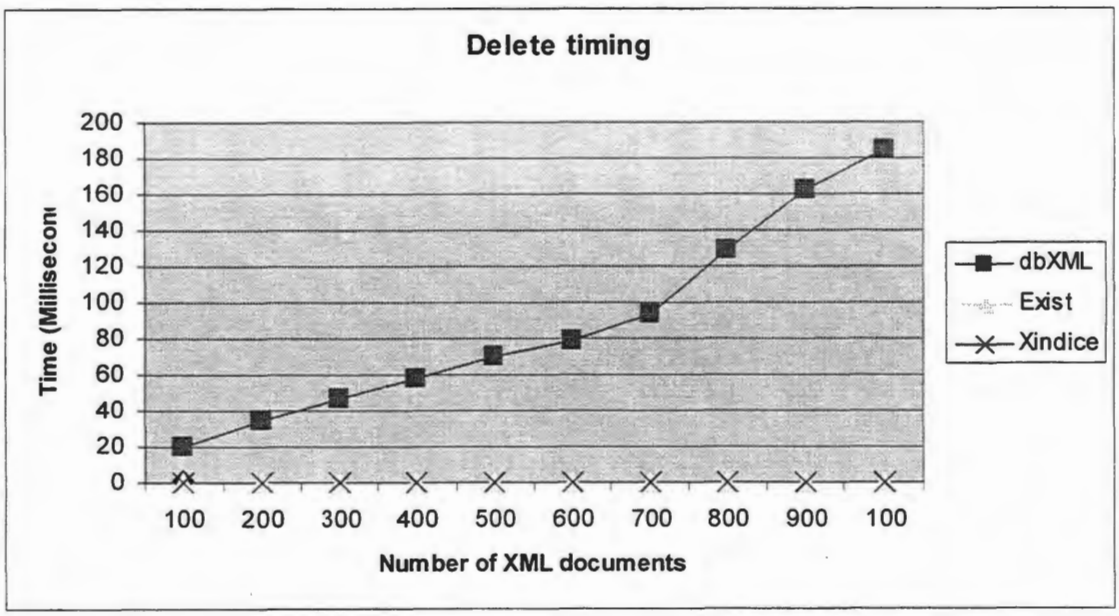


Figure 5.15: comparison of time taken for executing delete (D1b)

Earlier in this subsection it was suggested that there are similarities in the manner in which deletion and modification operations are carried out. The test results in Figure 5.15 show also some similarities in the database behaviors compared to the previous results obtained in modification tests (M1b). For example, looking at Figure 5.15, both products Xindice and eXist have steady performances. They both outperform dbXML as in the previous experiments' results. The reason behind these databases performances are the same as the ones previously given for the Figure 5.12. Therefore no further explanation is needed but to simply refer readers to the explanation given in modification (M1b) in subsection 5.3.6.1.

With regard to the timing obtained for dbXML, the results obtained for deletion (D1b) are better than the ones for modification (M1b). This is reasonable, because a delete operation involves searching and removing a node whereas modification involves searching for the node and writing into the database collection. This operation can be time consuming especially for a big size d[illegible] For example in the case of 1000 XML documents, it took 469 milliseconds for executing a modification (M1b) and it took only 185 milliseconds for executing a deletion (D1b).

The timings for Xindice and eXist remain the same for executing a modification (M1b) and a deletion (D1b).

The second part of the deletion performance tests involves deleting an attribute of an element from the database.

5.3.8.2. The deletion "<xu:remove select=\"/parts/part/@sku[.='509']\"/>"

The deletion (D2b) operation above, involves removing the attribute of an element from the database. Here, an attempt is made to delete the attribute <sku> with value equal to 509. The <sku> attribute that needs to be removed from the database is the child of the <part> element. To recap, in section 5.3.6.2, it was pointed out that every <part> element

contained inside the generated XML document was assigned a randomly generated sku (or part number) when the XML file was created. 509 is one example of this randomly generated part number.

Figure 5.16 is a graphical representation of different timing results that we got when the studied XML databases executed the delete (D2b) operation.

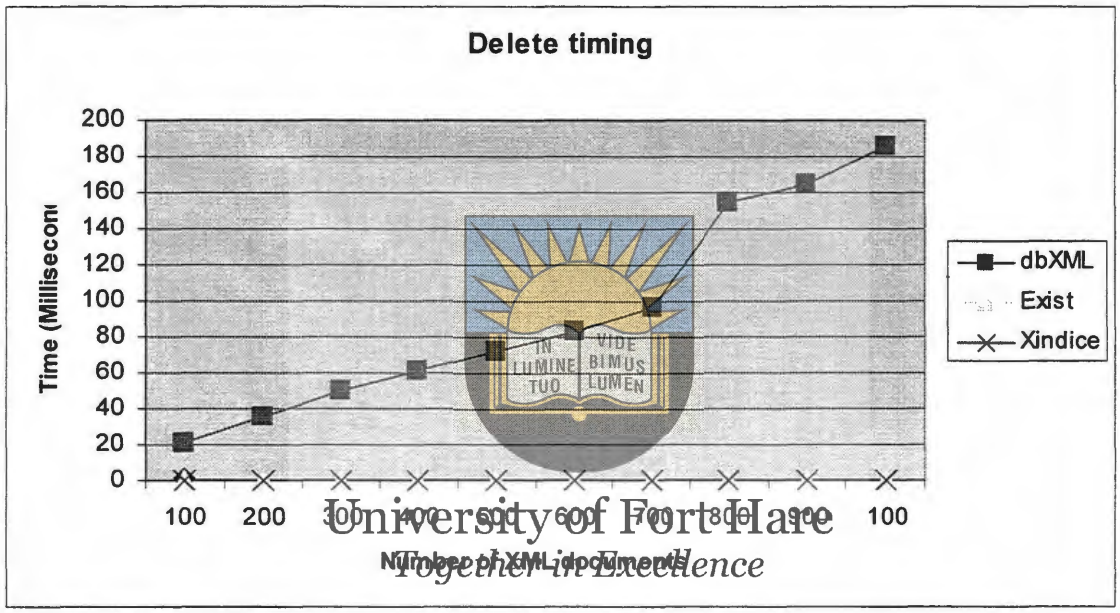


Figure 5.16: comparison of time taken for executing delete (D2b)

The performance's results obtained from this delete test, as shown in Figure 5.16, are very similar to the one in Figure 5.13.

Due to the performance result similarity of (D2b) and (M1e), no further explanation about the behaviors of the databases will be given except to refer readers to the explanation given in modification (M1e).

5.3.8.3. Other deletions

Additional to the deletion operations which were discussed (D1b) and (D2b), several other deletion performance tests were also conducted to study the deletion performance of the studied XML databases. These include D(1a), D(1c), D(2a), and D(2c).

D(1a): + "<xu:remove select=\"/parts/part[1]/price\"/>"
Like D(1b), deletion D(1a) also involves removing the <price> element which is a child of <part>, which occupies position 1 within the hierarchy of node elements contained inside the generated XML document.

D(1c): + "<xu:remove select=\"/parts/part[last()]/price\"/>"
This is similar to D(1a) and D(1b), but the <price> element that needs to be deleted is a child of <part> which currently occupies the last position within the hierarchy of node elements.

The test results that were got from the two above listed deletion operations, D(1a) and D(1c) have not shown much change compared to the one for D(1b). Due to their similar behavior, there is no need to discuss them here. The results obtained in the two deletion operations can be found in Appendix C.

D(2a): + "<xu:remove select=\"/parts/part/@sku[.='65']\"/>"
Similar to D(2b), in D(2a), we would like to remove the attribute <sku> with value equal to 65. This <sku> attribute is the child of the <part> element.

D(2c): "<xu:remove select=\"/parts/part/@sku[.='2636']\"/>"
Similar to D(2a) and D(2b), the operation involves removing the attribute <sku> whose value is equal to 2636 which is a child of the <part> element.

When the deletion operations D(2a) and D(2c) were executed the performance of D(2a) was similar to D(2b). On the other hand D(2c) shows very little change compared to the

other two. The results obtained in the two deletion operations can be found in Appendix C.

In conclusion, the results that were got in different delete tests as shown in Figure 5.15 and Figure 5.16 show that, for executing the delete operation, eXist and Xindice are the fastest and slowest is dbXML.

As with the previous tests, in deletion operations as well, the experiment for eXist and Xindice were re-run by increasing the number of deletion repetitions. The results obtained from the re-run tests prove that eXist's performance is better than Xindice. For example for executing deletion (D2b), the timings for both databases appear different (from 100 to 1000 XML documents). The difference was Xindice took 6.0 milliseconds whereas eXist took 2.0 milliseconds to complete the operation. The re-run test results for deletion can be found in the delete results' table in Appendix C.

5.4. Comparison of Findings

This section compares our experimental test results with some previous findings from other related works.

In general, most of the experimental test results discussed in the previous section 5.3 supported some of the related previous research findings which were highlighted in section 5.2.

The database storage test results that we obtained show that Xindice performed better than eXist. This supports previous research results by [17, 18] indicating the superiority of Xindice compared to eXist when carrying out the database insertion operation.

Regarding the database query operation, the results from the re-run query tests confirm that eXist outperforms Xindice. This finding is similar to the one made by authors in [17, 18] who claimed that eXist's query execution was quicker than Xindice's one. The re-run test results furthermore show that the Xindice's modification test results were satisfying. This finding once more is similar to [18].

5.5. Summary

This chapter started by introducing some of the previous related works' findings. Additional to this, the experimental results that we obtained using each of the studied Open Source native XML databases were presented and discussed. Before interpreting and comparing the obtained test results shown in the graphs, various technologies used by native XML databases for carrying out the basic database operations that were investigated were discussed, with the aim of giving some insight into these Native XML database related technologies. The generated XML documents that were used in these experimental tests were the basis for this discussion. The reason for doing this was to understand how the various Native XML databases related technologies managed the experimental datasets. Getting a better understanding of these various techniques was a prerequisite for us in understanding, explaining, and comparing the final experimental tests results obtained from each of the studied Open Source XML databases.

The experimental results obtained from the performance tests conducted indicated that Berkeley DB XML was the fastest when executing database storage operation. For the others three database operations (e.g. query, modification, and delete), the results show that the performances of eXist and Xindice were better. Both performed equally well when executing the query, modification, and the delete operations.

Low timings were recorded for eXist and Xindice when executing the three mentioned database operations. Due to that, it was decided to re-run the experiment for the two

databases. The results from the re-run experiments reveal that eXist was the fastest when executing the query, modification, and delete operations.

The results obtained of our database performance tests do agree with findings from some previous related works.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1. Introduction

This chapter brings to a conclusion this thesis on the comparison of the performance of Open Source native XML databases. It summarizes the most important findings of this work, and provides a summary of different steps taken during this project as well as the performance results.

6.2. Thesis Contribution

A number of issues surrounding Open Source native XML databases systems have been examined. A comprehensive analysis and investigation of the various XML technologies behind Open Source native XML databases was carried out. Further, diverse techniques used by these databases for carrying out the primitive database operations, how these work and what factors can influence their performance were examined. Most importantly, the amount of time taken by each of the studied Open Source native XML databases for executing database basic operations serves as the core work of this thesis. A more concise summary of this aspect of the work appears in [71]. Furthermore, recently, this work has been selected for presentation at [72].

6.3. Work covered

There has been an increase in the number of XML documents generated from business transactions. As a result, there is a need to find an effective approach for storing and processing XML documents.

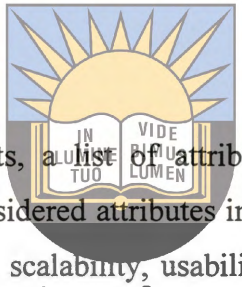
This project was inspired by a desire to make an informed choice about the efficiency of database technology for storing and processing XML documents. The project focused on

Native XML database products, namely Open Source. Some criteria were applied for selecting the Open Source native XML databases that were used in this project. The selection criteria were the following:

- The databases products should be written in the Java language.
- They must have a comprehensive documentation.
- They must have support in the form of news groups.

Based on the above criteria, the four selected products were the following:

- Berkeley DB XML
- dbXML
- eXist
- Xindice.



For evaluating the chosen products, a list of attributes that were considered to be important was compiled. These considered attributes include functionality, market share, support, maintenance, performance, scalability, usability, security, and flexibility. More emphasis was placed on the products' performance issues.

Often the selection of a particular Open Source native XML database is based on several factors. An imperative factor is performance. Performance tests were conducted on each of the chosen Open Source products, with the aim of measuring the response time taken to execute an instruction for carrying out a particular basic database operation. The basic database operations considered in these tests include storage, query, modification, and deletion. The purpose of the performance tests was twofold. Firstly, to determine which database product generated the fastest response (e.g. time taken when executing a particular task) and, secondly, to determine which open Source product performed best when performing a specific task. The performance test results in this experiment were compared with previous related works.

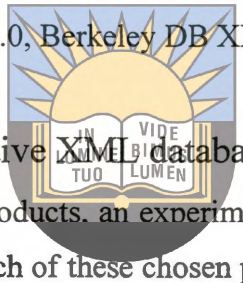
However the metric performance was not the only issue considered. An attempt was made to understand how the different XML-related technologies used by the studied

database systems work and to gain some insight into these technologies. The understanding of these technologies helped in identifying different factors that can influence their performance.

Below are the different stages that were carried out during the project.

Selection of the Open Source XML database products

The first step was to define the criteria for selecting the studied Open Source native XML database products. The Java programming language, comprehensive documentation, popularity, and support in the form of newsgroups were the criteria used in our product selection process. Based on these criteria the following products were selected: Xindice version 1.1b4, eXist 0.9.2, dbXML 2.0, Berkeley DB XML 1.1 Win 32.



Setting up the Open Source Native XML databases

After downloading these selected products, an experimental test environment was set up as explained in section 4.3 to test each of these chosen products and, in the same chapter, to describe the proposed tests, including the test hypotheses that were formulated for the purpose of this exercise.

Experimental data sets used

A Java program was written that generated the data sets that were used for testing the performance of the studied open source XML database products in handling basic database operations.

Experimental results

Graphs were used as shown in chapter five for comparing the results that were obtained from the various performance test operations conducted on the studied products. Each of these results displayed in the comparison graphs was discussed and a short conclusion was drawn based on the results obtained with regard to the products' performances in the specific operation.

6.3.1. Overall impression about Open Source Products

At the start of this project, we were still new to the field of Open Source Native XML databases. As the project progressed, an effort was made to learn the studied Open Source native XML database technology.

A lot of understanding has been gained about the technology, and how things are done in the Open Source Community. Hitherto unknown features, functionalities, and capabilities of some of these products have been discovered.
At the same time also, this journey into the Open Source world proved challenging. There were different problems experienced with some of these products, for example installing and running some of them.

With regard to the diverse problems encountered throughout the duration of this project, very little help or even no help at all was forthcoming from the forum for solving problems. A frequent answer in the mailing lists was "*check archives*". It did not help at all because even in the archives no answers could be found for the problems. In such a situation, the only option was to research more about the problem and finds a way to solve it. Solving such problems led to a gain in knowledge of the studied Open Source XML databases technologies.

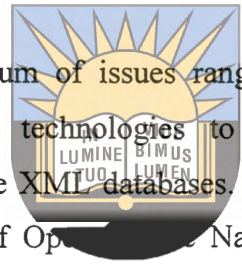
On the other hand, the good thing about the OSS community is that they are a helpful faceless community. With some of the problems that were encountered, when a request was posted for help on the forum, positive replies were received and the problems were thus solved.

The other issue with these Open Source products was the speed of change in terms of version releases. There are frequent version releases for some of these Open Source

products. This was confusing sometimes because in such a situation one does not have time to get very deep knowledge about a particular version of a product.

In summary, the experience as being part of Open Source Community and also of working with Open Source Native XML databases throughout the duration of this work confirms the claims by Dan Farber [73] that there are still significant barriers to overcome before Open Source projects are broadly accepted across enterprises.

6.4. Limitations of this project



This work covered a broad spectrum of issues ranging from an overview of XML databases and their XML-related technologies to different factors affecting the performance of Open Source Native XML databases. The purpose of this work was to compare the metric performance of Op[illegible] Native XML databases in handling basic database operations. The evaluator's focus was on the amount of time taken by each of the studied Open Source native XML databases for executing a particular database basic operation. The approaches taken during the evaluation process were subject to the evaluator's needs. Based on this the following were some of the limitations of this work:

- Not all the features of studied Open Source native XML databases were investigated in this project.
- The studied databases were only a small selection of the Open Source native XML databases available for consideration in this project.
- Being new to the Open Source native XML database field, it took some time to grasp the technology and, due to time constraints, there was not enough time to keep up with the new version releases of some of these studied databases products.
- The datasets used in the performance test experimental were simple XML documents generated for the purpose of the project. It would be interesting to have had real world data.

- Tests were only conducted in the Windows operating System.

6.5. Future work

The previous section 6.4 highlighted some of the limitations of this project. Many areas of study can be initiated from this project and these are:

- An extension to the research could include an investigation of new version releases of some of the studied databases products. Hopefully, these would show improvement over their predecessors that were used in this project.
- Conducting performance evaluation tests with other Open Source native XML database products would be useful.
- Another interesting extension to this work could be to conduct the performance test evaluation using a complex XML document.
- In section 4.3.2, it was mentioned that the hardware used to carry out all the experiments consists of a single machine. An extension of the work conducted in this thesis would be to conduct a performance tests on different database architectures such as three-tier client-server architectures.

6.6. Conclusion

With regard to the various tests that were conducted on the four studied XML databases, the following observations were made.

- Berkeley DB XML outperforms the other three Open Source native XML databases (i.e. dbXML, eXist, and Xindice) when storing XML documents of varying size in the database.
- Based on the experimental results, it is shown that Berkeley DB XML had a very poor performance when executing general kinds of queries as the size of the XML documents increases. Thereafter, when executing a specific query (e.g. query based on the position occupied by an element in the hierarchy of nodes, or query based on the value of an attribute of an element), Berkeley DB XML showed a much better performance.
- Furthermore, when executing a specific query, dbXML's performance was faster than in the previous general query. Xindice and eXist were faster than dbXML and Berkeley DB XML when executing both general and specific queries.
- The experimental results, further reveal that Xindice and eXist perform equally well when executing modification and delete operations. Both outperform dbXML.
- The results from the test re-run demonstrate that eXist performs better than Xindice for executing query, modification, and delete operations.

Other findings

- All the four Open source native XML database systems, which were used in this test performance evaluation, can basically do what is needed.
- The storage techniques used by these database products have an impact on their performance when carrying out the basic database operations. Furthermore, each of these techniques was shown to have some advantages and disadvantages. For

example, the model-based database (i.e. dbXML, eXist, and Xindice) has to parse and build a model for the XML documents being stored whereas the text-based model just stores the XML documents as it is.

- The experimental results have shown that indexing techniques used by these databases can, in some cases, improve the database performance. For example, a model- based storage database spends time building indexes when storing data. In return they are faster when executing queries, because they use the built indexes.
- Some of our test results were similar to some of the results of other researchers.

In review, the experimental results from the performance tests have revealed that none of the considered Open Source native XML databases presents the perfect solution for handling XML documents. Each of these databases has weaknesses and strengths, but the choice of the Open Source native XML database depends primarily on the structure and the size of the XML documents being used, as well as on which operations are considered most important by the user.

References

[1] W3C. **Extensible Markup Language (XML)**. Available [On-line]: http://www.w3.org/XML [Last access: November 2005]

[2] Michael H. Kay. **XML five years On: A Review of the Achievements so far and Challenges ahead**. Available at http://www.saxonica.com/papers/DocEng2003.pdf [Last access: November 2005]

[3] **The X007 Benchmark**. Available [On-line]: http://www.comp.nus.edu.sg/~ebh/XOO7.html [Last access: November 2005]

[4] Stephane Bressan, Mong Li Lee, Ying Guang Li, Zoe Lacroix and Ullas Nambiar. **The X007 XML Management System Benchmark**. November, 2001. Available [On-line]: http://www.comp.nus.edu.sg/~ebh/XOO7/download/XOO7_TechReport.pdf [Last access: November 2005]

[5] Ying Guang Li, Stephane Bressan, Gillian Dobbie, Zoe Lacroix, Mong Li Lee, Ullas Nambiar, Bimlesh Wadhwa. **X007: Applying 007 Benchmark to XML Query Processing Tool**. In Proceedings of the 10th ACM International Conference on Information and [illegible]age Management (CIKM2001), Pages 167-174. ACM Press, 2001.

[6] Ullas Namibiar, Zoe Lacroix, Stephane Bressan, Mong Li Lee and Ying Guang Li. **XML Benchmarks Put to the Test**. In Proceedings of the Third international Conference on information integration and Web-based Applications & Services (IIWAS), Linz, Austria, September 2001. Available [On-line]: http://www.comp.nus.edu.sg/~ebh/XOO7/download/pIIWAS01.pdf [Last access: November 2005]

[7] **Xmark**. Available [On-line]: http://monetdb.cwi.nl/xml [Last access: November 2005]

[8] **XMach-1**. Available [On-line]: http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html [Last access: November 2005]

[9] Timo Bohme and Erhard Rahm. **XMach-1: A benchmark for XML data management**. Available [On-line]: http://dbs.uni-leipzig.de/en/projekte/XML/paper/XMach-1.html [Last access: November 2005]

[10] **XBench**. Available [On-line]: http://se.uwaterloo.ca/~ddbms/projects/xbench [Last access: November 2005]

[11] M. Tamer Ozsu and Benjamin B. Yao. **Evaluation of DBMSs Using XBench Benchmark**. Available [On-line]: http://www.cs.uwaterloo.ca/research/tr/2003/24/TR-CS-2003-24-1.pdf [Last access: November 2005]

[12] B.B. Yao, M.T. Ozsu, and J. Keenleyside. **XBench-a family of Benchmarks for XML DBMSs**. In Proceedings of EEXTT 2002 and DiWeb 2002, Pages 162-164, 2002. Lecture Notes in Computer Science, Vol.2590.

[13] Benjamin Bin Yao, M. Tamer Ozsu, Nitin Khandelwal. **XBench Benchmark and Performance Testing of XML DBMSs**. In Proceedings of the 20th International Conference on Data Engineering, Page 621. IEEE Computer Society, 2004.

[14] XML JOURNAL. **A Comparison Of XML-Enabled And Native XML Data Management Techniques**. Available [On-line]: http://xml.sys-con.com/read/104980_p.htm [Last access: November 2005]

[15] Akmal B. Chaudhri, Awais rashid, and Roberto Zicari. **XML Data Management: Native XML and XML-Enabled Database Systems**. Addison-Wesley, 2003. Pages 519- 546.

[16] Akmal B. Chaudhri, Awais rashid, and Roberto Zicari. **XML Data Management: Native XML and XML-Enabled Database Systems**. Addison-Wesley, 2003. Pages 547- 565.

[17] Anand Vivek Srivastava. **Comparison and Benchmarking of Native XML Databases**. Available [On-line]: http://www.cse.iitk.ac.in/report-repository/2004/Y1043.pdf [Last access: November 2005]

[18] Courtois Vincent, Percot Patrick, Rospars Urvan. **Etude d'un Systeme de Recherche Documentarie Livrable**, Mars 2004. Available [On-line]: http://www-valoria.univ-ubs.fr/dess-asir/projets/courant/docs/SRD_livrable.pdf [Last access: November 2005]

[19] David M. Kroenke. **Database Processing Fundementals, Design and Implementation**. Prentice Hall, Upper Sadlle River, New Jersey, 2000. Page 237

[20] Ronald Bourret. **XML Databases Products: Native XML Databases**. Available [On-line]: http://www.rpbourret.com/xml/ProdsNative.htm [Last access: November 2005]

[21] **Apache Xindice**. Available [On-line]: http://xml.apache.org/xindice [Last access: November 2005]

[22] **eXist**. Available [On-line]: http://exist.sourceforge.net [Last access: November 2005]

[23] **dbXML-Native XML Database**. Available [On-line]: http://www.dbxml.com [Last access: November 2005]

[24] **SleepyCat: Products: Berkeley DB XML**. Available [On-line]: http://www.sleepycat.com/products/bdbxml.html [Last access: November 2005]

[25] Mark Graves. **Designing XML databases**. Prentice Hall, Upper Sadlle River, New Jersey, 2002. Pages 14-15

[26] Ronald Bourret. **XML and Databases**. Available [On-line]: http://www.rpbourret.com/xml/XMLAndDatabases.htm [Last access: November 2005]

[27] Akmal B. Chaudhri, Awais Rashid and Roberto Zicari. **XML Data Management: Native XML and XML-Enabled Database Systems**. Addison-Wesley, 2003. Page xxi.

[28] XNLT Software B.V. **Handling XML Documents using traditional databases**. Available [On-line]: http://www.surfnet.nl/innovatie/surfworks/xml/xml-databases.pdf July/ August 2002 [Last access: November 2005]

[29] Robert Cooney. **XML storage has a niche**. Available [On-line]: http://www.computerworld.com/printthis/2003/0,4814,83317,00.html [Last access: November 2005]

[30] Kimbro Staken. **Introduction to Native XML databases**, October 31, 2001. Available [On-line]: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html [Last access: November 2005]

[31] **XML:DB Database API Draft**. Available [On-line]: http://xmldb-org.sourceforge.net/xapi/xapi-draft.html [Last access: November 2005]

[32] Kimbro Staken. **Introduction to Native XML databases**, Januuary09, 2002. Available [On-line]: http://www.xml.com/pub/a/2002/01/09/xmldb_api.html [Last access: November 2005]

[33] Dave Addey, Chris Auld, Oli Gauti Gudmundsson, Jon James, Allan Kent, Jeff Rafter, Alex Shiell, Paul Spencer, Inigo Surguy. **Practical XML for the web**. Glasshaus, Acocks Green, Birmingham, 2002. Pages 38-39

[34] Alexiei. **Parser Evaluation**. Available [On-line]: http://www.dcs.shef.ac.uk/~alexiei/WebSite/University/MultiPLAT/WebDocs/AdditionalDocs/Parser%20Evaluation.pdf [Last access: November 2005]

[35] Ken Sall. **XML software Guide: XML Parsers**, July 29, 2000. Available [On-line]: http://wdvl.com/Software/XML/parsers.html [Last access: November 2005]

[36] W3C. **Document Object Model (DOM)**. Available at http://www.w3.org/DOM [Last access: November 2005]

[37] **XML Parser for Java**. Available [On-line]: http://www.lc.leidenuniv.nl/awcourse/oracle/appdev.920/a96621/adx04paj.htm [Last access: November 2005]

[38] Deteil, Dietel, Nieto, Lin & Sadhu. **XML: How to Program**. Prentice Hall, Upper Sadlle River, New Jersey, 2001. Pages 182-216

[39] SAX. **About SAX**. Available [On-line]: http://www.saxproject.org [Last access: November 2005]

[40] **Java Architecture for XML Binding (JAXB)**. Available [On-line]: http://java.sun.com/webservices/jaxb [Last access: November 2005]

[41] **Java API for XML Processing (JAXP)**. Available [On-line]: http://java.sun.com/webservices/jaxp [Last access: November 2005]

[42] **JDOM**. Available [On-line]: http://www.jdom.org [Last access: November 2005]

[43] **DOM4J**. Available [On-line]: http://www.dom4j.org [Last access: November 2005]

[44] W3C. **XML Path Language (Xpath)**. Available [On-line]: http://www.w3.org/TR/xpath [Last access: November 2005]

[45] Kal Ahmed, Danny Ayers, Mark Birbeck, Jay Cousins, David Dodds, Josh Lubell, Miloslav Nic, Daniel Rivers-Moore, Andrew Watt, Robert Worden, Ann Wrightson. **Professional XML Meta Data**. Wrox Press, Acocks Green, Birmingham, 2001. Pages 75-78

[46] **XPath Tutorial**. Available [On-line]: http://www.w3schools.com/xpath [Last access: November 2005]

[47] **XPath Tutorial**. Available [On-line]: http://www.zvon.org/xxl/XPathTutorial/General/examples.html [Last access: November 2005]

[48] **XML:DB Initiative:Xupdate-XML Update Language**. Available [On-line]: http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html [Last access: November 2005]

[49] Kimbro Staken. **XML:DB XUpdate Use Cases**. Available [On-line]: http://www.xmldatabases.org/projects/XUpdate-UseCases [Last access: November 2005]

[50] W3C. **XSL Transformations (XSLT)**. Available [On-line]: http://www.w3.org/TR/xslt [Last access: November 2005]

[51] W3C. **XML Schema**. Available [On-line]: http://www.w3.org/XML/Schema [Last access: November 2005]

[52] **DTD Tutorial**. Available [On-line]: http://www.w3schools.com/dtd/default.asp [Last access: November 2005]

[53] Whatis.com definition. **Open Source**. Available [On-line]: http://whatis.techtarget.com/definition/0,289893,sid9_gci212709,00.html [Last access: November 2005]

[54] David A. Wheeler. **How to Evaluate Open Source Software/ Free Software (OSS/FS) Programs**. Available [On-line]: http://www.dwheeler.com/oss_fs_eval.html [Last access: November 2005]

[55] **The Michigan Benchmark**. Available [On-line]: http://www.eecs.umich.edu/db/mbench [Last access: November 2005]

[56] Mark J. Heindl. **Native XML databases**. Available [On-line]: http://students.depaul.edu/~mheindl [Last access: April 2004]

[57] Werner Frieb. **XML Databases for Augmented Reality**. Available [On-line]: http://www.ims.tuwien.ac.at/media/documents/publications/Thesis_Frieb.pdf [Last access November 2005]

[58] **Freshmeat.net**. Available [On-line]: http://freshmeat.net [Last access: November 2005]

[59] **Freshmeat.net. About**. Available [On-line]: http://freshmeat.net/about [Last access: November 2005]

[60] Roy C. Hoobler. **Combining XML and databases**. Available [On-line]: http://builder.com.com/5100-6387-1044928.html [Last access: November 2005]

[61] Micheal S. Dougherty. **Web Developer: Are XML Databases Necessary?**. Available [On-line]: http://www.db2mag.com/db_area/archives/2003/q1/webdev.shtml [Last access: November 2005]

[62] **Frequently Asked Questions**. Available [On-line]: http://xml.apache.org/xindice/faq.html [Last access: November 2005]

[63] Wolfang Meier. **eXist: An Open S... ...ative XML Database**. Available [On-line]: http://exist-db.org/exist.pdf [Last access: November 2005]

[64] Kimbro Staken. **Berkeley DB XML Basics**, 13 February 2004. Available [On-line]: http://www.linux-mag.com/2004-02/berkeley_01.html [last access: November 2005]

[65] Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr, Robert C. Tekampe, Robert J. Watson. **Performance Measurement Methodologies for Database Systems**. In Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective: mid-80's perspective, Pages: 16-28. ACM Press, 1985.

[66] Arun Gaikwad. **Introduction to Xindice**, 01 September 2002. Available [On-line]: http://www-128.ibm.com/developerworks/web/library/wa-xindice.html [Last access: November 2005]

[67] Pierre Geneves. **Improving Efficiency of XPath-based XML Querying**, March 8, 2004. Available [On-line]: http://wam.inrialpes.fr/publications/2004/toward-xpath-efficiency.pdf [Last access: November 2005]

[68] Jack D. Herrington. **Getting the Most Out of XML with XPath**. Available [On-line]: http://www.devx.com/xml/Article/21930 [Last access: November 2005]

[69] Mike Jasnowski. **Writing a Native XML Database**. Available [On-line]: http://xml.sys-con.com/read/40514.htm [Last access: November 2005]

[70] Dale Waldt. GILBANE REPORT,White Paper.**Using XML and Databases**. Available [On-line]: http://www.gilbane.com/whitepapers/X-Hive/XML_DB_Gilbane_White_Paper_final.pdf [Last access: November 2005]

[71] N Mabanza, J Chadwick. **A Comparison of Open Source Native XML Database products**. In Proceedings of SATNAC (Southern African Telecommunication Networks & Applications Conference) 2004, Western Cape, South Africa, September 2004.

[72] Ntima Mabanza, Jim Chadwick, Radha Krishna Rao GSV. **Performance Evaluation of Open Source Native XML Databases – A Case Study**. In Proceedings of the 8th International Conference on Advanced Communication Technology (ICACT) 2006, Phoenix park, Korea , February 2006.

[73] Dan Farber. **Six barriers to open [illegible] adoption**, March 20, 2004. Available [On-line]: http://techupdate.zdnet.com/techupdate/stories/main/Six_barriers_to_open_source_adoption.html [Last access: November 2005]

Appendices

University of Fort Hare
Together in Excellence

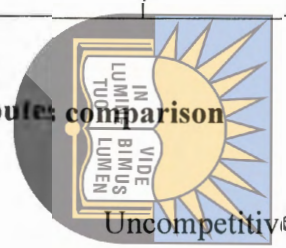
Appendix A

Products	Attributes								Total
	Functionality	Market share	Support	Maintenance	Scalability	Usability	Security	Flexibility	
Xindice	3	2	1	1	2	2	1	3	15
eXist	3	3	2	2	2	3	2	3	20
dbXML	3	1	1	1	2	3	2	3	16
Berkeley DB XML	3	1	3	2	2	2	2	3	19

Table 3.2 attributes comparison

Legends

Very Good: 3 Not Bad: 2 Uncompetitive: 1



Appendix B

B.1. Java Code for generating part elements

```
// This program Create an XML file and generate a random number of items in the file

import java.io.*;

public class CreateXML {
        public static String makeXML(int sku) {

                String str = " <part sku=" + "\"" + sku + "\"" +">\n";
        str += "<desc> Ball Bearing </desc>\n";
        str += "<maker> S.K.F. </maker>\n";
        str += "<instock> Yes </instock>\n";
        str += "<price> $20.00 </price>\n";
        str += "</part>\n";
           return(str);
           }
           public static void makeXMLFile(String filename, int num) {
                   try {
                           FileWriter f = new FileWriter(filename);
                           PrintWriter p = new PrintWriter(f);
                           p.println("<?xml version=\"1.0\" ?>\n");
                           p.write ("<parts>\n");
                           for(int k = 0; k < num; k++) {
                                   int sku = (int) (10000 * Math.random());
                                   p.println(makeXML(sku));
                                   }
                           p.println("</parts>");
                           p.close();
                           f.close();
           }
                   catch(Exception ex) {    }
           }
                   public static void main(String[] args) {
                   if(args.length < 1) {
                           System.out.println("USAGE: java CReateXML size");
                           System.exit(0);
                   }
                   int size = Integer.parseInt(args[0]);
                   makeXMLFile("jim.xml", size);
           }
}
```

B.2. Databases Code used for storage operation

B.2.1. Berkeley Database

```
// Timing Insertion Operation

import com.sleepycat.db.*;
import com.sleepycat.dbxml.*;
import java.util.*;
import java.io.*;


public class Insertion
{
        public static void main(String[] args) {

                long t1,t2,total;
                XmlDocument document = null;

                try {
                        FileWriter f = new FileWriter("DBXMLCreationXlarge1.txt");
                        PrintWriter p = new PrintWriter(f);

                        String data = readFileFromDisk(args[0]);
                        for(int Size = 100; Size <= 1000; Size += 100) {

                         String dbname = "db" + Size + ".dbxml";

                         long sumTimes = 0;
                         int num = 3;

                        for(int counter = 0; counter < num; counter++) {
                XmlContainer container = new XmlContainer(null, dbname +  "-" + counter, 0);

                        container.open(null, Db.DB_CREATE, 0);

                        document = new XmlDocument();
                        byte[] raw_content = data.getBytes();
                        document.setContent(raw_content);



                        Date date = new Date();
                        t1 = date.getTime();
                for(int k = 0; k < Size; k++){
```

```
                container.putDocument(null, document, null, 0);

                }
                container.close(0);

                date= new Date();
                t2 = date.getTime();

                total = t2 - t1;
                sumTimes += total;

                }

                double avgTime = (double) sumTimes / num;

                p.println(Size + "," + (avgTime/1000));
        }
                p.close();
                f.close();

        } catch (XmlException e) {
                System.err.println("Got an XML exception: " + e);
                e.printStackTrace();
        }
        catch(Exception e) {
                System.out.println(e);
                        }
    }
    public static String readFileFromDisk(String fileName) throws Exception {
File file = new File(fileName);
FileInputStream insr = new FileInputStream(file);
byte[] fileBuffer = new byte[(int) file.length()];
insr.read(fileBuffer);
insr.close();
return new String(fileBuffer);

}
}
```

B.2.2. dbXML database

// Timing for insertion operations

```
import com.dbxml.db.core.*;
import com.dbxml.db.client.CollectionClient;
import com.dbxml.db.client.dbXMLClient;
import com.dbxml.db.client.local.dbXMLClientImpl;
import com.dbxml.util.dbXMLException;
import com.dbxml.db.embedded.*;
import com.dbxml.xml.dom.DOMHelper;
import java.io.*;
import org.w3c.dom.*;
import java.util.Properties;
import java.io.File;
import java.io.FileInputStream;
import java.util.*;


public class Insertion {

        private static org.w3c.dom.Document getConfig(String name){
                org.w3c.dom.Document doc=null;
                String CLASS = "class";
                String DEFAULT_CLASS =
com.dbxml.db.server.dbXMLContainer";
                String PAGESIZE = "pagesize";
                String DEFAULT_PAGESIZE = "4096";
                String TYPE = "type";
                String DEFAULT_TYPE = "compressed";
                Properties props = new Properties();
                props.put(CLASS, DEFAULT_CLASS);
                props.put(PAGESIZE, DEFAULT_PAGESIZE);
                props.put(TYPE, DEFAULT_TYPE);
                try {
                        String cname = props.getProperty(CLASS,DEFAULT_CLASS);
                        props.remove(CLASS);
                        String type = props.getProperty(TYPE,DEFAULT_TYPE);
                        props.remove(TYPE);
                        doc = com.dbxml.xml.dom.DOMHelper.newDocument();
                        org.w3c.dom.Element root = doc.createElement("collection");
                        root.setAttribute("name", name);
                        root.setAttribute("type", type);
                        doc.appendChild(root);
                        org.w3c.dom.Element filer = doc.createElement("filer");
                        filer.setAttribute("class", cname);
                        java.util.Iterator iter = props.entrySet().iterator();
```

```
                while ( iter.hasNext() ) {
                        java.util.Map.Entry entry = (java.util.Map.Entry)iter.next();
                        String key = (String)entry.getKey();
                        String value = (String)entry.getValue();
                        filer.setAttribute(key, value);
                        }
                root.appendChild(filer);
                root.appendChild(doc.createElement("indexes"));
                root.appendChild(doc.createElement("triggers"));
                root.appendChild(doc.createElement("extensions"));
                }
         catch (Exception e) {
                e.printStackTrace();
                }
        return doc;
        }

public static void main(String[] args) {

                long t1,t2,total;
   try {
                FileWriter f = new FileWriter("dbXMLDBcreationxXlarge1.txt");
                PrintWriter p = ne          ter(f);
        try {
                File dataDir = new File("./db");
EmbeddedConfiguration cfg = new EmbeddedConfiguration("db", dataDir);
                EmbeddedDatabase embdb = new EmbeddedDatabase(cfg);
                embdb.createCollection("data",cfg.getConfig());
                dbXMLClient client = new dbXMLClientImpl();
                CollectionClient col = client.getDatabase();

                String data = readFileFromDisk(args[0]);

                for (int Size = 100; Size <= 1000; Size+= 100) {

                long sumTimes = 0;
                int num = 3;

                for(int count = 0; count < num; count++) {
                String COLLECTION_NAME = "mycollection" + Size;

col.createCollection(COLLECTION_NAME, getConfig(COLLECTION_NAME + "-" +
count));

CollectionClient cc = client.getCollection("/" + COLLECTION_NAME + "-" + count);
```

```
                Date date = new Date();
                t1 = date.getTime();
        for(int k = 0; k < Size; k++)
                cc.insertDocumentAsText(data);

                date = new Date();
                t2 = date.getTime();

                total = t2 - t1;
                sumTimes += total;
                }
                double avgTime = (double) sumTimes / num;
                p.println(Size + "," + ((double) avgTime/1000));
                        }
                p.close();
                f.close();

                System.exit(0);
        }
        catch (Exception ex) {
                System.out.println(ex);
                ex.printStackTrace();
                }
        } catch (IOException e) {
                System.out.println(e);
                }
        }

        public static String readFileFromDisk(String fileName) throws Exception
{
    File file = new File(fileName);
    FileInputStream insr = new FileInputStream(file);
    byte[] fileBuffer = new byte[(int) file.length()];
    insr.read(fileBuffer);
    insr.close();
    return new String(fileBuffer);
       }
       }
```

B.2.3. eXist database

```
// Timing for insertion operations

import org.exist.xmldb.*;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import java.io.*;
import java.util.Date;

public class Insertion {

        protected static void usage() {
                System.out.println("usage: org.exist.examples.xmldb.Put docName");
                System.exit(0);
        }

        public static void main(String args[]) throws Exception {
                if(args.length < 1)
                        usage();

                String  file = args[0];

                String driver = "org.exist.xmldb.DatabaseImpl";
                Class cl = Class.forName(driver);
                Database database = (Database) cl.newInstance();
                database.setProperty("create-database", "true");
                DatabaseManager.registerDatabase(database);

                Collection col = null;
                XMLResource document= null;
                CollectionManagementService mgtService = null;
                 long t1,t2,total;
        try {
                        FileWriter f = new FileWriter("ExistDBcreationXXLarge1a.txt");
                        PrintWriter p = new PrintWriter(f);
         try {
                        String xml = readFile(file);



                        for(int Size = 100; Size <= 1000;Size += 100) {

                        long sumTimes = 0;
                        int num = 3;
```

```
                    for(int count = 0; count < num; count++) {

                    String collection = "mycollection" + Size;

        col = DatabaseManager.getCollection("xmldb:exist:///db/" + collection +  "-" +
count,"admin", null);


                if(col == null) {

        Collection root = DatabaseManager.getCollection("xmldb:exist:///" + "db" + "/"
        ,"admin", null);

        mgtService =
(CollectionManagementService)root.getService("CollectionManagementService", "1.0");
        col = mgtService.createCollection(collection +  "-" + count);

            }

                document = (XMLResource)col.createResource(null, "XMLResource");
                        document.setContent(xml);

                        Date date = new Date();
                        t1 = date.getTime();

                for(int k = 0; k < Size; k++){
                        col.storeResource(document);
                        }
                        date = new Date();
                        t2 = date.getTime();

                        total = t2 - t1;
                        sumTimes += total;
                        }
                        double avgTime = (double) sumTimes / num;
                        p.println(Size + "," + ((double) avgTime/1000));

        DatabaseInstanceManager manager =
(DatabaseInstanceManager)col.getService("DatabaseInstanceManager","1.0");
                manager.shutdown();
                }
                        p.close();
                        f.close();

            } catch (XMLDBException e) {
```

```
System.err.println("XML:DB Exception occured " + e.errorCode + " " + e.getMessage());
                }

    } catch (IOException e) {
                        System.out.println(e);
                                }
        }

protected static String readFile(String fileName ) throws IOException {

        File file = new File(fileName);
        FileInputStream insr = new FileInputStream(file);
        byte[] fileBuffer = new byte[(int) file.length()];
        insr.read(fileBuffer);
         insr.close();
         return new String(fileBuffer);
   }
}
```

B.2.4. Xindice database

```
// Timing for insertion operations

import java.io.FileInputStream;
import java.io.*;
import java.util.*;

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XMLResource;
import org.apache.xindice.client.xmldb.services.CollectionManager;
import org.apache.xindice.xml.dom.DOMParser;

public class Insertion  extends AbstractExample {

   public static void main(String[] args) throws Exception {

      Collection collection = null;
      XMLResource document = null;
      long t1,t2,total;

      try {
                        FileWriter f = new
                        FileWriter("XINDICEDBcreationXXLarge1.txt");
                        PrintWriter p = new PrintWriter(f);

      try {
                        String data = readFileFromDisk(args[0]);

                        for(int Size = 100; Size <= 1000;Size += 100) {

                        long sumTimes = 0;
                        int num = 3;

                        for(int count = 0; count < num; count++) {

                        String COLLECTION_NAME = "mycollection" + Size;

                         collection = getCollection("xmldb:xindice-embed:///db/");

                        CollectionManager service = (CollectionManager)
                        collection.getService("CollectionManager", "1.0");

                        String collectionConfig =
          "<collection compressed=\"true\" name=\"" + COLLECTION_NAME +  "-"
       + count + "\">"  + "   <filer class=\"org.apache.xindice.core.filer.BTreeFiler\"/>"
```

```
        + "</collection>";
    service.createCollection(COLLECTION_NAME +  "-" + count,
    DOMParser.toDocument(collectionConfig));



    collection = getCollection("xmldb:xindice-embed:///db/" +
    COLLECTION_NAME +  "-" + count);

    document = (XMLResource) collection.createResource(null, "XMLResource");
    document.setContent(data);


                    Date date = new Date();
                    t1 = date.getTime();
    for(int k = 0; k < Size; k++){
                   collection.storeResource(document);
    }
                    date = new Date();
                    t2 = date.getTime();

                    total = t2 - t1;
                    sumTimes += tot
                    }
                    double avgTime = (double) sumTimes / num;
                    p.println(Size + "," + ((double) avgTime/1000));

  }
                    p.close();
                    f.close();

  } catch (XMLDBException e) {
System.err.println("XML:DB Exception occured " + e.errorCode + " " + e.getMessage());

                                                    }

  finally {
    if (collection != null){
      collection.close();
    }
  }

} catch (IOException e) {
                    System.out.println(e);
                                    }
```

```
    }

  public static String readFileFromDisk(String fileName) throws Exception {
    File file = new File(fileName);
    FileInputStream insr = new FileInputStream(file);
    byte[] fileBuffer = new byte[(int) file.length()];
    insr.read(fileBuffer);
    insr.close();
    return new String(fileBuffer);

}

}
```

B.3. List of query operations considered in the performance test

B.3.1. Berkeley DB XML

```
//Get query-service     (Query formulation)
                        //1. General Queries
 results = container.queryWithXPath(null,"/parts", context, 0); //1.a
 results = container.queryWithXPath(null,"/parts/part", context, 0);//1.b
results = container.queryWithXPath(null,"/parts/part/@sku", context, 0);//1.c

                                //2. Query path and conditional path
results = container.queryWithXPath(null,"/parts/part[@sku > 2414]", context, 0);//2.a

                        //3. Queries by position within the stored XML document
                                //Element
results = container.queryWithXPath(null,"/parts/part[1]/desc", context, 0);//3.a
results = container.queryWithXPath(null,"/parts/part[75]/desc", context, 0);//3.b
results = container.queryWithXPath(null,"/parts/part[last()]/desc", context, 0);//3.c

                                //Attribute
results =
        container.queryWithXPath(null, [@sku='3119']/desc", context, 0);//3.d
results =
        container.queryWith[illegible]/desc", context, 0);//3.e
results =
        container.queryWithXPath(null,"/parts/part[@sku='8834']/desc", context, 0);//3.f

                        //4. Queries by operators
results =
container.queryWithXPath(null,"/parts/part[@sku ='7064' and maker='S.K.F.']", context,
0);//4.a
results =
container.queryWithXPath(null,"/parts/part[@sku ='7199' or @sku='3191']", context,
0);//4.b
results =
container.queryWithXPath(null,"/parts/part[@sku !='5604']", context, 0);//4.c //consider

                                //5. Queries by missing information
                                        // Missing element
results = container.queryWithXPath(null,"/parts/part[178]/price", context, 0);//5.a
                                        //Missing attribute
results = container.queryWithXPath(null,"/parts/part[@sku='107']", context, 0);//5.b
```

The queries listed above were inserted into the database code (Berkeley DB XML) and execute one at a time.

B.4. Database Code used for query operation

B.4.1. Berkeley DB XML

```
// Timing for query operations

import com.sleepycat.db.*;
import com.sleepycat.dbxml.*;
import java.util.*;
import java.io.*;

public class Query {
        public static void main(String[] args) throws XmlException {

                long t1,t2,total=0;
                XmlResults results = null;

    try {

                        FileWriter f = new FileWriter("DBXMLQuery5b.txt");
                        PrintWriter p = new PrintWriter(f);

        try {
                for(int Size = 100; Size <= 1000; Size += 100) {
                        String dbname = "db" + Size + ".dbxml";
                XmlContainer container = new XmlContainer(null, dbname +  "-0", 0);;
                        container.open(null,0, 0);
        XmlQueryContext context = new
XmlQueryContext(XmlQueryContext.ResultValues, XmlQueryContext.Eager);

                        Date date = new Date(); // create object of the class
                        t1 = date.getTime();


                        for(int k = 0; k < 5; k++){

results = container.queryWithXPath(null,"/parts/part[@sku='107']", context, 0);//5.b

                for (XmlValue value; (value = results.next(null)) != null; ) {
                        }
                        }
                        container.close(0);

                        date = new Date();
                        t2 = date.getTime();
                        total = t2 - t1;
```

```
                        p.println(Size + "," + (double)(total/1000));
        }
                        p.close();
                        f.close();

} catch (XmlException e) {
                        System.err.println("Got an XML exception: " + e);
                        e.printStackTrace(System.err);
}
        } catch (IOException e) {
                        System.out.println(e);
                                }
}
}
```

B.5. List of query operations considered in the performance test

B.5.1. dbXML, eXist, and Xindice

```
String query = "/parts/part";//1.b
String query = "/parts/part/desc";//1.b1 later

                                    //Attribute
String query = "/parts/part/@sku"; //1.c
String query = "/parts/part/@sku/desc";//1.c1//not working

                                    //2.Query Path and Conditional path
String query = "/parts/part[@sku >'2414']"; //2.a

                                    // 3.Queries by Positon within the stored XML file
                                    //Element
String query = "/parts/part[1]/desc"; //3.a
String query = "/parts/part[75]/desc";//3.b
String query = "/parts/part[150]/desc"; //3.c
String query = "/parts/part[last()]/desc"; //3.c check later not working

                                    //Attribute
String query = "/parts/part[@sku='3119']/desc"; //3.d
String query = "/parts/part[@sku='7199']/desc";//3.e
String query = "/parts/part[@sku='8834']/desc";//3.f

                                // 4.Queries by Operators
String query = "/parts/part[@sku='7064'and maker='S.K.F.']"; // 4.a
String query = "/parts/part[@sku='7199'or @sku='3191']";//4.b
String query = "/parts/part[@sku!='5604']"; //4.c

                                //5.Queries by Missing Information
                                    //Missing element
String query = "/parts/part[178]/price"; // 5.a Missing element
                                    //Missing attribute
String query = "/parts/part[@sku='107']/desc"; // 5.b Missing attribute
```

The queries listed above were inserted into the database code (dbXML, eXist, and Xindice) and execute one at a time.

B.6. Database Codes used for query operation

B.6.1. dbXML database

```
// Timing for query operations

import com.dbxml.db.core.*;
import com.dbxml.db.client.CollectionClient;
import com.dbxml.db.client.dbXMLClient;
import com.dbxml.db.client.local.dbXMLClientImpl;
import com.dbxml.util.dbXMLException;
import com.dbxml.db.embedded.*;
import com.dbxml.xml.dom.DOMHelper;
import java.io.*;
import org.w3c.dom.*;
import java.util.Properties;
import com.dbxml.db.client.ResultSetClient;
import java.util.HashMap;
import java.io.File;
import java.io.FileInputStream;
import java.util.*;

public class Query {

        private static org.w3c.dom.Document getConfig(String name){
                org.w3c.dom.Document doc=null;
                String CLASS = "class";
                String DEFAULT_CLASS =
com.dbxml.db.server.dbXML.DEFAULT_FILER;
                String PAGESIZE = "pagesize";
                String DEFAULT_PAGESIZE = "4096";
                String TYPE = "type";
                String DEFAULT_TYPE = "compressed";
                Properties props = new Properties();
                props.put(CLASS, DEFAULT_CLASS);
                props.put(PAGESIZE, DEFAULT_PAGESIZE);
                props.put(TYPE, DEFAULT_TYPE);
                try {
                        String cname = props.getProperty(CLASS,DEFAULT_CLASS);
                        props.remove(CLASS);
                        String type = props.getProperty(TYPE,DEFAULT_TYPE);
                        props.remove(TYPE);
                        doc = com.dbxml.xml.dom.DOMHelper.newDocument();
                        org.w3c.dom.Element root = doc.createElement("collection");
                        root.setAttribute("name", name);
                        root.setAttribute("type", type);
```

```
                doc.appendChild(root);
                org.w3c.dom.Element filer = doc.createElement("filer");
                filer.setAttribute("class", cname);
                java.util.Iterator iter = props.entrySet().iterator();
                while ( iter.hasNext() ) {
                        java.util.Map.Entry entry = (java.util.Map.Entry)iter.next();
                        String key = (String)entry.getKey();
                        String value = (String)entry.getValue();
                        filer.setAttribute(key, value);
                        }
                root.appendChild(filer);
                root.appendChild(doc.createElement("indexes"));
                root.appendChild(doc.createElement("triggers"));
                root.appendChild(doc.createElement("extensions"));
                }
        catch (Exception e) {
                e.printStackTrace();
                }
        return doc;
        }

public static void main(String[] args) {

                long t1,t2,total=0;
                CollectionClient cc = null;
                ResultSetClient rs= null;
try {
                FileWriter f = new FileWriter("dbXMLDBquery1b1.txt");
                PrintWriter p = new PrintWriter(f);
    try {
                File dataDir = new File("./db");
EmbeddedConfiguration cfg = new EmbeddedConfiguration("db", dataDir);
                EmbeddedDatabase embdb = new EmbeddedDatabase(cfg);
                embdb.createCollection("data",cfg.getConfig());

                dbXMLClient client = new dbXMLClientImpl();
                CollectionClient col = client.getDatabase();

                for (int Size = 100; Size <= 1000; Size+= 100) {

                String COLLECTION_NAME = "mycollection" + Size;

                cc = client.getCollection("/" + COLLECTION_NAME + "-0");

                HashMap nsMap = new HashMap();
```

```
        String query = "/parts"; //1.a

        Date date = new Date();
        t1 = date.getTime();

        for(int k = 0; k < 5; k++){
        rs = cc.queryCollection("XPath", query, nsMap);

        while ( rs.next() ) {
         rs.getResultAsText();
        }

        rs.close();
        }

        date = new Date();
        t2 = date.getTime();

        total = t2 - t1;

        p.println(Size + "," + ((double)(total/1000)));
        }
        p.close();
        f.close();
        }
    catch (Exception ex) { }

    } catch (IOException e) {
        System.out.println(e);
                    }
    System.exit(0);
    }
}
```

B.6.2. eXist database

```
// Timing for query operations

import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.modules.XMLResource;
import org.xmldb.api.modules.XPathQueryService;
import java.util.Date;
import java.io.*;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.*;

public class Query {

        protected static String URI = "xmldb:exist:///db/";
        protected static String driver = "org.exist.xmldb.DatabaseImpl";

    public static void main( String args[] ) {
        Collection col = null;
        ResourceSet result= null;
        ResourceIterator resourceIterator=null;
        XMLResource resource = null;
         long t1,t2,total=0;
try {
        Class cl = Class.forName( driver );
        Database database = (Database) cl.newInstance();
        database.setProperty( "create-database", "true" );
        DatabaseManager.registerDatabase( database );
try {
                        FileWriter f = new FileWriter("EXISTDBquery1005b.txt");
                        PrintWriter p = new PrintWriter(f);
        try {

                        for(int Size = 100; Size <= 1000;Size += 100) {
                String COLLECTION_NAME = "mycollection" + Size + "-0";

    col = DatabaseManager.getCollection(URI + COLLECTION_NAME,"admin", null);

                String query = "/parts"; //1.a

        XPathQueryService service =
                (XPathQueryService) col.getService( "XPathQueryService", "1.0" );
```

```
                    Date date = new Date();
                    t1 = date.getTime();

for(int k = 0; k < 5; k++){

                    result = service.query(query);
                    resourceIterator = result.getIterator();

          while (resourceIterator.hasMoreResources()) {

                    resource = (XMLResource) resourceIterator.nextResource();
     }
                    }

                    date = new Date();
                    t2 = date.getTime();

                    total = t2 - t1;

p.println(Size + "," + ((double)(total/1000)));
                    }
                    p.close();
                    f.close();

} catch (Exception ex) {
          System.out.println(ex); ex.printStackTrace();
                    }

} catch (IOException e) {
                              System.out.println(e);
                                        }
} catch ( Exception e ) {
e.printStackTrace();
}
System.exit(0);
}

}
```

B.6.3. Xindice database

// Timing for query operations

```
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XPathQueryService;
import java.util.*;
import java.io.*;

public class Query extends AbstractExample {

   public static void main(String[] args) throws Exception {
                        Collection collection = null;
                        ResourceSet resourceSet = null;
                        ResourceIterator resourceIterator = null;
                        Resource resource = null;
                        long t1,t2,total=0;

     try {
                        FileWriter f = new FileWriter("XINDICEDBquery1005b.txt");
                        PrintWriter p = new PrintWriter(f);
     try {

               for(int Size = 100; Size <= 1000;Size += 100) {
                        String COLLECTION_NAME = "mycollection" + Size;

                        collection =
getCollection("xmldb:xindice-embed:///db/"+ COLLECTION_NAME +  "-0");

                        String xpath = "/parts/part[@sku >'2414']"; //2.a

                        XPathQueryService service =
               (XPathQueryService) collection.getService("XPathQueryService", "1.0");

                        Date date = new Date();
                        t1 = date.getTime();

                for(int k = 0; k < 5; k++){

                        resourceSet = service.query(xpath);
                        resourceIterator = resourceSet.getIterator();
```

```
                while (resourceIterator.hasMoreResources()){
                        resource = resourceIterator.nextResource();
                        resource.getContent();
                    }
        }

                        date = new Date();
                        t2 = date.getTime();

                        total = t2 - t1;

                        p.println(Size + "," + ((double)(total/1000)));

                        }

                        p.close();
                        f.close();

    } catch (XMLDBException e) {
System.err.println("XML:DB Exception occured " + e.errorCode + " " + e.getMessage());
    } finally {
       if (collection != null) {
          collection.close();
       }
    }
  } catch (IOException e) {
                        System.out.println(e);
                                }
}
}
```

B.7. Modification operations considered in the performance test

B.7.1. dbXML, eXist, and Xindice

```
String xupdate = "<xu:modifications version=\"1.0\""
        + "xmlns:xu=\"http://www.xmldb.org/xupdate\">"

        //(1.1) Element Value Modification
        // 1.a
        + "<xu:update select=\"/parts/part[1]/price\">"
        + "$400.00"
        + "</xu:update>"
        //1.b
        + "<xu:update select=\"/parts/part[75]/price\">"
        + "$700.00"
        + "</xu:update>"
        //1.c
        + "<xu:update select=\"/parts/part[150]/price\">"
        + "$540.00"
        + "</xu:update>"

        //(1.2)Attribute Value Modification
        //1.d
        + "<xu:update select=\"/parts/part/@sku[.='8134']\">"
        + "65"
         + "</xu:update>"
        //1.e
         + "<xu:update select=\"/parts/part/@sku[.='7199']\">"
        + "509"
        + "</xu:update>"
        //1.f
        + "<xu:update select=\"/parts/part/@sku[.='8834']\">"
         + "2636"

        + "</xu:update>"
```

The Modification operations listed above were inserted into the database code (dbXML, eXist, and Xindice) and execute one at a time.

B.8. Delete operations considered in the performance test

B.8.1. dbXML, eXist, and Xindice

```
String xupdate = "<xu:modifications version=\"1.0\""
        + " xmlns:xu=\"http://www.xmldb.org/xupdate\">"

        // (2)Delete [delete an element and to delete attribute]
                //(2.1)Element
        //1.a
        // + "<xu:remove select=\"/parts/part[1]/price\"/>"

        //1.b
        // + "<xu:remove select=\"/parts/part[75]/price\"/>"

        //1.c
        //  + "<xu:remove select=\"/parts/part[150]/price\"/>"

                //(2.2)Atribute
        //2a
        // + "<xu:remove select=\"/parts/part/@sku[.='65']\"/>"

        //2.b
        // + "<xu:remove select=\"/parts/part/@sku[.='1a9']\"/>"

         //2.c
        //  + "<xu:remove select=\"/parts/part/@sku[.='2623']\"/>"

        + "</xu:modifications>";
```


The delete operations listed above were inserted into the database code (dbXML, eXist, and Xindice) and execute one at a time.

B.9. Database Codes used for modification and delete operation

B.9.1. dbXML database

// Timing for modification and delete operations

```
import com.dbxml.db.core.*;
import com.dbxml.db.client.CollectionClient;
import com.dbxml.db.client.dbXMLClient;
import com.dbxml.db.client.local.dbXMLClientImpl;
import com.dbxml.util.dbXMLException;
import com.dbxml.db.embedded.*;
import com.dbxml.xml.dom.DOMHelper;
import java.io.*;
import org.w3c.dom.*;
import java.util.Properties;
import com.dbxml.db.client.ResultSetClient;
import java.util.HashMap;
import java.io.File;
import java.io.FileInputStream;
import com.dbxml.db.client.xmldb.*;
import java.util.*;
```



```
public class Update {
        private static org.w3c.dom.Document getConfig(String name){
                org.w3c.dom.Document doc=null;
                String CLASS = "class";
                String DEFAULT_CLASS =
com.dbxml.db.server.dbXML.DEFAULT_FILER;
                String PAGESIZE = "pagesize";
                String DEFAULT_PAGESIZE = "4096";
                String TYPE = "type";
                String DEFAULT_TYPE = "compressed";
                Properties props = new Properties();
                props.put(CLASS, DEFAULT_CLASS);
                props.put(PAGESIZE, DEFAULT_PAGESIZE);
                props.put(TYPE, DEFAULT_TYPE);
                try {
                        String cname = props.getProperty(CLASS,DEFAULT_CLASS);
                        props.remove(CLASS);
                        String type = props.getProperty(TYPE,DEFAULT_TYPE);
                        props.remove(TYPE);
                        doc = com.dbxml.xml.dom.DOMHelper.newDocument();
                        org.w3c.dom.Element root = doc.createElement("collection");
```

```
                root.setAttribute("name", name);
                root.setAttribute("type", type);
                doc.appendChild(root);
                org.w3c.dom.Element filer = doc.createElement("filer");
                filer.setAttribute("class", cname);
                java.util.Iterator iter = props.entrySet().iterator();
                while ( iter.hasNext() ) {
                        java.util.Map.Entry entry = (java.util.Map.Entry)iter.next();
                        String key = (String)entry.getKey();
                        String value = (String)entry.getValue();
                        filer.setAttribute(key, value);
                        }
                root.appendChild(filer);
                root.appendChild(doc.createElement("indexes"));
                root.appendChild(doc.createElement("triggers"));
                root.appendChild(doc.createElement("extensions"));
                }
        catch (Exception e) {
                e.printStackTrace();
                }
        return doc;
        }

  public static void main(String[] args) {
        long t1,t2,total=0;
        CollectionClient cc = null;
try {

                FileWriter f = new FileWriter("dbXMLDBupdate1a.txt");
                PrintWriter p = new PrintWriter(f);

        try {
                File dataDir = new File("./db");
  EmbeddedConfiguration cfg = new EmbeddedConfiguration("db", dataDir);
                EmbeddedDatabase embdb = new EmbeddedDatabase(cfg);
                embdb.createCollection("data",cfg.getConfig());

                dbXMLClient client = new dbXMLClientImpl();
                CollectionClient col = client.getDatabase();

                for (int Size = 100; Size <= 1000; Size+= 100) {

                String COLLECTION_NAME = "mycollection" + Size;
```

```
            cc = client.getCollection("/" + COLLECTION_NAME + "-0");

            String xupdate = "<xu:modifications version=\"1.0\""
     + "      xmlns:xu=\"http://www.xmldb.org/xupdate\">"

    + "<xu:update select=\"/parts/part[1]/price\">" // 1.a
    + "$400.00"
    + "</xu:update>"

    + "</xu:modifications>";

CollectionImpl impl = new CollectionImpl(cc);
 XUpdateQueryServiceImpl service = new XUpdateQueryServiceImpl(impl);

            Date date = new Date();
            t1 = date.getTime();

     for( int k=0; k<5; k++){
            service.update(xupdate);
    }
            date = new Date();
            t2 = date.getTime();

            total = t2 - t1;
    p.println(Size + "," + ((double)(total/1000)));

            }
            p.close();
            f.close();

            }
    catch (Exception ex) { }

    } catch (IOException e) {
            System.out.println(e);
                        }
    System.exit(0);
    }
    }
```

B.9.2. eXist database

```
// Timing for modification and delete operations

import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XUpdateQueryService;
import java.util.Date;
import java.io.*;

public class Update {

        protected static String URI = "xmldb:exist:///db/";
        protected static String driver = "org.exist.xmldb.DatabaseImpl";


    public static void main( String args[] ) {
        Collection col = null;
         long t1,t2,total=0;
        long count=0;
     try {
        Class cl = Class.forName( driver );
        Database database = (Database) cl.newInstance();
        database.setProperty( "create-database", "true" );
        DatabaseManager.registerDatabase( database );
    try {
                        FileWriter f = new FileWriter("EXISTDBupdate1a.txt");
                        PrintWriter p = new PrintWriter(f);
        try {

                        for(int Size = 100; Size <= 1000;Size += 100) {

                        String COLLECTION_NAME = "mycollection" + Size + "-0";

col = DatabaseManager.getCollection(URI + COLLECTION_NAME,"admin", null);

        String xupdate =  "<xu:modifications version=\"1.0\""
                        + "  xmlns:xu=\"http://www.xmldb.org/xupdate\">"

                        + "<xu:update select=\"/parts/part[1]/price\">" // 1.a
                        + "$500.00"
                        + "</xu:update>"

                        + "</xu:modifications>";
```

```
        XUpdateQueryService service =
(XUpdateQueryService) col.getService("XUpdateQueryService", "1.0");

                        Date date = new Date();
                        t1 = date.getTime();

                         for( int k=0; k<5; k++){

                        service.update(xupdate);
                        }

                        date = new Date();
                        t2 = date.getTime();

                        total = t2 - t1;
                p.println(Size + "," + ((double)(total/1000)));

                        }
                        p.close();
                        f.close();

     } catch (Exception ex) {
       System.out.println(ex); ex.printSt
                        } finally {
        if (col != null) {
           col.close();
        }
       }
     } catch (IOException e) {
                        System.out.println(e);
                                        }
     } catch ( Exception e ) {
       e.printStackTrace();
      }

     System.exit(0);

   }
}
```

B.9.3. Xindice database

```
// Timing for modification and delete operations

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XUpdateQueryService;
import java.util.*;
import java.io.*;


public class Update  extends AbstractExample {

   public static void main(String[] args) throws Exception {
      Collection collection = null;
      long t1,t2,total;

      try {

                        FileWriter f = new FileWriter("update1a.txt");
                        PrintWriter p = new PrintWriter(f);

      try {

                for(int Size = 100; Size <= 1000;Size += 100) {
                        String COLLECTION_NAME = "mytestCollection" + Size;

                        collection =
getCollection("xmldb:xindice-embed:///db/"+ COLLECTION_NAME +  "-0");

                        String xupdate = "<xu:modifications version=\"1.0\""
                         + " xmlns:xu=\"http://www.xmldb.org/xupdate\">"

                        + "<xu:update select=\"/parts/part[1]/price\">"
                        + "$400.00"
                        + "</xu:update>" // 1.a

                        + "</xu:modifications>";

        XUpdateQueryService service =
(XUpdateQueryService) collection.getService("XUpdateQueryService", "1.0");

                        Date date = new Date(); // create object of the class
                        t1 = date.getTime();

                        for( int k=0; k < 5; k++){
```

```
                    service.update(xupdate);

                    }

                    date = new Date();
                    t2 = date.getTime();

                    total = t2 - t1;

                    p.println(Size + "," + ((double)(total/1000)));
                    }
                    p.close();
                    f.close();
    } catch (XMLDBException e) {
System.err.println("XML:DB Exception occured " + e.errorCode + " " + e.getMessage());
    } finally {
      if (collection != null) {
        collection.close();
      }
    }
  } catch (IOException e) {
                    System.out.println(e);
                    }
  }
}
```

APPENDIX C

C.1. Timing Results for storage operations

(See subsections 5.3.2.1, 5.3.2.2, and 5.3.2.3)

C.1.1. Berkeley DB XML

Size	50 generated part items	150 generated part items	100 generated part items
100	0.64	1.19	3.67
200	1.05	1.80	7.25
300	1.47	2.58	9.16
400	1.83	2.84	10.51
500	1.86	3.63	11.63
600	2.03	4.09	13.02
700	2.34	4.86	14.86
800	2.81	5.23	17.04
900	2.79	5.66	20.36
1000	3.25		21.31

C.1.2. dbXML

Size	50 generated part items	150 generated part items	100 generated part items
100	6.67	8.32	13.11
200	11.90	14.33	28.47
300	19.39	23.18	43.12
400	26.09	31.38	55.18
500	33.29	37.36	73.70
600	38.44	42.54	90.69
700	44.07	51.80	102.28
800	53.46	61.08	126.10
900	65.12	73.23	137.06
1000	68.63	79.65	148.01

C.1.3. eXist

Size	50 generated part items	150 generated part items	100 generated part items
100	17.14	21.22	64.14
200	33.86	41.46	125.99
300	50.33	61.43	187.73
400	66.99	81.29	248.84
500	83.93	100.74	321.05
600	100.24	122.40	
700	117.06	141.89	
800	133.89	163.78	
900	150.53	184.20	
1000	167.42	204.58	

C.1.4. Xindice

Size	50 generated part items	150 generated part items	100 generated part items
100	3.63	6.72	25.43
200	7.05	13.16	50.94
300	10.48	19.65	75.93
400	14.06	26.22	100.31
500	17.90	32.79	126.46
600	21.13	39.27	151.39
700	24.67	45.98	176.09
800	27.53	48.24	200.42
900	29.96	52.34	233.23
1000	33.34	59.46	279.12

C.2. Timing Results for Query operations

(See subsections 5.3.4.1, 5.3.4.2, 5.3.4.3, and 5.3.4.4)

C.2.1. Berkeley DB XML

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	4.0	5.0	6.0	4.0	3.0	2.0	2.0	3.0	3.0	3.0	3.0	3.0	4.0	4.0	3.0
200	7.0	8.0	12.0	8.0	5.0	5.0	6.0	6.0	6.0	6.0	6.0	7.0	9.0	6.0	6.0
300	23.0	29.0	17.0	15.0	9.0	8.0	8.0	9.0	9.0	9.0	10.0	11.0	19.0	9.0	9.0
400	32.0	34.0	25.0	26.0	12.0	11.0	11.0	13.0	13.0	13.0	14.0	14.0	37.0	11.0	13.0
500	77.0	78.0	29.0	40.0	13.0	14.0	14.0	16.0	15.0	16.0	16.0	18.0	114.0	14.0	16.0
600					20.0	17.0	16.0	19.0	19.0	19.0	20.0	22.0		17.0	20.0
700					22.0	20.0	20.0	23.0	23.0	23.0	24.0	25.0		21.0	23.0
800					25.0	23.0	23.0	26.0	26.0	27.0	27.0	32.0		24.0	25.0
900					28.0	26.0	27.0	30.0	30.0	31.0	30.0	33.0		25.0	29.0
1000					31.0	29.0	29.0	32.0	33.0	33.0	33.0	38.0		30.0	32.0

C.2.2. dbXML

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	15	20	16	21	10	10	10	11	11	11	11	13	23	10	11
200	25	36	27	35	17	16	16	17	17	17	18	21	40	17	17
300	33	52	39	51	23	22	23	24	24	25	25	27	61	23	24
400	45	68	52	64	29	27	28	30	30	31	31	34	75	28	30
500	54	75	58	73	36	34	35	36	36	38	38	39	91	34	37
600	66	89	65	96	44	41	43	44	43	44	46	47	100	41	45
700	77	109	78	112	51	49	51	51	52	51	52	57	121	50	51
800	139	173	139	165	84	113	119	122	113	115	119	123	183	96	117
900	161	198	157	187	133	128	129	135	136	131	131	137	209	127	131
1000	175	217	179	213	143	151	144	146	151	148	146	152	233	143	145

C.2.3. eXist

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
700	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

C.2.4. Xindice

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
700	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

C.3. Timing Results for the Query re-run operations

(See subsection 5.3.4.4)

C.3.1. eXist

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	1.0	1.0	2.0	3.0	2.0	2.0	2.0	4.0	4.0	4.0	7.0	7.0	3.0	1.0	4.0
200	0.0	1.0	2.0	3.0	1.0	1.0	1.0	4.0	4.0	4.0	6.0	6.0	3.0	1.0	4.0
300	0.0	1.0	2.0	3.0	1.0	1.0	1.0	4.0	4.0	4.0	6.0	6.0	3.0	1.0	3.0
400	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	4.0	4.0	6.0	6.0	3.0	1.0	3.0
500	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	3.0	3.0	6.0	6.0	3.0	1.0	3.0
600	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	3.0	3.0	7.0	6.0	3.0	1.0	3.0
700	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	3.0	3.0	7.0	6.0	3.0	1.0	3.0
800	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	4.0	3.0	7.0	6.0	3.0	1.0	3.0
900	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	3.0	3.0	7.0	6.0	3.0	1.0	3.0
1000	0.0	1.0	2.0	3.0	1.0	1.0	1.0	3.0	3.0	3.0	7.0	6.0	3.0	1.0	3.0

C.3.2. Xindice

Size	Q1a	Q1b	Q1c	Q2a	Q3a	Q3b	Q3c	Q3d	Q3e	Q3f	Q4a	Q4b	Q4c	Q5a	Q5b
100	7.0	11.0	7.0	9.0	5.0	5.0	7.0	5.0	5.0	5.0	5.0	6.0	11.0	5.0	5.0
200	6.0	11.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
300	6.0	11.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
400	6.0	11.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
500	6.0	11.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
600	6.0	11.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
700	6.0	9.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
800	6.0	8.0	7.0	8.0	5.0	5.0	7.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
900	6.0	8.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0
1000	6.0	8.0	7.0	8.0	5.0	5.0	6.0	5.0	5.0	5.0	5.0	5.0	11.0	5.0	5.0

C.4. Timing Results for Modification operations

(See subsections 5.3.6.1, 5.3.6.2, and 5.3.6.3)

C.4.1. dbXML

Size	M(1a)	M(1b)	M(1c)	M(1d)	M(1e)	M(1f)
100	52.0	50.0	51.0	20.0	20.0	21.0
200	99.0	96.0	96.0	35.0	35.0	36.0
300	149.0	144.0	145.0	49.0	48.0	48.0
400	196.0	191.0	193.0	60.0	60.0	60.0
500	251.0	243.0	244.0	72.0	72.0	73.0
600	292.0	280.0	279.0	81.0	82.0	83.0
700	351.0	335.0	340.0	100.0	101.0	101.0
800	399.0	389.0	386.0	157.0	161.0	158.0
900	428.0	408.0	406.0	166.0	167.0	167.0
1000	485.0	469.0	471.0	189.0	190.0	198.0

C.4.2. eXist

Size	M(1a)	M(1b)	M(1c)	M(1d)	M(1e)	M(1f)
100	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0
700	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0

C.4.3. Xindice

Size	M(1a)	M(1b)	M(1c)	M(1d)	M(1e)	M(1f)
100	0.0	0.0	1.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	00
700	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0

C.5. Timing Results for Delete operations

(See subsections 5.3.8.1, 5.3.8.2, and 5.3.8.3)

C.5.1. dbXML

Size	D(1a)	D(1b)	D(1c)	D(2a)	D(2b)	D(2c)
100	21.0	20.0	19.0	20.0	21.0	11.0
200	35.0	34.0	33.0	35.0	36.0	18.0
300	51.0	47.0	46.0	47.0	50.0	23.0
400	62.0	58.0	57.0	60.0	61.0	28.0
500	75.0	70.0	69.0	72.0	72.0	35.0
600	84.0	79.0	78.0	82.0	83.0	43.0
700	102.0	93.0	94.0	99.0	97.0	52.0
800	168.0	129.0	156.0	161.0	154.0	111.0
900	164.0	162.0	166.0	167.0	165.0	121.0
1000	185.0	185.0	186.0	189.0	186.0	131.0

C.5.2. eXist

Size	D(1a)	D(1b)	D(1c)	D(2a)	D(2b)	D(2c)
100	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0
700	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0

C.5.3. Xindice

Size	D(1a)	D(1b)	D(1c)	D(2a)	D(2b)	D(2c)
100	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0
300	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0
700	0.0	0.0	0.0	0.0	0.0	0.0
800	0.0	0.0	0.0	0.0	0.0	0.0
900	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0

C.6. Timing Results for the Modification re-run

(See subsection 5.3.6.3)

C.6.1. eXist

Size	M(1a)	M(1b)	M(1c)	M(1d)	M(1e)	M(1f)
100	2.0	2.0	2.0	3.0	3.0	3.0
200	2.0	2.0	2.0	3.0	3.0	3.0
300	2.0	2.0	2.0	3.0	3.0	3.0
400	2.0	2.0	2.0	3.0	3.0	3.0
500	2.0	2.0	2.0	3.0	3.0	3.0
600	2.0	2.0	2.0	3.0	3.0	3.0
700	2.0	2.0	2.0	3.0	3.0	3.0
800	2.0	2.0	2.0	3.0	3.0	3.0
900	2.0	2.0	2.0	3.0	3.0	3.0
1000	2.0	2.0	2.0	3.0	3.0	3.0

C.6.2. Xindice

Size	M(1a)	M(1b)	M(1c)	M(1d)	M(1e)	M(1f)
100	8.0	8.0	12.0	6.0	6.0	6.0
200	8.0	8.0	11.0	5.0	5.0	5.0
300	8.0	8.0	11.0	5.0	5.0	5.0
400	8.0	8.0	11.0	5.0	5.0	5.0
500	8.0	8.0	11.0	5.0	5.0	5.0
600	8.0	8.0	11.0	5.0	5.0	5.0
700	8.0	8.0	11.0	5.0	5.0	5.0
800	8.0	8.0	11.0	5.0	5.0	5.0
900	8.0	8.0	11.0	5.0	5.0	5.0
1000	8.0	8.0	11.0	5.0	5.0	5.0

C.7. Timing Results for the Delete re-run

(See subsection 5.3.8.3)

C.7.1. eXist

Size	D(1a)	D(1b)	D(1c)	D(2a)	D(2b)	D(2c)
100	2.0	2.0	2.0	4.0	4.0	4.0
200	2.0	2.0	2.0	4.0	4.0	4.0
300	2.0	2.0	2.0	3.0	3.0	3.0
400	2.0	2.0	2.0	3.0	3.0	3.0
500	2.0	2.0	2.0	3.0	3.0	3.0
600	2.0	2.0	2.0	3.0	3.0	3.0
700	2.0	2.0	2.0	3.0	3.0	3.0
800	2.0	2.0	2.0	3.0	3.0	3.0
900	2.0	2.0	2.0	3.0	3.0	3.0
1000	2.0	2.0	2.0	3.0	3.0	3.0

C.7.2. Xindice

Size	D(1a)	D(1b)	D(1c)	D(2a)	D(2b)	D(2c)
100	5.0	6.0	7.0	6.0	6.0	6.0
200	5.0	5.0	7.0	5.0	5.0	5.0
300	5.0	5.0	7.0	5.0	5.0	5.0
400	5.0	5.0	7.0	5.0	5.0	5.0
500	5.0	5.0	7.0	5.0	5.0	5.0
600	5.0	5.0	7.0	5.0	5.0	5.0
700	5.0	5.0	7.0	5.0	5.0	5.0
800	5.0	5.0	7.0	5.0	5.0	5.0
900	5.0	5.0	7.0	5.0	5.0	5.0
1000	5.0	5.0	7.0	5.0	5.0	5.0