“STUDY OF OPTIMAL INDEXING SCHEMA FOR XML QUERY PROCESSING AND EFFECTIVE STORAGE MANAGEMENT-(2009-2013)”

A thesis submitted to

Tilak Maharashtra Vidyapeeth, Pune

For the Degree of Doctor of Philosophy (Ph. D.)

In Management Subject

Under the Board of Studies

Management

Submitted By

Mrs. Anagha Vijay Vaidya

Under the guidance of

Dr. Vilas Dattu Nandavadekar

April 2014

Declaration by the Candidate

I hereby declare that the thesis entitled "Study of Optimal Indexing Schema for XML Query Processing and Effective Storage Management-(2009-2013)" completed and written by me has not previously formed the basis for the award of any Degree or other similar title upon me this or any other University or examining body.

I future declare that the material obtained from other sources has been acknowledge in the thesis.

Mrs Anagha V Vaidya

(Research Student)

Place : Pune

Date : 25th April 2014

Certificate of the Guide

This is to certify that the thesis entitle "Study of Optimal Indexing Schema for XML Query Processing and Effective Storage Management-(20090-2013)" which is being submitted herewith for the award of the Degree of Philosophy (Ph.D) under the faculty of Management of Tilak Maharashtra Vidyapeet , Pune, is the result of original research work completed by Mrs Anagha Vijay Vaidya, under my supervision and guidance.

To the best of my knowledge and belief the work incorporated in this thesis has not formed the basis for the award of any Degree or similar title of this or any other University or examining body under her.

Dr Vilas Dattu Nandavadekar

(Research Guide)

Place: Pune

Date: 25th April 2014

Acknowledgement

I would like express my deepest gratitude to my guide and mentor Dr. Vilas Dattu Nandavadekar . His constant encouragement and resolute approach towards life in general and research in particular has inspired me in more than one ways.

I am also grateful to Dr. R. N. Dhamdhere for his timely guidance and valuable input. I would like to give a special thanks to Mrs Wate Madam for her encouragement and support in all my problems.

I would like to give thanks to Dr Sagar Jambhorkar whose wise advice and constructive criticism was the guiding factor behind this work. Sir without your support, I cannot complete my research work. Thank you, sir.

My heartfelt thanks go to my friends, Dr Manish Kumbhar , Mrs Vidya Gavekar, Mrs Tejaswini Apte whose support and 'push' kept me on track. Exchange of ideas and animated discussions on totally unrelated topics of our respective research fields helped me think out of the box, which is a key ingredient for any investigative endeavor.

I would also like to thank all of my friends and my students who have helped to make this research foray an enjoyable and learning experience possible in one way or the others. Thank you all.

I deeply grateful to my husband Mr. Vijay Vaidya for constantly edging me on to go on and sharing the household responsibilities with me to let me have time to focus on my work. My daughter, Ms Pranjal and Ms Sanjyot, who always were enthusiastic to know when and where is my paper being published and always giving me incentives to work. Thanks.

Last but definitely not the least, my mother and my father, whose blessings are always there whenever I take up something, had been instrumental in many ways. Thank you.

CONTENTS

Sr. No	Contents	Pg. No
I	List of Tables and Graphs	V
II	List of Figures	IX
III	List of Algorithms	X
IV	List of Psudocode	XI
V	List of Annexure	XII
VI	Abbreviations	XIII

Chapter -1: Introduction

1.1	Introduction		1
1.2	Challenges in XML		11
1.3	Objective of Research		14
1.4	Organization of thesis		14
		References	16-19

Chapter-2: Background and History

2.1	Introduction of Database and History of Database		20
2.2	XML History		24
2.3	XML Data Representation Components		25
2.4	XML- Query Languages		26
2.5	XML Query Processing Method		29
2.6	XML Query Processing Approaches		31
2.7	XML Indexes Approaches		34
		References	38—39

Chapter - 3 : Review of Literature

3.1	Introduction	42
3.2	XML – RDBMS Storage	44

3.3	Native Storage and Query Processing	48
3.4	Index Strategy used in XML	54
3.5	The Outcome of the literature review	66
3.6	Summary and Conclusion	70
	References	71..78

Chapter 4 : Research Methodology

4.1	Introduction	79
4.2	Importance of the Study	80
4.3	Scope of the study	81
4.4	Problem Statement of the Research	82
4.5	Objectives of the Research	83
4.6	Statement of Hypothesis	83
4.7	Research Methodology	87
4.7.1	Secondary Data	87
4.7.2	Primary Data	87
4.8	Case Study Method	99
4.9	Content Analysis	99
4.10	Summary and Conclusion	101
	References	102..103

Chapter 5: Data Presentation, Analysis And Interpretation

5.1	Introduction	104
5.2	Presentation and Analysis of Data-Part-I: XML utilization in Business Framework	107
5.3	Presentation and Analysis of Part II: XML storage and query execution policy.	125
5.4	Comparative study of three Commercial Products	130
5.5	The Experimental results on Query execution and storage strategy	132

5.5.1	Query Execution Time Measurement of IBM DB2.9 and SAXON	132
5.5.2	Comparative Study of Query Execution time SAXON and IBM DB2.9	155
5.5.3	Comparative Study of Disk Space SAXON and IBM DB2.9	160
5.5.4	Comparative Study of Disk traffic SAXON and IBM DB2.9	161
5.5.5	Query execution Plan for XML-RDBMS and XML-Native Storage	162
5.5.6	Index Creation Time measurement	168
5.6	General Observation and finding	169
5.7	Testing of Hypothesis	170
5.8	Conclusion and Summary	177
	references	178..180

Chapter 6: The Proposed Model : Path Encoding Structural Summary (PESS) Model

6.1	Introduction	181
6.2	Model Overview: Path Encoding and Structural Summary (PESS)	182
6.3	Architecture of Path Encoding and Structural Summary (PESS) Model	184
6.4	PESS Model : XML Data Organization Approach	186
6.5	PESS Model :Query Execution Approach	201
6.6	Results and Analysis of PESS Model	206
6.6.1	Measurement Parameter for experiment	206
6.6.2	Criteria 1 - Query Performance Measurement	206
6.6.3	Criteria-II Measurement Parameter for Node identifier	207
6.6.4	Result and Analysis of Criteria I: Query Performance Measurement	208
6.6.4.1	Result and Analysis of Existing method.- Reverse Path Approach	212
6.6.4.2	Result and Analysis of proposed model - PESS Model	219
6.6.4.3	Conclusion of Criteria 1 - : Query Performance Measurement	225

6.6.4.3.1	Result and Analysis of Query Performance Measurement	225
6.6.4.3.2	Result and Analysis of Disk Space of RM Approach storage and PESS Model	231
6.6.4.3.3	Result and Analysis of Index Structure	232
6.6.4.3.4	Result and Analysis of Disk Traffic	235
6.6.5	Result and Analysis of Criteria II: Measurement of new label insertion cost and Index creation cost.	237
6.7	Contribution of Researcher	240
	References	240-243

Chapter 7 : Conclusion and Future Work

7.1	Conclusion	244
7.2	PESS Model	249
7.3	Limitation of PESS model	251
7.4	Scope for future Research	252
7.5	Suggestion	255

Appendices

Annexure-I: Comparative Study of XML Storage	257
Annexure-I I : Case Study of PESS Model and Query Execution Plan	265
Annexure-III :Sample of XML files and their Node Structure used for Experiment	269
Annexure-I V: IBMDB2.9 and SAXON Introduction	278
Bibliography	283

List of Tables and Graphs

Sr No	Name of Table	Table	Page No	Graph	Page No
1	XML data Storage into RDBMs	1.1	6	-	-
2	DBMS Evaluation	2.1	23	-	-
3	XML Query Languages	2.2	29	-	-
4	Respondent Sample	4.1	88	-	-
5	Software List for Experiment	4.2	89	-	-
6	Input Data Files for Experiment	4.3	90	-	-
7	XML File Structure	4.4	91	-	-
8	Types of XML Query	4.5	92	-	-
9	Query Set for Experiment	4.6	93	-	-
10	XML Business framework example	4.7	100	-	-
11	Selection of Sample	5.1	104	-	-
12	Summary of XML Utilization in different Company	5.2	111	-	-
13	Data Analysis: XML Utilization in different Company	5.3	112	-	-
14	Distribution of sample of XML Utilization in Business Framework	5.4	113	5.1	113
15	XML can be used as: Database, Documentation and Content management System	5.5	115	5.2	115
16	XML cannot replace RDBMS/ HTML	5.6	116	5.3	-
17	Extension of XML language	5.7	119	-	-
18	XML Protocol for data transformation	5.8	123	-	-

19	XML and RDBM Comparison	5.9	126	-	-
20	XML-RDBMS Approach towards Query Processing	5.10	128	-	-
21	XML index Strategy	5.11	129	-	-
22	XML-RDBMS –index strategy	5.12	129	-	-
23	SAXON: Query Execution Timing for Data Set 1.1	5.13	134	5.4	134
24	IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 1.1	5.15	135	5.5	136
25	IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 1.1	5.16	137	5.6	137
26	SAXON: Query Execution Timing for Data Set 1.2	5.17	139	5.7	139
27	IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 1.2	5.18	140	5.8	140
28	IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 1.2	5.19	141	5.9	142
29	SAXON: Query Execution Timing for Data Set 2.1	5.20	143	5.10	144
30	IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 2.1	5.21	145	5.11	146
31	IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 2.1	5.22	146	5.12	147
32	SAXON: Query Execution Timing for Data Set 2.2	5.23	148	5.13	149
33	XML-Native Storage Query Execution Timing for Data Set 2.2	5.24	150	5.14	150
34	IBM 2.9 XML-RDBMs Query Execution Timing for Data Set 2.2	5.25	151	5.15	152
35	Query Execution Time Analysis	5.26	154	-	-
36	Comparative Query Execution Time for	5.27	155	5.16	155

	Data Se 1.1				
37	Comparative Study of Query Execution Time for Data Set 1.2	5.28	156	5.17	157
38	Comparative Study of Query Execution Time for Data Set 2.1	5.29	158	5.18	158
39	Comparative Query Exaction Graph of IBM DB2 for XML storage	-	-	5.19	159
40	Comparative study of SAXON and IBM Query Execution Time	-	-	5.20	160
41	Disk Space Analysis for - IBM DB2 and SAXON	5.30	160	-	-
42	Disk Traffic - IBM DB2	5.31	161	-	-
43	Query Execution Plan and Statistical Analysis of Query Execution Plan	5.20	162	-	-
44	Index Creation Time	-	-	5.21	168
45	Node Label Schema Comparison	6.1	188	-	-
46	Query Token	6.2	202	-	-
47	Index Strategy of RPM and PESS	6.3	211	-	-
48	Reverse Path Approach -Query execution Time for CD.XML	6.4	213	6.1	213
49	Reverse Path Approach -Query execution Time for WS.XML	6.5	215	6.2	215
50	Reverse Path Approach -Query execution Time -book.XML	6.6	216	6.3	217
51	Reverse Path Approach -Query execution Time –Chp .XML	6.7	218	6.4	219
52	PESS -Query execution Time -CD.XML	6.8	220	6.5	220
53	PESS -Query execution Time -WS.XML	6.9	221	6.6	222
54	PESS -Query execution Time -book.XML	6.10	223	6.7	224
55	PESS -Query execution Time -chp.XML	6.11	225	6.8	225
56	PESS Model - Comparative study of Axis	-	-	6.9	227

	Query				
57	Comparative Study of Query Execution Time (RM Approach storage Vs PESS Model)	-	-	6.10	229
58	Comparative Study of Disk Space (RM Approach storage Vs PESS Model)	-	-	6.11	231
59	PESS- Index Structure – Axis Query	-	-	6.12	232
60	PESS- Index Structure - FLOWR Query	-	-	6.13	233
61	PESS- Comparative Study of Query Execution Time (CD.XML Vs Book.XML)	-	-	6.14	234
62	PESS- Comparative Study of Query Execution Time (WS.XML Vs Chp.XML)	-	-	6.15	235
63	Comparative Study of Disk Traffic	-	-	6.16	235
64	Comparative Study of Disk Space	-	-	6.17	236
65	Input File for Node Updating	6.12	618	-	-
66	Time Measurement for Label Insertion (leaf node)	-	-	6.18	239
67	Time Measurement for Label Insertion (in between)	-	-	6.19	240

List of Table Figures

Sr No	Title	Figure No	Page No
1	XML Progression	2.1	27
2	XML File structure	2.2	28
3	XML Query Processing	2.3	31
4	XML Index and XML Storage Structure	2.4	39
6	PESS Mode Abstraction	6.1	183
7	PESS Model	6.2	184
8	XML File	6.3	190
9	XML Node_ids	6.4	190
10	XML Path Segregation	6.5	191
11	XML node Insertion Before	6.6	194
12	XML node Insertion at end	6.7	194
13	XML node Insertion in between	6.8	194
14	XML insertion at leaf node	6.9	194
15	XML Node-id	6.10	198

List of Algorithm

Sr No	Title	Algorithm No	Page No
1	Query Measurement Time	Algorithm 4.1	95
2	Assign label to XML file	Algorithm 6.1.	191
3	New node insertion into XML file	Algorithm6.2	193
4	Node - Labeling Schema for XML Node relationships	Algorithm 6.3	195
5	Creating Path_id for XML File	Algorithm 6.4	199
6	XML Query Execution (For Native Storage)	Algorithm 6.5	201
7	Query Execution for XML-RDBMS Storage	Algorithm 6.6	204

List of Psudocode

Sr No	Title	Psudocode	Page No
1	Query Execution Time Measurement	4.1	96
2	Query Execution Time Measurement on SAXON	4.2	98
3	Assign XML node label to XML file	6.1	192
4	New node Insert into XML File	6.2	194
5	Generating Path identifier for XML file	6.3	199

List of Annexure

Annexure No.	Title	Page No.
1	Comparative Study of XML Storage	257
2	Case Study of PESS Model and Query Execution Plan	265
3	Sample of XML files and their Node Structure used for Experiment	269
4	IBMDB2.9 and SAXON Introduction	278

Abbreviations

APEX	Adaptive XML Index
A-D	Ancestor-Descendent Relationship
DOM	Document Object Model
DTD	Documents Type Definition
FLOWR	For , Let, Order, Where , Return
GUAM	Generalized Update Access Method
HTML	Hyper Text MarkUp Language
IMS	Information Management System
IR	Information Retrial
IDS	Integrated Data Store
JETM	Java Execution Time Measurement Library
NXD	Native Database System
OODBMS	Object Oriented Database Management System
ORDBMS	Object Relational Database Management System
P-C	Parent –Child Relationship
PESS	Path Encoding Structural Summary
PSVI	Post-Schema Validation Infoset
RDBMS	Relational Database System
SGML	Standard Generalized Mark-up Language

W3C	World Wide Web Consortium
XScheme	XML Schema
XML	eXtensible Mark-up Language

Chapter 1

Introduction

1.1.1 Introduction

Twenty first century is known as an era of information. Most of the businesses are shifted on the internet. These business transactions create huge amount of data. Thus today, designing of data management system is a major challenge for computer system which can exchange the information between two different applications or two different transactions over internet. This operation is supported by an XML. An XML is emerging as a de facto standard for information exchange over internet. Initially, all these businesses were using RDBMS technology for their data management system. But on internet, two different database systems want to communicate to each other on different platforms. This requirement cannot be solved by RDBMS. It is rigid and not user friendly and cannot be used for information exchange. This difficulty is overcome by XML because it has high flexibility, platform independency, portability, simplicity and open standard. Since February 1998, the eXtensible Mark-up Language (XML [57] [45]) has become the standard medium for data representation and exchange over the web [45]. As a result, the size and the number of XML's users increased dramatically. XML not only used as database but it caters the need, of all community users. For example the database community used it as database, programmer community used it as a programming language; document community used it for information searching processes. XML utilization is listed below:-

1. Complex Structural Information Management: - The web-application wants to store the data of user's performance like home page, security setting, and language setting, display setting etc. This information is hierarchical and nested. It is difficult to store it on RDBMs technology. But in XML it is represented easily. XML not only stores this information but also helps to interpret the information between different applications on different system.

2. In RDBMS, the complex structure is mapped using large number of tables that don't correspond directly to the objects that are being modeled. For retrieving a record from these tables may require large number of join operations. These join operation increased query

processing cost and; cost of data storage. All these cost can be reduced by XML technology. A single XML node maps a complete structure. XML has property of recursion so the same information can easily represented by XML node structure

3. Interoperability: - E-business requires data from outside the organization, or from another department. Therefore different databases are used for catering the need of a single application. In many situations, department or organization does not allow the direct access to database but willing to provide limited forms of information through predefined interface. In such applications, user provides the input and this information is utilized by software application, instead of HTML format XML format is used for this purpose.

4. Many companies are moving to XML to write their internal documentation. The advantage of XML for documentation is that it can be used to define the common traits. The best thing about XML for documentation is that the XML is easy to understand for humans, both of the actual documentation, but also the XML code surrounding it. XML can be used for any type of documentation, from a publishing house to marketing materials.

5. XML's pervasiveness and universality allow individuals to combine information from disparate sources and enterprise applications in a process called information aggregation. Today most enterprise applications can import and export data in XML and several technologies have emerged that can process and transform XML without having to understand the content in detail.

6. XML is used for data transformation within business. One of the examples is EDI; Electronic data interchange community, which was used by the large company for selling and buying their product. This service was not affordable by the small company due to high paying cost and as large company have major stock holder of it hence they have their say for changing their standers which also affect small company business. These standards are replaced by XML. XML based e-commerce is penetrate to the small companies. Most of the people agree that XML is easy to work with more extensible.

All above mention application suggests the XML is used as international data standard, understanding business data, used for translating and moving data from one platform into another platform. As considering the importance of technology, the literature to bring this new

technology up to acceptable maturity levels is extremely large including proposals for; XML query languages (e.g. Xpath [58], XQuery [58] and XML-QL [45]), mapping XML to relational databases (examples are [47][6][25]), commercial XML-enabled databases (such as OpenXML package in SQL-Server [27] and XML data type in IBM-DB2 [26]), native XML databases (e.g. Natix [52], Timber [15], and DB4XML[50]), and XML data accessibility and security (e.g. Compressed Accessibility Model "CAM" [55], SQL-like XML security model [57]).

This research work focused on identification of best storage and indexing technique of XML hence researcher considered XML as database technology view. Once XML is considered as a database system, it must provide all database features for effective database management like transaction management, security, access control and query optimization. This research study is about the existing query optimization and indexing techniques and identifies their limitation, and design a new index structure for effective query execution. The study is more focused on XML index strategy and its impact on XML query evaluation. XML indexes are built of three different aspects: (1) index is built on the type of the document i.e. index on data centric file and index on document centric file. (2) Index residency –In XML two types of index are created, memory based index and disk based. (3) Indexing on the XML structural relationships.

1.1.2 Different Approaches of XML Storage and Query Processing Techniques:-

XML data is represented into three types of file: data-centric file, document-centric file and mix-mode file which is combination of data-centric and document- centric file. The stored information is retrieved by using different types of queries. Two types of queries are used on these file: IR queries and data centric queries. Information Retrial (IR) style of queries are used by document centric file. The database styles of queries are used by data centric file. Therefore, the different strategies are used for retrieving desired data e.g. for IR style of queries keyword search algorithm and ranking results are major issues while desire data retrieval with minimum query processing time is major challenge for database queries. For data centric query, different types of indexing techniques are applied on these file and have different query processing algorithm. The index strategies are classified as on residential type, structural relationships between different XML elements. The residential indexes are classified as main memory based index and disk-based index. The main memory based indexes are resided into computer's main

memory and disk- based index are reside on secondary memory. The main memory style of index defines how to store entire documents into main memory. These types of index have excellent response time for structural-join queries because they avoid expensive cost of I/O operations. Its implementation is straightforward and feasible for quarrying small XML document and inefficient for large XML document. Because loading entire XML file from secondary memory into main memory increased high I/O cost, the need for disk-based algorithms is required. Now a day memory is not a problem thus disk base indexes can use memory as per their requirement but reduce query processing time is a major challenge for these indexes. Alternatively, XML indexing algorithms can be evaluated on the basis of how the XML structural relationship is encoded in the index file [28] .These indexes map the document property like edge structure, values and their node relationships in the index using physical and logical address. These properties are mapped as per the query work load. Thus two parameters need to be considered while designing this index i.e. flexibility and selectivity. Flexibility means index structure such that it can handle any arbitrary query work load. And selectivity means instead of indexing entire document create index on document fragments which are frequently queried by the users or application. Like RDBMS technology XML also have secondary index for has reducing index size and accelerates query performance. The secondary indexes are well established in relational database but it is still not well developed in XML, because XML data is hierarchical and semi-structure data, creating index is a major challenge on it. In recent years, significant effort has been devoted for developing high-performance XML database system by designing different style of index on document property like value index on content node value, structural relationship on XML elements, by creating unique identification of each XML node and on different feature of XML document. For the study indexing techniques are classified into two types: (1) XML-RDBMS approach (2) Native Approach

In XML- RDBMs, XML data is mapped into relational table. Once the data is mapped into RDBMS format, it can take all the advantage of existing RDBMS features like query optimizer, query processor etc. Different approaches are proposed in literature for mapping XML data into relational table [14, 7, 13, 48, and 29]. The main challenge in RDBMS approach is converting XML file into relational tables. Because XML represents hierarchical and recursive data while RDBMS establish relationship between different tables. Therefore, it is difficult to represent document order of XML data into RDBMs. Since XML data is split into different tables it

required maximum query retrieval time. This approach is grouping into: edge approach, node encoding approach and path materialization approach. The edge approach [14] is first approach for XML-RDBMS storage. The document order semantic is addressed by node labeling [29] but still have high join operation in query processing, the path materialization tries to reduce the 'join' operation up to certain level [48]. All these approaches are discussed into chapter 3"literature review".

The second category of the XML storage and indexing is 'Native Approach'. This approach uses the document's structure-encoding methodology. In 'Native Approach', data retrieval is faster than relational database because entire document is stored together physically or use physical pointer between the parts of document. This allows document to be retrieved either without joins or with physical joins, both of which are faster than the logical joins used by relational database. This technique performs full text search and proximity search more effectively then RDBMs techniques [24]. The native index grouped into :path-encoding and join approach[43,30,51,60] sequence-based approach[38,16,36] and feature base approach [35,22,21,23], Navigational approach [59, 42, 3, 32]. For each group have their own advantages and disadvantages. The detail storage are discuss below :-

1.1.2.1 RDBMS Approach for XML storage and Query Processing :-

The XML file is a simple text file. Storing the data into a text file has certain disadvantages like concurrency, atomicity, data access security etc. A text file doesn't have a good support for query optimization [20].

Before 1990, all business applications (before e-business) were using RDBMS technology for storing and querying data in their business. These applications have difficulty in integrating the existing data with new technology. In order to work in closer cooperation with such traditional data, it is useful to store XML data in the same type of databases. Another advantage of RDBMS system is it has matured database organization from a quarter of a century, and have reached full growth.

XML is a hierarchical, recursive and ordered database. Hence, it is difficult to store the information into RDBMS format. Prof A Silberschatz et [1] suggests five approaches for storing XML data into relational system. They are describes in Table 1.1:-

RDBMS Storage Type	Characteristics	Disadvantage
Store as String	• XML data is stored in CLOB and BLOB values in tuples in relational storage. • This is simple and easy representation. The database system doesn't require knowing the database schema.	• Cannot query the data directly • The large part of XML document is stored as a string
Tree Representation	• XML data can be modeled as a tree and store using relational is represented using relational schema : Node(id, parent_id, type, label, value) Where ○ Id - node identifier , ○ parent_id represent parent node, ○ type of node, ○ label is name of the element or attribute, ○ Value is text value of the element or attribute.	• This method has drawback: - element gets broken into many pieces and a large number of joins are required to reasonable sub-elements into an element.
Map to Relations	This type of storage is preferred when XML schema and relation is known in advance i.e. This approach is convenient for mapping XML files whose DTD and schema is known in advance	• Major drawback: - the XML document order must be maintained. Set of values must be represented and easily convert from XML to RDBM and RDBMS to XML.
Publishing and Shredding XML Data	• A simple relation for XML mapping is an XML element for every row of a table, and makes each column in that row a sub element of the XML element. • XML view of relational data can be treated as virtual XML document, and XML queries can be executed against the virtual XML document.	• File size is huge. • It is difficult to create an index on such document. • Cannot perform data-style of queries on these database
Native Storage within a RDBMS	• A Specialized data type ".xml" is used for storing a complete XML file i.e. native storage. • Each document is stored as a value of type "xml" in a separate tuple. • Special purpose indices are created to index XML data. • XQuery queries are embedded with SQL queries	• The large XML file cannot be stored into a single row. Thus it should be shredded into different rows; this type of storage is called as medium grain storage. When the data is shredded for storing medium gain; it is difficult to transfer it back into the XML file.

Table 1.1: XML data Storage into RDBMs

From the above table the major challenge with mapping XML data into tree structure format because mapping P-C and A-D relation is the major challenge. In literature, XML-RDBMS are classified into three types: Edge approach, Node approach and Path encoding approach. In Edge approach, [14] XML data is converted into edge label data model and map it into table. This approach increases number of join operation in query processing and cannot process 'twig query' effectively. Another disadvantage of this approach is some XML file information may be lost while shredding XML file into RDBMs. The containment relationship is maintained as well as original XML file can again reconstructed from the node approach. But this approach cannot solve partial matching queries and difficult in performing equijoin operation on the table. Query computation cost is very high and has high updating cost.

In path materialization approach, unique identity is created for all possible path and store XML files in RDBMS format. This approach is effective but the numbers of tables are increased for storing XML file. As numbers of tables are increased disk I/O cost is also increased. In query execution, number of join operations is also increased.

1.1.2.2 Native Approach

Native XML databases are systems developed purely for storing XML documents. Their data model is flexible, so that documents do not need to be transformed to equivalent tables and columns. They support the complexity of XQuery more efficiently than relational database.

Native Approach of XML database model integrates specific techniques for storing the data, and accessing XML documents. These techniques include new storage methods, indexing structure, and join algorithm for query processing.

The native database data is stored on container. Unlike relational database, no tables have to be set up, and no complicated designing have to be made. Each container can have sub-containers. The XML data is stored into this container and maintain some relationship between them. The difference with relational database is that the structure of the XML data in a container does not have to be fixed. From the literature review, native XML database approach has three ways to represent it: join approach, sequence based approach, navigation approach.

In Join approach, XML file is represented into inverted list format. The inverted list each XML tag is store with (start no: end no, level) information. Different join operations like MPMGJN [23], Stack Tree approach [46], Holistic Approach [31], Index-Graph-Aided approach [41] are

designed. The major drawback of this approach is it increased intermediate result while executing XML query.

Sequence based approach [37] [17] [36] the index are created on a specified sequence of XML file. It doesn't have supports to ordered twig queries. For answering a general twig query, Q has to transfer into simple queries first then evaluate it which results in high query costs. It involves a large number of index probes which increased high disk I/O costs. It repeatedly visits many data nodes unnecessarily.

Navigation approach [4, 33, 61, and 34] XML data arrives in the form of data streams, without any associated indexes, and the entire XML data tree has to be traversed node by node in the depth-first order, which preserves the original XML document order. The run-time stack data structure is used for query evaluation. XML file is read character by character and its tokens are stored into a stack. When an opening tag is encountered, the stack-top element is transformed to new state. These newly generated states are collected into a new stack. After calculation on these states, they are then pushed into the stack. When a closing tag is encountered, the stack-top element is simply popped out of the stack.

1.1.3 XML Indexed Structure :-

Indexes prebuilt on XML data can facilitate XML queries processing by locating data quickly. Index is built on traversing paths with the element set or on the value of elements and attributes. Broadly, XML indexes are classified into two types: 'Value Indexes' and 'Structural Indexes'. Value indexes are built on XML data values i.e. from value node. The structural indexes are built on structure of XML documents. The 'Value Indexes' answer content based query effectively while 'Structural Indexes' answer all types queries i.e. Contentment relationship, Order query etc. The XML indexes are discussed below:-

(1) **Structural Summary**: - A structural summary is smaller version of an XML tree where all paths from root node to any leaf node in the actual XML tree are preserved. The main goal of a structural summary is to eliminate redundant structural information without losing structural constraints; that is the structural relationship between XML element such as P-C and A-D relationships. Data Guide, forward and backward similarities, inverted list etc data structures are used for creating the structural summary information.

Data Guide, Strong Data Guide [39], 1-Index and 2-Index [53] are research proposal. The drawback of these proposals is it has very high index size. It grows exponentially with XML file, sometime it is equal to the XML file itself. These proposals can solve total matching queries effectively but cannot solve partial matching and twig queries.

(2) **Selectivity Indexes**: -These index techniques are implemented from the RDBMS literature. In RDBMS literature, the indexes are created on certain fields which are selected by Database Administrator (DBA). The field of indexing is selected on different criteria like on frequently triggered XML queries, the fields which satisfy certain SQL queries. This concept was implemented by different XML index proposal like T-index index [54], index fabric [10]. In these proposals frequently queries templates are created and these templates are index for querying XML data. The major drawback of this index is index size. Another problem is query rewriting problem i.e. XML query need to be rewrite into SQL query.

The index size is reduced by new proposal i.e. 'Adaptive XML indexes'. The Adaptive XML Index (APEX) does not keep all paths starting from the root node. It utilizes frequently used paths to improve the query performance. APEX updates index structure incrementally according to changes of query workload.

The selectivity indexes have listed three drawbacks [12]: (1) the index will involve expensive computation complexity to alter the index structure during the query-evaluation processes (2) More storage space is required to keep the queries and changes in the document's structure (3) This technique is only concerned with frequently triggered queries and ignores other queries which might involve heavy comparison operations. This type of index is inefficient for solving partial and twig query.

(3) **Structural Join Indexes**: - The basic framework of any structural index is to encode some structural relationships between XML nodes (elements attribute etc) so that query processor is able to predict results by simply approaching a corresponding index without accessing the actual data file. The parent-child and the ancestor-descendent relationships are sufficient to answer most classes of the structural queries [13][19][54][44] The structural-based indexes are grouped into three categories: (a) node-encoding approaches, (b) path-encoding approaches and (3) sequence based approaches

A. **Node Index approach: -** The main goal of the node-labeling approach is: a) assigning a unique code for each node in the XML tree. b) Preserving the nested hierarchical relationship

(structural) during XML updates, c) minimizing the re-labelling cost (including the processing time and I/O accessibility) in the case of data updates, and d) reducing the storage space for store generated code [11]. The Label of the node suggests the position of the node in tree, and relationship with other nodes. This technique is classified into two types: prefix-based approach and regional-based approach.

The main problem of this approach is updating cost. Index updating cost is very high because when new node is inserted or deleted from the tree, the entire tree needs to re-label. Another problem is 'Label length'. As XML tree grows depth wise, the node-label length is increased and after 5th level it is impossible to read. [3]. Most of the research tried to reduce re-labelling. But this problem is still open. These new techniques fail to support specific requirement of processing hierarchical, semi-structured data. Node Label indexes can process 'Total Match Query' but they are insufficient for processing 'Partial Matching' and 'Content-based' query.

B. **Path Encoding Approach:** - This type of index has idea of creating a path summary of XML data to speed up the processes of query evaluation. This index stores all path information starting from root node to any arbitrary XML node and indexes are created on these paths. This technique is implemented in RDBMS storage [64, 32, 72, 62, and 63] and native storage [40, 38, and 12]. RDBMS storage has following advantage (1) speed up XML queries over a relational database engine (2) reduced storage space (3) avoid expensive structural joins that are required to evaluate branching path queries. This technique has several drawbacks such as recursive self-joins increase query processing cost and index updating cost is high. As XML data file increases, database updating cost is high.

In native XML database path indexing is implemented by using 'bi-similarity' graph data structure. This reduced the query performance in terms of speed and efficiency. But index size is very large. This index size is adjusted by implementing hashing techniques and by calculating query workload i.e. indexes are created only of those paths which are frequently used in query.

C. **Sequence based Approach**:- Sequence based indexes convert both the XML query and XML documents into sequences and use the well-established sequences matching techniques to obtain query answers (e.g. [37] [17][36]). The good feature of sequence-based indexing techniques is that they use the entire query tree as one unit, thus the expensive join operations are avoided during the query evaluation. In this approach, XML documents and branching

queries are represented as sequence, and subsequent matching provides the query answers. These approaches don't support queries with selection condition over nodes. The Sequence approach has the following limitations: 1) it supports only the ordered twig queries. 2) It may involve a large number of index investigation and index-probe operations might result in high random disk I/O costs. 3) It might repeatedly visit many data nodes unnecessarily.

1.2 Challenges in XML :-

1.2.1 Challenges : XML Index Structure:-

The XML data is stored into different data model like relational, object oriented, object relational data model, navigational, native data model. These data model has different indexing techniques for effective organization and data retrieval. With reference to these indexing techniques, following issues are noted:-

1. A proper designing of Index structure is necessary which reflects the structural and content oriented property of XML data file.
2. XML data doesn't have fix schema. Therefore, mapping XML elements into different tables is a difficult task. .
3. Once the data is mapped then designing of index structure is difficult because index structure must represent the structural property so that desired XML nodes can easily return by it.
4. From the literature review, some indexing technique creates index on data fragment instead of on a complete path of XML tree. This type of index accelerates XML tree traversal and they are small in size. So these indexes perform query execution fast. The major challenge of this type of index is how to define and select indices that refer to arbitrary document fragment. And another challenge is index updating .When an XML file is updated the index document fragment gets updated so the desired index structure is changed.
5. Index Size: - From the literature, it is noted that XML index size is large; it grows exponentially as XML data file grows. Therefore, algorithm needs to be developed for preserving index size.
6. The different types of XML index have observed the following points :-
    - Memory based index always processes a query fast. When the data is stored into main memory, its size must be small. Designing such small size index is challenge.

- Node index is classified into two labeling schema: Interval Labeling and Prefix Labeling Schema. The size of Interval Indexing Schema grows constantly as depth of the tree increases, while it grows exponentially in the Prefix Labeling Schema. Higher the size of its index, longer query processing time.
- In structural summary index, index size is exponentially increased with XML data file.

7. In Structural Summary index, the XML file is summarized into a specialized data structure. This structure serves as index for query evaluation but sometime this structure is not efficient for answering queries, the query has to travel a complete XML tree. [39]
8. Lack of Scalability: - All the existing XML structural indexes are designed for answering one query at a time but they lack the scalability for answering multiple queries simultaneously. Considering the client-server environment, where the XML document and index reside on server side, the queries come from different clients. These queries are executed one by one with existing structural indexes, but the common parts among the queries are executed repeatedly. This increases query processing cost for server side and deteriorate the query performance [9].

1.2.2 Challenges : XML Query execution:-

XML queries are broadly classified into two types: IR-Style (Keyword Search) queries and Database Style of Queries. IR-Style of queries retrieved information on keyword matching techniques. The database style of queries returns the matching database matching records. These queries are similar to traditional SQL queries. From literature, researcher have notice the following problems in query execution. They are:-

1. The XML data is stored into different RDBMS tables. The desired query data is retrieved by joining all these different tables. Thus the "join" operation is very high in query retrieval.
2. The XML- RDBMS data supports inner join and outer join operation but they are not supporting "equijoin" operation.
3. XML file is stored in RDBMS format and index is on Path value, then it performs heavy self-join operation for effective answering XML queries.
4. If XML file is stored in Native database format, intermediate query execution result is very high in query execution. This intermediate result cost must be reduced while executing XML query which may reduced query processing time.

5. Redundant Traversal: - The query process with existing structural indexes includes redundant traversal. Not all the traversal operations lead to results. That is to say, many early matched nodes may not lead to the final output nodes for each query, and this makes the early matched nodes as redundant traversal. In the scenario where users do not know the structure of the XML document clearly, many queries can be executed on existing structural indexes without result, and these queries impose many redundant operations in query processing, however the data structures of existing structural indexes cannot avoid such redundant traversal [63].
6. XML documents filtering: - XML data is store into 'Main in-memory style' implementation, i.e. XML document is first loaded into main memory in the form of a tree and then query is executed on this tree. This style is feasible for querying small documents but it is inefficient for querying large documents, because loading the entire XML documents from secondary memory into main memory increases high disk I/O cost which affects the query performance.
7. If more than two predicates are involved in the query, the query processing cost is very high because it requires traversing XML tree for every predicate.
8. Query processing cost is very high in RDBMS storage because XML queries are first converted into SQL queries and then they are evaluated by RDBMS query optimizer which increased query execution cost as well as query processing time.
9. To retrieve the data from the XML tree is a challenging task. It retrieves either a single node (node-path query), or a small tree (twig query) from a complete XML file. Existing Indexing techniques work very well for retrieving a single node from XML file. But effective retrieval of a small tree (twig query) is still upcoming area of the research[8]
10. Some queries are executed on keyword searching pattern. Implementation of the traditional keyword searching algorithm is not possible in XML because XML data is a hierarchical data model. So instead of returning a node it returns a complete document; additional 'Documents Fragment '[5] algorithms are required for getting desired result. Hence query execution cost and processing time is increased.
11. Another challenge for keyword search queries is ranking the returning result. In general, ranking result is based on keyword proximity and weighted term frequencies algorithm [8]. But in XML these parameters fail because the data is in nested format.

This research concentrates on database style of queries on data centric application only. Therefore researcher studied XML database index structure in depth. From last decade, this technology is under development and it is universally accepted by commercial application as well as in scientific application.

1.3 **Objective Of the research** :-

The primary focus of this thesis is to provide flexible, storage structural for XML data storage and index on XML database.

- The primary focus of this thesis is to provide flexible, storage structural for XML data storage and index on XML database.
- Develop an index structure which will encode structural XML information and reduced the redundant traversal of the XML tree.
- Develop a framework for XML index structure.

1.4 Organization of Thesis

The thesis is organized as Chapter 2 "Background and History ", this chapter explain the initial information of XML and evolution of XML. The chapter briefly explains the XML data storage and XML query processing information and different approaches towards the query processing. Chapter3, "Review of literature" studies the literature study on XML index and XML storage strategy. Researcher performed the literature review from 1998 till today. The study is regarding the XML storage and their indexing strategy. The chapter list out the advantage and the limitation of the different proposals and concluded the chapter with open challenges of the study. In Chapter 4, "Research Methodology" explains, the research methodology followed in this thesis and pilot study. The chapter 5 is "Observation and Finding ". The experiment is performed on different type of data file. The different queries were used for retrieving the desired result. The finding and observations are explained in it. The chapter 6 "Proposed Model: Path Encoding Structural Summary (PESS) Model", this chapter explain the proposed frame work for new model. It explains the complete architecture of the model and different algorithm for model construction and query execution. The proposed model was tested with existing model i.e.

"Reversed Path Approach". The different laboratory experiment was performed to check the efficiency of proposed model. The results and analysis also explained in the same chapter. The Chapter 7 is conclusion and future work. This chapter concludes research work and explains the future enhancement on proposed model and future research work.

References :-

1 A Silberschatz, H F Korth, S Sudarshan Database Sysem Concept, Sixth Edition International Edition 2011, ISBM 978-007-128959-7

2 A. Gabillon. An Authorization Model for XML Databases. In Proceeding of the 2004 workshop n secure web service, Fairfax, Virginia, USA, pages 16-18, 2004

3 Alan Halverson, Josef Burger, Leonidas Galanis, Ameet Kini, Rajasekar Krishnamurthy,Ajith Nagaraja Rao, Feng Tian, Stratis D. Viglas, Yuan Wang, Jeffrey F. Naughton, David J. DeWitt Mixed Mode XML Query Processing Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

4 Alan Halverson, Josef Burger, Leonidas Galanis, Ameet Kini, Rajasekar Krishnamurthy,Ajith Nagaraja Rao, Feng Tian, Stratis D. Viglas, Yuan Wang, Jeffrey F. Naughton, David J. DeWitt Mixed Mode XML Query Processing Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

5 B. Sigurbjörnsson, J. Kamps, and M. Rijke. Processing Content-Oriented XPath Queries. In the Proceedings of the thirteenth ACM international conference on Information and knowledge management, Washington D.C., USA, pages 371-380, 2004

6 B. Sigurbjörnsson, J. Kamps, and M. Rijke. Processing Content-Oriented XPath Queries. Inthe Proceedings of the thirteenth ACM international conference on management, Washington D.C., USA, pages 371-380, 2004Information and knowledge

7 B.M. Monjurul Alom, Frans Henskens, Michael Hannaford DRXQP: A Dynamic Relational XML Query Processor ; 2010 Seventh International Conference on Information Technology 978-0-7695-3984-3/10 $26.00 © 2010 IEEE

8 Beyer , K et al : System RX:One part Relational , One Part XML SIGMOD 2005 , June 14 -16 2005b, Baltimore , Maryland , USA Copyright 2005 ACM 1-59593-060-4/05/06

9 Bo Zhang , Zhihua Geng , Aoying Zhou SIMP: Efficient XML Structural Index for Multiple Query Processing The Ninth International Conference on Web-Age Information Management 978-0-7695-3185-4/08 $25.00 © 2008 IEEE DOI 10.1109/WAIM.2008.25

10 Brian F. Cooper ,Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, Moshe Shadmon A Fast Index for Semistructured Data Proceedings of the 27th VLDB conference, Roma, Italy, 2001

11 C. Chung, J. Min, and K. Shim. APEX: An Adaptive Path Index for XML Data. In proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pages 121-132, 2002.

12 Chung, J. Min, and K. Shim. APEX: An Adaptive Path Index for XML Data. In proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pages 121-132, 2002.

13 Chun Zhang, Je_rey Naughton, David DeWitt, Qiong Luo , Guy Lohman On Supporting Containment Queries in Relational Database Management Systems , SIGMOD ’01 Santa Barbara, California USA Copyright 2001 ACM 089791886/ 97/05 pp 425-3

14 E. Florescu and D. Kossmann, “Storing and Querying XML Data Using an RDMBS,” IEEE Data Eng. Bull., vol. 22, pp. 27-34, 1999.

15 H. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A Native XML Database. The VLDB Journal- Volume 11, pages 274-291, 2002

16 H. Wang and X. Meng, “On the Sequencing of Tree Structures for XML Indexing,” Proc. 21st IEEE Int’l Conf. Data Eng. (ICDE ’05), 2005

17 H. Wang, S. Park, W. Fan, and P.S. Yu, “ViST: A Dynamic Index Method for Querying XML Data by Tree Structures,” Proc. 29th ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’03), 2003

18 Haifeng Jiangy Hongjun Luy Wei Wangy Jerey Xu Yuz Path Materialization Revisited: An Efficient Storage Model for XML Data (AICE2000), Canberra. Conferences in Research and Practice in Information Technology, Vol. 1

19 J. Shanmugasundaram,H. Gang, K. Tufte, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and opportunities. In proceedings of the 25th international conference on Very Large Data Bases, Edinburgh, Scotland, pages 302-314, 1999

20 J.-K. Min, C.-H. Lee, and C.-W. Chung, “XTRON: An XML data management system using relational database,” Information and Software Technology, vol. 50, pp. 462-479, 2008.

21 Jong P. Yoon ,Vijay Raghavan,Venu Chakilam "Bitmap Indexing-based Clustering and Retrieval of XML Documents " 2001

22 Jong P. Yoon, Sung H. Kim, Go U. Kim, and Venu Chakilam "Bitmap-Based Indexing for Multi-dimensional Multimedia XML Documents " E.-P. Lim et al. (Eds.): ADL 002, LNCS 2555, pp. 165-176, 2002. © Springer-Verlag Berlin Heidelberg 2002

23 Jong P. Yoon, Vijay Raghavan, Venu Chakilam, Larry Kerschberg BitCube: A Three-Dimensional Bitmap Indexing for XML Documents Jong Yoon, et. al., BitCube, Journal of Intelligent Information Systems, Vol. 17, November 2001

24 Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child

edges: a look-ahead approach. In CIKM, pages 533–542, 2004.

25 M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi. Efficient schema-based XML-to-Relational data mapping. Avaliable online at www.sciencedirect.com since 15/12/2006]20/1/2006,

26 M. Nicola, and B. Linden. Native XML Support in DB2 Universal Database. In proceedings of the 31st international conference on VLDB, Trondheim, Norway, pages 1164-1174, 2005.

27 M. Rys. XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005. In Proceeding of the 2005 ACM SIGMOD International conference Baltimore, Maryland, pages 958-962, 2005on Management of Data,

28 Mohammed Al-Badawi, Dr. Siobhán North, Dr. Barry Eaglestone , research memorandum Indexing XML Databases: Classifications, Problems Identification and a New Approach, 15th November, 2007 in The University of Sheffield Department of Computer Science

29 Mustafa Atay, Yezhou Sun, Dapeng Liu, Shiyong Lu, Farshad Fotouhi MAPPING XML DATA TO RELATIONAL DATA: A DOM-BASED APPROACH arXiv:1010.1746v1 8 Oct 2010

30 N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," Proc. 21st ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '02), 2002

31 N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," Proc. 21st ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '02), 2002 Conf. Management of Data (SIGMOD '02), 2002.

32 N. Zhang, V. Kacholia, and M.T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML,"Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004.

33 N. Zhang, V. Kacholia, and M.T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML,"Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004.

34 Nicolas Bruno, Luis Gravano,Nick Koudas, Divesh Srivastava Navigation- vs. Index-Based XML Multi-Query Processing Proceedings of the 19th International Conference on Data Engineering (ICDE'03) 1063-6382/03 $ 17.00 © 2003 IEEE

35 Ning Zhang M. Tamer O¨ zsu Ihab F. Ilyas Ashraf Aboulnaga FIX: Featurebased

Indexing Technique for XML Documents VLDB '06, September 1215, 2006, Seoul, korea. ACM 1595933859/ 06/09.

36 P. Rao and B. Moon, "PRIX: Indexing and Querying XML Using Prufer Sequences," Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004

37 P. Rao, and B. Moon. Sequencing XML Data and Query Twigs for Fast Pattern Matching. The ACM transactions on database system, Volume 31, Issue 1, pages 299-345, 2006.

38 Q. Chen, A. Lim, and K. Ong. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data. In proceedings of the 2003 ACM SIGMOD international conference on Management of data, CA, USA, pages 134-144, 2003.

39 R. Goldman, and J. Widom, "Data Guides : Enabling Query Formulation and Optimization in Semistructured Databases", Proceedings of the VLDB, 1997, pp. 436-445.

40 R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In proceedings of the 18th international conference on Data Engineering, Washington DC, USA, pages 129-141, 2002.

41 R. Kaushik, R. Krishnamurthy, J.F. Naughton, and R. Ramakrishnan, "On the Integration of Structure Indexes and Inverted Lists," Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'04), 2004.

42 Ryan H. Choi, Raymond K. Wong Efficient XQuery Join Processing in Publish/Subscribe Systems, Copyright c 2009, Australian Computer Society, Inc. This paper appeared at the 20th Australasian Database Conference (ADC 2009), Wellington, New Zealand, January 2009.

43 S. A1-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of the IEEE International Conference on Data Engineering, 2002.

44 S. Abiteboul et al. The Lorel Query Language for Semistructured Data. In International Journal on Digital Libraries, 1(1):68-88. 1997

45 S. Abiteboul, P. Buneman, and D. Suci. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, ISBN: 1-55860-622-X, 2000

46 S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Y. Wu, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02), 2002

47 S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In Proceedings of the 6th annual ACM international information and data management, Washington DC, USA, pages 31-38, 2004workshop on Web

48 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, “Indexing XML Data Stored in a Relational Database,” Proc. 30th Int’l Conf. Very Large Data Bases (VLDB ’04), 2004.

49 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, “Indexing XML Data Stored in a Relational Database,” Proc. 30th Int’l Conf. Very Large Data Bases (VLDB ’04), 2004.

50 S. Sipan, K. Verma, J. Miller, and B. Aleman-Meza. Designing a high-performance database engine for the ‘Db4XML’ native XML database system. The Journal of Software-69, pages 87-104, 2004Systems and

51 T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In SIGMOD, pages 455–466, 2005.

52 T. Fiebig, S. Helmer, C. Kanne, G. Moerkotte, J. Neumann, and R. Weld. Anatomy of a native XML base management system. VLDB Journal- Volume 11, Number 4, pages 292-314, 2002

53 T. Milo, and D. Suciu, “Index Structures for Path Expression”, Proceedings of the ICDT, 1999, pp. 277-295.

54 T. Milo, and D. Suciu. Index Structures for Path Expressions. In proceedings of the 7th international conference on Data Technology, Volume 1540/1998, Jerusalem, pages 277-295, 1999

55 T. Yu, D. Srivastava, L. Lakshmanan, and H. Jagadish. A Compressed Accessibility Map for XML. ACM Transactions on Database Systems (TODS) Volume 29 – Issue 2, pages 363-402, 2004

56 W. Lian, N. Mamoulist, W-l. David Cheung, and S.M. Yiu, “Indexing Useful Structural Patterns for XML Query Processing”, IEEE Transactions on Knowledge and Data Engineering, 17(7), 2005, pp. 997-1009

57 W3C Website. Extensible Markup Language (XML) 1.0 (Fourth Edition). Online:http://www.w3.org/TR/REC-xml/, Accessed on: 30/10/2006

58 W3C Website. XQuery 1.0: An XML Query Language. Online: http://www.w3.org/TR/xquery/,Access on: 30/01/2007

59 X. Yan, P. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In proceedings of the 2005 ACM SIGMOD international conference on Management of data,

Baltimore, Maryland, pages 766-777, 2005

60 Xiaojie Yuan Xin Wang Chenying Wang Efficient XPath Evaluation Using a Structural Summary Index, 978-0-7695-3336-0/08 © 2008 IEEE DOI 10.1109/CSSE.2008.1058

61 Yanlei Diao, Peter Fischer, Michael J. Franklin, Raymond YFilter: Efficient and Scalable Filtering of XML Documents Proceedings of the 18th International Conference on Data Engineering (ICDE'02) 1063-6382/02 $17.00 © 2002 IEEE

62 Yi Chen , Susan B. Davidson, Yifeng Zheng "A bi-labeling based XPath processing system" Information Systems 35, 2010, doi: 10.1016/j.is.2009.05. ACM

63 Yi Chen, Susan B. Davidson, Yifeng Zheng BLAS : An Efficient XPath Processing System published in Proceedings of the 2004 ACM SIGMOD International conference on Management of Data 2004, pages 47-58. Publisher URL: http://doi.acm.org/10.1145/1007568.1007577

Chapter 2
Background and History

2.1 Introduction of Database and History of Database

Database technology is having a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, low, education, and library science. A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. Database systems are designed to manage large information. Management of data involves both defining structure for storage of information and providing mechanism for manipulation of information.

The origins of the database go back in 1960, for 'Apollo' moon-landing project [4]. At that time, there was no system available that would manage the vast amount of information that the project would generate. The first database product GUAM (Generalized Update Access Method) was developed on the concept of hierarchical tree which is nothing but the first data model i.e. hierarchical database management system. In the mid 1960, IBM developed first commercial DBMS is known as IMS (Information Management System). It is still main hierarchical DBMS used by most large mainframe installations. In the mid-1960s, another significant development was IDS (Integrated Data Store) from General Electronic [3]. The project developed new type of database system known as network DBMS which represents more complex relationships. In 1970, E.F. Codd developed first relational data model which is considered as second generation of DBMS [2]. In response to the increasing complexity of database applications, two new systems have emerged: the Object Oriented DBMS (OODBMS) and the Object-Relational DBMS (ORDBMS). This evolution represents third-generation DBMSs. In 1990, onwards emerging of web-based application in computer, E-commerce etc hence the fourth generation DBMS is used. The data is represented by semi-structure data model i.e. by XML. Today to mange search engine data and Face book data, the new term "big data" is used. DBMS evolutions are listed in table 2.1 below:-

Table 2.1 DBMS Evaluation

Evolution of DBMS		
Years	**Database characteristics**	**Commercial Product available**
1960	Navigational Database System	**SABRE** that was used by IBM to help American Airline manage its reservation
	User need to know the physical structure of the database	
	Network model and Hierarchical model	
	The first Data Systems language (CODASYL) was developed	Information Management System the First DBMS product developed by IBM
	DBTG (Data Base Task Group) was formed in 1967.	IDS (Integrated Data Store) the first Network model developed by General Electronics.
1970	E F code introduced new relational model	**INGRES** was developed at the University of California-Berkeley
	New QUEL - query language was introduced	
	A new database model called Entity-Relationship, or ER, was proposed by P. Chen in 1976	
	The development of structural query language called SQL, which has since become standard query language for relational DBMS	**System R** was developed at IBM in San Jose and led **IBM's SQL/DS & DB2**, **ORACLE, HP's Allbase, and Tandem's Non-Stop SQL. DB2 became one of the first DBMS** (Database Management System) product based on the relational model (Oppel, 2011).
	Various commercial products DB2 and SQL/DS, Oracle were developed.	
1980	Network and hierarchical models lost their charm with no further development of these systems, but some legacy systems are still in use today	RIM, RBASE 5000, PARADOX, OS/2 Database Manager, Dbase III, IV (later FoxBASE, and Visual FoxPRO), and Watcom SQL. All these systems were introduced during the1980's and were based on the relational model.
	OODBMS System was developed	

Early 1990	**Client Server Model** development	World Wide Web , ASP, Front Page, JDBC, EJB , Apache and MySQL etc development of technology
	ORDBMS --- bridge the gap between relational database and the Object -Oriented model	
	Traditional RDBMS product focus on the effective management of data	
	Parallel Database products are in market	
Late 1990	Explosive growth of World Wide Web , the database system had to support very high transaction-processing rates as well as very high reliability and 24 * 7 availability	
	Database system support Web interfaces to data.	
In 2000	Major development which took place during 1997 was the introduction of the Extensible Markup Language (XML) (DuCharme, 2012).	Growth of open source database system - MySQL , PostgreSQL
	New query language XQuery was introduced for XML	
	Databases have become more sophisticated and have features such as triggers, cascading update, and delete. This prevented updaters from creating inconsistencies between the tables.	
	Databases also developed a simplified procedural language	
	Specialized techniques -- column stored system, highly parallel database system designed for analysis of very large database.	
	Specialized Distributed database system have been built to handle the requirement of very large Web site such as Amazon , Face book, Google , Yahoo!, Microsoft etc	
	Work on management and analysis of streaming data such as stock market ticker data or computer network monitoring data.	
	Data mining techniques are widely used. Applications are Web-based produce recommendation, automatic placement of relevant advertisements on Web pages.	
21st Century	Data is used in large science databases like the genome project, the geological, national	Sybase/SQL Servers' transact SQL, Oracle's PL/SQL, and

	security, and space exploration data	PostgresSQL.
	Huge terabyte systems are also appearing and will require novel means of handling and analyzing data,	
	New concept Data mining, data warehousing, and data marts was introduced for handling terabyte data	
	Carlo Strozz introdeced NoSQL - It is lighweight Open-Source relatioanl database that did not expose the standard SQL interface	
	In 2011, DB developers started working on **UnQL (Unstructured Query Language)**, a specification for a query language for NoSQL databases	
	UnQL is claimed to be a superset of SQL within which SQL is a very constrained type of UnQL for which the queries always return the same fields, same number, names and types	major internet companies, such as Google, Amazon, Twitter, and Facebook used NoSQL concept for data management
	UnQL does not cover the data definition language (DDL) SQL statements like CREATE TABLE or CREATE INDEX.	

The database evolution study suggests that from year 2000 the growth of internet application is increased. The need of database is not restricted to the database experts. Hence a user friendly, portable, flexible database as well as query language is required for manipulation of data information. The XML serves all these needs. It is acts as database as well as transforms the data from one application to another application. It is a textual data format with strong support via Unicode for the languages of the world. Thus XML is defined [24] as a general-purpose markup language that is used to describe both data and a corresponding structure in machine-human readable format. It is a simplified descendent version of Standard Generalized Mark-up Language [21] which was originally designed in 1960 to facilitate information-sharing among homogeneous systems, XML initially appeared in 1996 and in February 1998 it was accepted [24] by the World Wide Web Consortium (W3C) as the standard medium for exchanging the data over the web.

2.2 XML History

XML is emerging as a de facto standard for information exchange over internet. It is derived from SGML and HTML. The current infrastructure is available to deal with HTML content and the same infrastructure can be re-used to work with XML; hence most of the companies can easily integrate their business document. For example, General Motors [25], Jet Blue Airline use XML in their business.

Apart from these characteristics, the XML has additional characteristics and these are: - high flexibility, platform independency, portability, simplicity, and usability. It is a human language which is conversable and readable by the people who had no formal introduction to XML. Flexibility means programmer or user can create his own tags; which are not limited to standard tags; which are not predefined by programmer. XML is vendor independent and system independent also. If a program is using vendor dependent tags, there are limitations such as the browsers and other programs associated with it. In such a case, these tags need to be approved of by the concerned authorities. Even the users have to get accustomed to the usage. This is a time consuming process. This limitation is overcome by XML. It is fully compatible with applications like JAVA, C etc and it can be combined with any other application which is capable of processing XML, irrespective of the platform. Due to all these characteristics, today XML is in use in most of the businesses.

XML was emerged from SGML [24]. In 1980, SGML (Standard Generalized Markup Language) was defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than decade, but it was not well suited to serve documents over the web. In 1990 HTML came into picture. It is simple and has predefined tag semantics and tag sets, but unable to represent structural relationship and search utility. These drawbacks are overcome by XML. It provides a facility to define tags and structural relationships between them. There is no predefining meaning of tag set and preconceive semantics. It will either define by the applications that process them or by style sheet this enhances the functionality of the internet. Following fig 2.1 summarizes XML development.

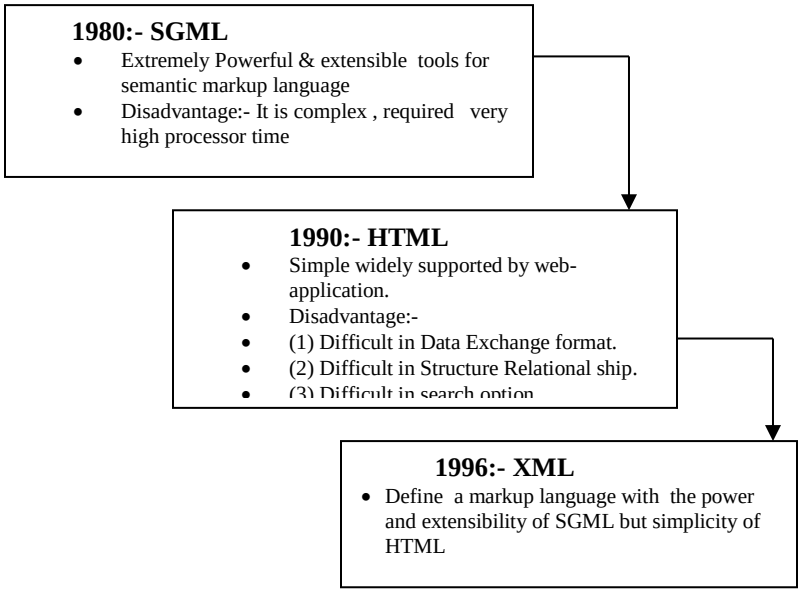


Fig 2.1: XML Progression

The XML file is simple text file which is composed of element node, attribute node and comment node.

2.3 XML Data Representation Components:-

The XML element is a piece of text that is enclosed within a start tag (<-- /) and end tag (/ -- >) It represents the object or entity. Element node is simple element, complex element and mixed element. An XML element can be associated with one or more XML attributes. The attribute node represents the characteristics of element node. There are two major issues to be considered when using attributes. An attribute cannot be repeated more than once within a single element definition and its value part must be enclosed in double-quotes as a string regardless its data-type. Three special-purposes attributes namely "id", "idref" and "idrefs" are used to support references between different parts of an XML documents. The benefit of this layout (1) minimizing the XML document size by preventing data redundancy (2) reducing the cost of updating such reference data by ensuring that update will take place at one position in the XML document. Comment node represents comments of the XML document. It can be found

anywhere inside XML documents and they are ignored by most of the API that manipulates XML documents. The XML document structure is mapped in fig 2.2 as follows:-

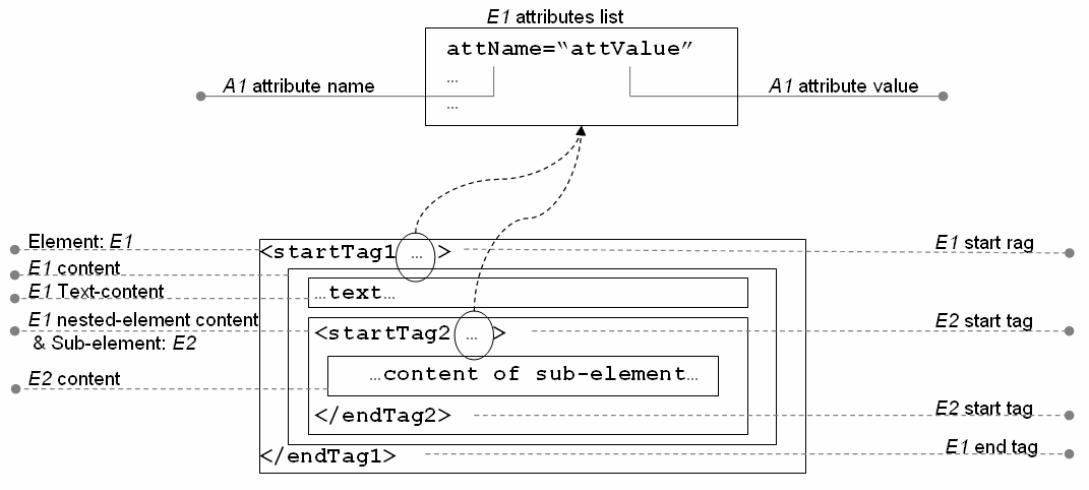


Fig 2.2 XML File structure

Other XML components are DTD (Documents Type Definition) and XML schema. They are used for validating XML document.

This XML file is converted into predefined data representation format which is called as XML data model. XML data is represented into different data model, some of the important data models are: (1) XPath (2) DOM (3) XML infoset (4) edge label tree (5) node label tree (6) OEM etc. This model has an important role in query processing. The desired data must be retrieved as per the requirement.

2.4 **XML- Query Languages:-**

XML supports two types of queries: IR Style and Database Style of query. Database style of queries is used on 'Data-Centric' application.

The Database queries are broadly classified into containment queries, order queries and content-based queries [1]. The containment queries retrieve 'Parent-Child', 'Ancestor-Descendent' relationships. The order sensitive queries are forward and backward navigation queries. They are following, following-sibling, preceding, preceding-sibling, child axis, self axis queries. The content-based queries are associated with element or attribute node content .These query check the attribute and element content and retrieve it.

These queries are further classified into: tree structure queries and starting node queries. The tree structure queries return a small tree. They are further classified into simple path query which corresponds to a chain-path and branch path expressions – which corresponds to a small tree, called twig. The starting node queries are further divided into "total matching queries and partial matching queries. The total matching queries start from the root of the document where as partial matching queries start from some internal node.

IR-Style of queries is used to query text-dense XML repository whose value elements are involving long text. These types of queries do not work well in common database query standards. These queries are classified into DB+IR Style and IR-Style of queries. DB+IR queries enhance database-style XML queries such as XPath and XQuery queries. For IR-Style of queries, a full-fledged retrieval technique is required, for example 'keyword' search is used for it. Some of the query language supports IR Style of query, or Database Style of query or both.

XML has a rich set of query languages. Milestone of XML languages [14] are listed on the following tables2.2:-

Table 2.2: XML Query Languages

Query Language	Language Type	Input Model	Class of Query	Public Recognition
Lorel[49]	Declarative	OEM	Path expressions within OQL	1997
XML-QL[48]	Functional	Node-Labeled Tree Data Model	Pattern Matching	1998
XQL[50]	Functional	Node-Labeled Tree Data Model	XQL based on path expressions	1999
Quilt[51]	Functional	Edge - Labeled Tree Data Model	Quilt Expression	2000
XPath[16]	Functional	XPath	Axis query and predicate query	2002
XQuey[15]	Functional	XPath	FLOWR expression	2007

XPath and XQuery [5] are two major languages for database-style of queries and originally developed and recommended by the W3C Consortium [21]. XPath [22] is a basic query language that selects the nodes from XML documents from the root path. By using XPath query, the XML tree is traverse in different direction. XQuery [23] language is more expressive than XPath. An XQuery include axis traversal of XML tree as well as For-Let-Where-Return (FLWR) clauses, which can be nested and composed with full generality [7] means each clause includes a sub-XQuery.

The XPath is used for traversal of a complete XML file in different directions like forward, backward direction. There is lots of work is performed on the query optimization of the XML file. XQuery not only traverse a complete tree but also performed selective operation on it. It performs almost all type of query operation of XML database. Much expressive powered of XQuery increased the optimization and evaluation complexity. In literature different XQuery optimizer techniques are proposed by tree algebra for XML (TAX) [8] generalized tree pattern (GTP) [27], and tree logical class (TLC) [17] etc. For our research we select XQuery language and all different types of query, especially FLOWR clause which may return a single node, twig pattern and all order node i.e. following, following-sibling, preceding, preceding-sibling etc.

Another reason of selection of XQuery Language, it satisfies the need of all community i.e. "Document Community ", "Database Community" and "Programming Community".

- Document-Centric applications often require text search, XQuery supports operations on document order and axis expressions, which are used to navigate in the document and to access the context of particular document fragments.
- Data-Centric applications often require very efficient selection, retrieval, and transformation of small fragments of data stored in massively large databases. XQuery incorporates features from query languages for relational databases (SQL) and Object-Oriented databases (OQL).

XQuery is a purely functional language which supports user defined functions and recursion.

The different feature of XQuery Language as per specification of [W3C3, W3CS4] is:-

- It is an XML query language with some programming language features and SQL-like semantics.
- It is developed from XPath data model.

- It has all functionalities, libraries and capabilities of XPath2.0 (i.e. XPath 2.0 is a subset of XQuery)
- It is supported by most commercial RDBMS such as IBM, Oracle and Microsoft SQL-Server.
- In addition to XPath2.0 capabilities, XQuery supports FLWOR [W3CS4] expressions (FLWOR is an acronym for "For, Let, Where, Order by, Return")
- XQuery1.0, a W3C Recommendation on 23/Jan/2007, is the latest version of XQuery.

Hence the researcher used the XQuery language for the research the detail of XML query processing is discusses as following:-

2.5 XML Query Processing :-

As per W3C [8] XML processes are divided into two steps: - "External Query is processing "and "Internal Query processing "[5]. The fig 2.3 provides the abstract view of XML query processing.

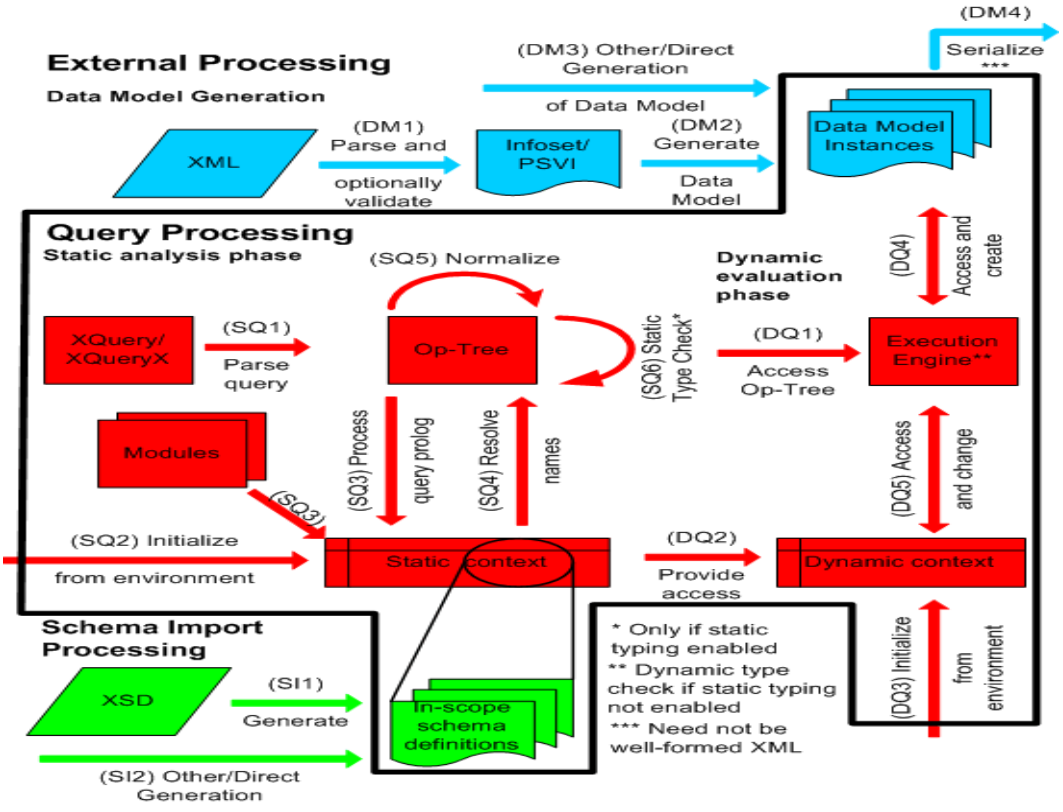


Fig 2.3: XML Query Processing

External Query Processing: - In these steps, the XML document is converted into data model. This data model is used for query processing. It is a two step process:-

Step I: - Data Model Generation: - XML file is parsed. Different tokens are generated and are validated against elements and attributes declarations and target namespace. It generates information set called as PSVI (Post-Schema Validation Infoset). From PVSI XML-data model is generated.

Step II: - In scope schema define: - XDM i.e. XML Schema Definition Model is used for defining schema definition. It defines all schema used by XML file. The file is divided into three parts: - in-scope schema types, in-scope element definition and in-scope-attribute declaration. 'In-scope schema type' includes definition of all types of definitions found in imported schema. 'In-scope element declaration' includes all element declarations found in imported schema. 'In-scope attribute declaration' defines all the attributes found in imported schema.

Internal Query Processing: - In this step XQuery is processed against XML file which is stored into XML Data model**.** Following steps are used for query processing:-

1. **Parsing**: - The query is parsed and checked for grammatical error, i.e. syntax error. If no error is found, an internal operation tree of the parsed query is created.
2. **Normalization or OP-Tree Generation: -** In this process, complex query is rewritten into simple query expression i.e. converting the query expression into core language. If the query is not well-typed, static type errors are generated. For instance, a comparison between an integer value and a string value might be detected as a type error during the static type analysis.

For instance, the following [expression/query]

```
for $i in (1, 2),
   $j in (3, 4)
Return
  Element pair { ($i,$j)
```

is normalized to the Core expression

```
For $i in (1, 2) return
  for $j in (3, 4) return
     element pair { ($i,$j)
```

3. **Static type analysis**: - This is an optional step. Static type analysis checks whether each query is well-typed, and if so, determines its static type. Static type analysis is defined only for query Core language. Static type analysis works by recursive application of the static typing rules over a given expression.
4. **Dynamic Processing:-**This step is divided into two sub steps.
    - Dynamic Context Processing: - The dynamic semantics of query depends on the dynamic input. Dynamic input defines processing environment.
    - Dynamic Evaluation: - In this phase, dynamic value of a query is computed. All dynamic values are provided in this phase. It may be intermediate result of the operation, or reading the value of variable at runtime, dynamic variable. This Dynamic evaluation may result in a value OR a dynamic error, which may be a non-type error or a type error.

In XML Query Processes, important task is XML Data Model generation i.e. Organizing XML data file in such a way that desired data can be accessed easily.

XML is a hierarchical database and semi structure database. It doesn't have fixed tuple structures for storing a record in the database. The data in XML is tagged with an infinite set of user-defined tags that come in pairs and can be nested to arbitrary levels [26]. Therefore, storing data into proper table structure is a challenge. The XML data should be organized in such a way that it should maintain 'Sibling', 'P-C' and 'A-D' relationship. Once the data is stored into a file, then indexing must be created for fast accessing data which will provide a good query performance result. In literature, different approaches are proposed for XML indexing and query processing they are:-

2.6 XML Query Processing Approaches:-

Indexing plays an important role in any database to retrieve specific data without scanning a complete file. Well known Indexing structure (hash tables and B-structure as well as multidimensional index structures) has been developed in RDBMS as well as OOAD databases [20]. In XML, index creation is a difficult process because data is hierarchical or semi-structured. In literature, different types of indices are created on it. For example, structure index (index on XML document structure), Value index – index on node value and Path index (index

on tree path). In the following section, we describe XML indexes and query processing approaches.

XML is a text file. For effective query processing, researchers have developed different types of storage of the XML file. Broadly these storages are classified into [9] 1) Relational Storage. 2) Native Storage and 3) Navigation Storage.

A. RDBMS Approach:-

XML is a hierarchical, recursive and ordered database. Relational database provides week support for hierarchical data. Therefore, it is difficult to convert XML data into relational format. In relational conversion, XML to RDBMS, schema defines a set of table for fixed or frequent structure; sometime additional ordering column is added for supporting order data. It avoids storage of label and factorizing storage of values for good query performance.

Once data is shredded into tables, full power of relational engine such as indexing and query capabilities can be used to manage, and query data [8]. There are two approaches to design database schemas for XML documents: Structural mapping and Model Mapping.

(a) In Structural mapping approach, relational schema is defined for each XML structure, a relation or a class. It is created for each element type in the XML documents.

(b) In Model mapping, a fixed database schema is defined for XML document which supports dynamic operations such as insertion, updates and deletion effectively. This approach is suitable for dynamic XML document. In this approach, fixed databases schema is used to store the structure of all XML documents.

B. Native Approach:-

Native Approach of XML database model integrates specific techniques for storing the data, and accessing XML documents. These techniques include new storage methods, indexing structure, and join algorithm for query processing. The Native Database Architecture is divided into two types: Text-Based Native XML Database and Model-Based Native XML Database.

(a) **Text-Based Native XML Database**: - XML file is stored as a text file and indexed on element content or element path. The file is stored using ‘BLOB’ or ‘CLOB’ data type. Text-Based Native XML Database architecture speeds up the data retrieval process, because the data is stored into continuous bytes on the disk. Therefore, single index lookup retrieves the desired result.

(b) **Model-based Native XML database**: - XML document builds an internal object model. This object can be stored into Relational database or Object Oriented database. The objects are interpreted according to database style. For example, XML file is read from DOM object. If it is to be stored into rational database, it has to be stored into different tables such as Elements table, Attributes table, PCDATA tables, Entities tables and Entity-References tables. If it is stored into OODBMS, each XML node is interpreted as an object, and it is stored into ORDBMS format. This format is being used by Oracle and DB2 since 2007, after introducing new data type '.XML', which stores XML files in hierarchical format.

The advantage of this is a new XML storage technique is listed below:-

1. Native XML database has ability to confirm an XML document against different XML Schema; but in RDBMS, a document is always confirmed against a schema. One as to one relationship is maintained in RDBMS.
2. Native XML database supports XQuery language.
3. XQuery expression contains different 'Conditional Expressions' and user-defined functions, which require the confirmation of the documents against the schema. This schema may have multiple versions. This is possible only in Native XML Storage.
4. Data retrieval is faster than RDBMS because Native Database stores the entire XML document physically together, and uses physical pointer between the parts of this document. This allows document to be retrieved either without join or with physical join. This join is always faster than the logical join which is used by the RDBMS system.

C. **Navigation Approach**:-

In this approach, XML data arrives in the form of data streams, without any associated indexes, and the entire XML data tree has to be traversed node by node in the depth-first order, which preserves the original XML document order. The key idea of this approach is the use of run-time stack data structure for query evaluation. XML file is read character by character and its tokens are stored into a stack. When an opening tag is encountered, the stack-top element is transformed to new state. These newly generated states are collected into a new stack. After calculation on these states they are then pushed into the stack. When a closing tag is encountered, the stack-top element is simply popped out of the stack.

2.7 XML Indexes Approaches: -

Indexes prebuilt on XML data can facilitate XML query processing by locating data quickly. Index is built on traversing paths with the element set or on the value of elements and attributes. Broadly, XML indices are classified into three types: Structural Summary Index, Structural Join Index, and Sequence Index. Detail discussion is as below:-

a. **Structural Summary Index: - In** 'Structural Summary index', indices are built on XML document paths. Therefore simple path query i.e. identification of P-C relationship can be solved by this index. The Structural Summary Index includes associated XML document path, element set and the paths emerging from these elements. The Structural Summary Index is implemented as tree structure data type in which each node represents a tag name and its associated path starting from root node till the concerned node. These indexes are further classified as:-

a. **Selective Index***:* - From the literature study these indexes are created on the attribute or element value. They are further classified into: Simple index, Full-text**.**

b. **Simple index**: - This index returns the set of elements and attributes that satisfy a certain condition. The condition might be the content associated with an element or attribute (value-based index), or their tag names (name index). Developers typically build this index using relational index technologies, such as B-tree, Order index.

b. **Full-text indexes: -** This index returns the set of elements that satisfy a certain condition over their textual content. Full-text index typically rely on inverted index, which associates each word with its position in its host document. Developers can implement a full-text index as a B-tree. The location scheme is used for building it, and can answer full-text queries easily.

c. **Structural Join Index:** - Structural Join Index, index attributes and elements with a particular name. In general, developers use these indexes in the context of join-based processing in which the query processor first determines elements that match query tree nodes. Optimization of XML queries is a key issue in Structural join and researchers have proposed various techniques for solving it. Two major techniques are: i) Merge Join Algorithm. ii) Stack-based Algorithm. According to literature review, this type of index is very important for queries evaluation because they process both DB and IR Style of queries. This index is classified in three main groups: Node Index, Path Encoding and Sequence based index.

1. Node Index:-

The researchers have identified two effective approaches for storing and accessing XML document: XML-to-RDBMS mapping approach, and Native database approach. The first category uses the mature relational database management system to store and manipulate the XML data and XML queries into SQL. Once data is converted into relational tables, then it takes advantage of existing indexing, recovery mechanism and concurrency mechanism. The main concern of such mapping approaches is to find the best relational representation for the XML which preserves the whole XML semantics, while minimizing the cost of the mapping process in terms of processing-time and storage-space. The main crisis with this mapping is document order semantics and unique identification of a record but it is solved by Node Indexing Schema. In this schema, a unique code is assigned to each node in the XML tree and preserves the nested hierarchical relationship and structural relationship. It is structure-orientation approach which reflects the document hierarchy in the index structure. These types of indices are good for answering content based queries. These indices are created on XML data storage. According to Weigel et [8], a labeling schema for a document tree D is a decentralized structure summary of a specific set of tree relationships in D.

The main goal of the existing node-labeling solution is: a) assigning a unique code for each node in the XML tree. b) Preserving the nested hierarchical relationship (structural) relationship during XML updates, c)Minimizing the re-labeling cost (including the processing time and I/O accessibility) in the case of data updates, and d) reducing the required storage to store generated code [18]. The label of the node suggests the position of the node in tree, and relationship with other nodes.

The main problem of this approach is updating cost is high because when new node is inserted or deleted from the tree, re-labeling is required i.e. each node is assigned a new label again. Another problem is 'Label length'. As XML tree grows depth wise, the node-label length is increased and after 5th level it is impossible to read. [15]. Most of the research tried to reduce re-labeling. But these new techniques fail to support specific requirement of processing hierarchical, semi-structured data. Node Label indexes can process 'Total Match Query' but they are insufficient for processing 'Partial Matching' and 'Content-based' query.

2. Path Encoding:-

The Path encoding schema stores the entire path summary information. The XML distinct paths from root node to any arbitrary node in the document are stored. The queries are executed by assessing this path summary information instead of executing it on the structure of XML document. This indexing technique is ideal for evaluating simple path query instead of evaluating multiple path queries. Because it involved multiple join operation for the execution.

3. **Sequence-based Indexes*:-***

Sequence based indexes convert both the XML query and XML documents into sequences and use the well-established sequences matching techniques to obtain query answers (e.g. [12] [13]). The good feature of sequence-based indexing techniques is that they use the entire query tree as one unit, thus the expensive join operations are avoided during the query evaluation.

In this approach, XML documents and branching queries are represented as sequence, and subsequent matching provides the query answers. These approaches don't support queries with selection condition over nodes. The Sequence approach has the following limitations: 1) it supports only the ordered twig queries. 2) It may involve a large number of index investigation and index-probe operations might result in high random disk I/O costs. 3) It might repeatedly visit many data nodes unnecessarily. The pictorial representation of all above approaches is listed in fig 2.4

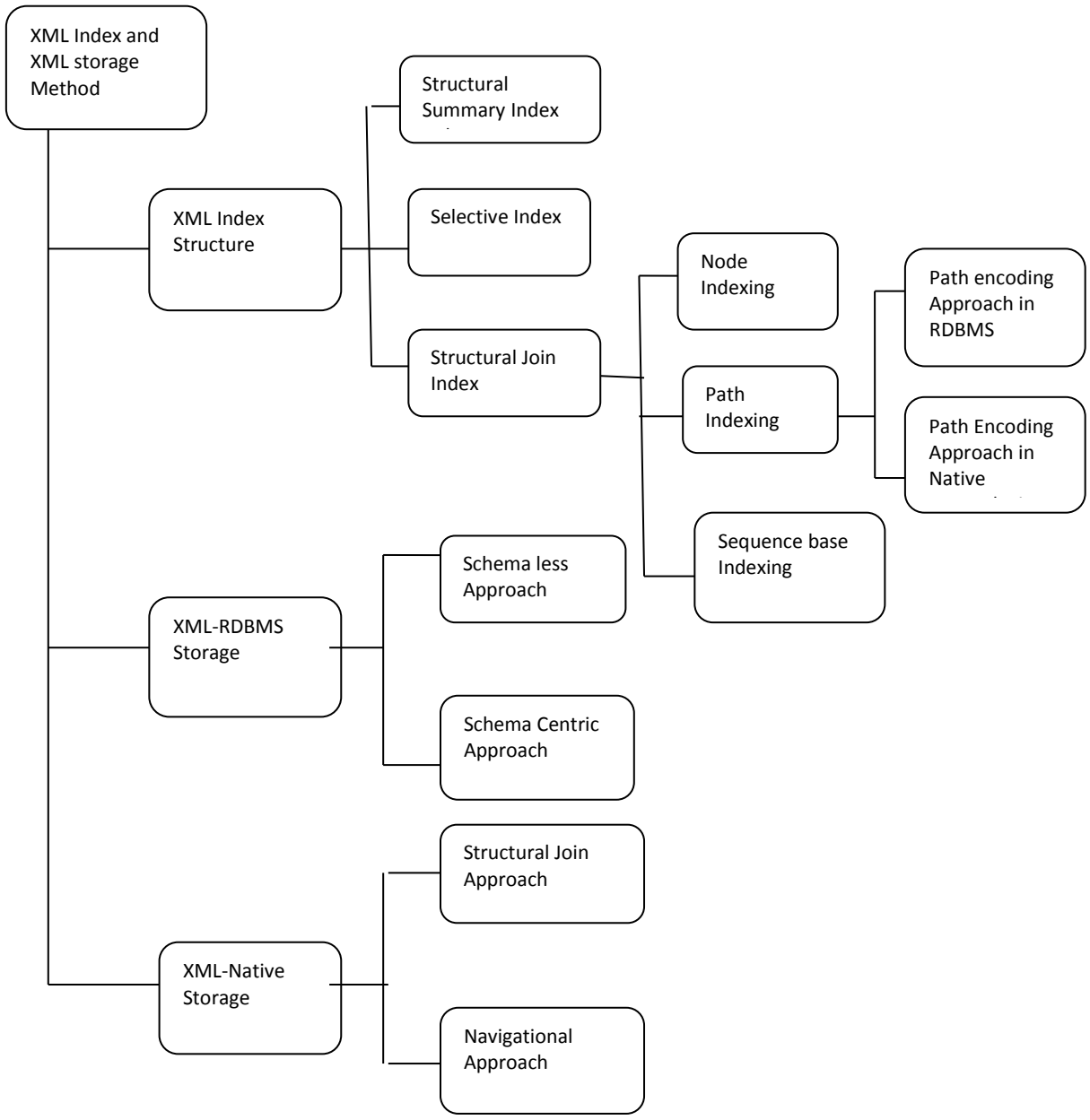


Fig 2.4 XML Index

The detail discussion of XML indexing and XML storage are discussed into next chapter.

Conclusion and Summary:-

This chapter discusses the progression of DBMS. Then it discusses the preliminary information of XML and its storage and indexing technique. We also discuss the query processing technique and XML indexing approach. The detail literature review and limitations and challenges are discussed in the next chapter.

References:-

1 Barbara Catania and Anna Maddalena, Athena Vakali "XML Document Indexes: A Classification " SEPTEMBER • OCTOBER 2005 Published by the IEEE Computer Society 1089-7801/05/$20.00 © 2005 IEEE IEEE INTERNET COMPUTING

2 Burton Grad "Relational Database Management Systems: The Formative Years " IEEE Annals of the History of Computing Volume 34, Number 4, october–December 2012 pp. 7-8 | 10.1353/ahc.2012.0037

3 Burton Grad "Relational Database Management Systems: The Business Explosion" IEEE Annals of the History of Computing Volume 35, Number 2, April–June 013 pp. 8-9 | 10.1353/ahc.2013.0025

4 Burton Grad, Thomas J. Bergin "History of Database Management Systems " IEEE Annals of the History of Computing Volume 31, Number 4, October-December 2009

5 Byron Choi, Mary Fernandez, Jerome SimeonThe XQuery Formal Semantics: A Foundation for Implementation and Optimization May 31, 2002

6 D. Chamberlin, J. Robie, D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Source. In WebDB (Informal Proceedings), pages 63-62. 2000

7 D.D. Chamberlin, "XQuery: An XML Query Language," IBM Systems J., vol. 41, no. 4, 2002.

8 H. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A Native XML Database. The VLDB Journal- Volume 11, pages 274-291, 2002

9 Ioana Manolescu-Goujot INRIA Futurs - LRI - PCRI, France Gemo group XML query processing: storage and query model interplay

10 J. Robie et al. XQL (XML Query Language). http://www.ibiblio.org/xql/xql-proposal.html. 8/1/1999

11 Jonathan RobieXML Processing and Data Integration with Xquery Published by the IEEE Computer Society 089-7801/07 2007 IEEE

12 M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi. Efficient schema-based XML-to-Relational data mapping. Avaliable online at www.sciencedirect.com since 15/12/2006]20/1/2006,

13 M. Rys. XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005. In Proceeding of the 2005 ACM SIGMOD International conference Baltimore, Maryland, pages 958-962, 2005on Management of Data,

14 Mikael Fernandus Simalango "XML Query Processing and Query Languages: A Survey" Property of Amikelive.com – Technical Paper Series 25/10/2008

15 R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In proceedings of the 18th international conference on Data Engineering, Washington DC, USA, pages 129-141, 2002.

16 S. Abiteboul et al. The Lorel Query Language for Semistructured Data. In International Journal on Digital Libraries, 1(1):68-88. 1997

17 S. Paparizos, Y. Wu, L.V.S. Lakshmanan, and H.V. Jagadish, "Tree Logical Classes for Efficient Evaluation of XQuery," Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), 2004.

18 S. Sipan, K. Verma, J. Miller, and B. Aleman-Meza. Designing a high-performance database engine for the 'Db4XML' native XML database system. The Journal of Software-69, pages 87-104, 2004Systems

19 Silberschatz-Korth-Sudarshan " Database System Concepts 6th Edition , ISBN: 0390586099

20 T. Fiebig, S. Helmer, C. Kanne, G. Moerkotte, J. Neumann, and R. Weld. Anatomy of a native XML base management system. VLDB Journal- Volume 11, Number 4, pages 292-314, 2002

21 W3C Consortium, http://www.w3.org, 2006

22 W3C Consortium, XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/, 2006.

23 W3C Consortium, XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/, 2006

24 W3C Website. Extensible Markup Language (XML) 1.0 (Fourth Edition). Online:http://www.w3.org/TR/REC-xml/, Accessed on: 30/10/2006

25 Yi Chen , Susan B. Davidson, Yifeng Zheng "A bi-labeling based XPath processing system" Information Systems 35, 2010, doi: 10.1016/j.is.2009.05. ACM

26 YoshiKawa, M. and Amagasa, T. (2001). XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology,

27 Z. Chen, H.V. Jagadish, L.V.S. Lakshmanan, and S. Paparizos, "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

Chapter 3

Review of Literature

3.1. Introduction

Since from last decade, XML is used as a standard media for data representation, query processing, and interchanging information among various web applications. As a result, the size and the number of XML's users are increased dramatically. There are two types of groups who have interest on XML utilization, the first group is mainly interested in XML as a humanly consumable exchange format. They are looking up for tools that allow easy manipulation and transformation of small XML document. The second community is mainly interested in XML as a data storage format. Their primary focus is to design the query languages and storage methods to select data from vast amounts of data efficiently. Thus both communities are required a proper data storage and proper data retrieval method. These requirements classify the XML documents into two types: data centric document and document centric document.

The query technique is different for both types of documents. The data centric document required to store the XML information into the structural format and text are less narrative. On other hand, text in document centric documents is more narrative, and loosely structured, while the element's order being is not significance. So the query processing technique is different for both the files. In document centric application the main interest is of retrieval of data only, the structure is less important; therefore information retrieval (IR) techniques are used to processes such documents. The proposal [44] had identified three major challenges for IR techniques. The challenge is the result return by KWS techniques is no longer the entire matching document. The second challenge is how to rank these results. The third challenge is proximity computation. The proximity competition is a process of computing the keyword in the text file. But in XML document it is very complex process because XML document structure is hierarchical and recursive.

A new language XIRQL [48] was design for address the problems of IR technique. It is content-based XML query language built on top of XQL [70]. (XML Query Language) To solved the IR problem; the structural semantic of XML document is stored. To store the structural semantic of the XML document index is prebuilt on XML leaf node. XIRQL[An Extension of XQL for Information Retrieval 48] pre build index on XML leaf node. This pre-setting of this index node increased updating overheads. This overhead is avoided by XRank [38] and XSearch [62].

But these proposals ignore the type of searched-keyword (i.e. whether a keyword is textual word or node-label word). Hence it returns wrong result. Therefore this approach fails to solve keyword-enriched query. This problem is investigated in [30] and the proposal introduced new operator "about" for searching operation. The "about" operator has the same syntax as the standard "contains" operator [W3C4] but with two more unique features. The "about" operator allows the formulation the search requirement in terms of mixing the contents and the structure of XML document and it allows the use of best-match querying of content-based XML documents. So, the XML structural semantic still can be preserved by using strict XPath axes while the contents can be searched by injecting one or more "about" operator into the XPath expression. A recent solution for the XPath's "about" extension problem is proposed in [29]. Suggested that, incorporating some semantic issues from XML Fragments query language [14] into traditional keyword searching approaches will result in a better query performance for document-centric XML databases.

From the literature review, the most of the research proposal address IR challenge and find out effective solution on it. In data centric document the queries are retrieving the XML node or a branch of a XML file. The different axis operations are used for XML query execution. For retrieving desire data information a proper indexing mechanism is required. This study is more focused on evaluating data centric query and indexing technique for it. Therefore the study is performed with two facets: XML database storage with query execution policy and XML indexing structure.

Broadly literature study is classified into two parts: XML storage and XML index technique. The detail discussions are as follows:-

There are three main alternatives to store XML document. The first approach is mapping XML data into relational database system (RDBMS), there are many variations involved into XML-RDBMS storage like Object Oriented Database System (OODBMS), XML-ORDBMS storage and second approach specialized data management system known as native database (NXD), the third approach is native storage in which XML data must be stored in commonly used flat flies in the form of original XML file. The indexing technique is not applicable of it. Complete XML documents needs to scan during query processing technique. This technique is applied for special-purposed XML applications in which data must be stored in flat file format.

3.2. XML – RDBMS Storage: -

In this storage XML file is stored into different RDBMS-tables. These proposals address the following problems:-

1. Lack of schema definition:-One of the important characterizes of XML is "user friendly "means is the user can add the data as per the user choice. User can create his/her own new tag as per the requirement. This characteristic suggests the XML data is always extensible; it doesn't have fixed format. So designing of XML schema for it is a challenge while in RDBMS the table has predefined schema and the data can added in it.
2. XML always represents hierarchical relationship between different elements, mapping this hierarchical relationship into relational table is difficult.
3. XML data is recursive in nature means the same tag name are repetitive e.g name tag represent book name, sometime it represent author name, publisher name also. Means there is redundancy in same tag name. Mapping this redundancy is very difficult in RDBMs.
4. XML data is mapped from different document into relational table. It is very difficult to maintain this information into relational table because XML data in mapped into different tables, maintain 'Document order' is redundant task.
5. Since, the XML data is represented into different tables the query execution time is very high.

In literature XML-RDBMS proposals are classify into two types (1) Schema less storage and (2) Schema centric storage.

3.2.1 XML-RDBMS Schema less Approach:-

As name suggest, the schema definition is absent, the XML data is mapped into different tables. The hierarchical order of database is maintain by adding additional field into relational table. From last decade the research has been done on manipulation of this characteristic. These proposals shred RDBMs Data into number of tables. Hence the 'joint' operation is increased while executing the query.

This multiple join operation increase high query execution time. To solve this problem different proposal have been introduced but these proposal can perform inner join and outer join query evaluation effectively but cannot execute equijoin operation. One of the major issues is mapping the documents; there is probability of loss of information due to shredding XML document [35]. Another important issue is orthogonality, i.e. after conversion of XML-RDBMS format; it is

difficult to preserve the original XML document information back [25] that is it cannot convert back to XML documents.

In 1999 a very first XML-RDBMS mapping technique [15] was designed. This is called as basic edge approach. It defined three proposals for storage: - Edge Approach, Attribute Approach, and Universal Table Approach. These entire proposals maintain XML information with a single table. Since only a single table is maintain so it is very easy for storing XML information. The start and end tag information is maintain by the table so P-C type of queries can solved easily. But these approaches are unable to maintain the structural relationships between the two nodes. Thus unable to answer A-D or twig queries. This the order attribute is also not maintain by these proposal so couldn't solved order queries.

The proposal [19] introduced additional feature for query execution. This proposal [19] designed a single table instead of storing information into multiple tables. This table is more compact and an external program translate XML query into standard SQL. This format provides maturity, scalability, portability and stability since majority of the data on the web are currently resides in RDBMS. But this is unable to maintain the structural relationship between the XML tables.

The proposal [13] maintains structural relationship in which each XML node is represented by (startid, endid, level) number. The 'started' and 'endid' numbers maintain structural relationships between nodes. The 'Level no' determine P-C and A-D relationship very well. But query performance is increased because it involves maximum join operation and has high updating cost. The high updating cost means when new node is added or deleted from the XML file then XML node_ids are need to reliable. Therefore new node addition or deletion needs to reconstruct the XML file which is very costly. This approach is unable to execute equijoin operation

[64] Suggest the solution of high updating cost and equijoin operation. The proposal maintains two tables for XML data - Parent table and Child table. The key idea of the proposal is in parent table the 'LParent' column solves P-C and sibling queries. But this column is not sufficient to solved A-D and twig queries; it requires additional computation processing cost. The node identification cost increased query processing cost. The proposal provides the solution for join query but unable to answer the order queries.

In literature different proposals were suggested similar to proposal [64].These proposals were good for answering small XML file, but as number of records increased, the proposal failed to

answer the different queries. This problem was solved by s-XML [17] mapping method. But this model was not capable for answering order types of queries. For answering the order queries MAXDOR [28] model was proposed. The model proposed new concept of node_id. These node_id represents in four dimensions. This node_id maintain structural relationship between document and preserve the document order for answering order queries. All content of XML files is stored into only one table so the 'join' operations were reduced. The node_id itself help in answering all types of queries. The major drawback of this approach is high node_id computational cost. As the depth of the file is increased the length of the node_id is also increased. Since the XML data is maintained only into one table hence the inner join operations are increased for answering these queries. These join operations are reduced by Compressed XPath Query Tree (CXQT) model [75]. As per the proposal, the document order is maintain two kinds of indices: Basic Label Index (BLI); and Structural Summary Index (SSI). The BLI is used for XML data Storage and SSI is used for XML query.

In literature "path indexing" technique is also introduced. The path index is nothing but the each XML path was encoded and the index was created on this path. The XREL [81] proposed basic 'Path Materialization' approach. It stored the tag path from the root to each node and node_id information. It involved fewer join operations than 'Node-Approach' because it answered queries in term of path rather than an XML node. It also answered twig queries effectively by using the Optional String Pattern Matching function ("Like") provided by SQL. Although the PM approach reduced the number of joins operation but it increased the complexity of the selection operation in query processing operation. The PM approach creates B+ tree indexes on the path strings. However, B+tree indexes cannot support "Like" efficiently. Another limitation of PM was it might result in incorrect query answers when recursion exists in XML data. The Reverse Path approach [63] overcomes this drawback of the PM approach. The key idea of RP was stored reversed root paths of data nodes. The proposal not only stored path information but also stored the node_id information of XML tree. The RP approach used ORDPATH [53] encoding schema for answering maintaining the structural relationship between nodes. This approach always returns correct query result although recursion exist into XML file. This approach is integrated into IBM system RX [37], Microsoft SQL Server 2005[53] and Oracle DB [17].

The RP approach reduced general purpose "Like" matching to the easier task of string prefix matching (SPM) [76]; it didn't discuss efficient implementation SPM. This implementation is proposed in BLAS [76] approach. The key idea is encode each Reverse Path into a 'PLabel' value. Each XML nodes were labeled and stored the information as D-label. This D-labeling schema was used for maintaining structural relationships between nodes. This approach answers content-based queries very effectively but cannot provide a solution on sibling queries as well as order queries. Moreover this schema was expensive in query execution cost because cost of P-label and D-label is high. The same proposal was modified for evaluating order queries [78]. This approach executes all types of XPath queries.

The conclusion of schema-less approach, it required high query execution time due to multiple join operation on different tables. These proposal supports inner and outer join operation effectively but unable to support equijoin operation. Another important issue, during mapping process the documents may loss of information due to shredding XML document [35]. Once the document is converted into RDBMS file than it is difficult to preserve the original XML document information [25] i.e. it is difficult to convert back to XML file.

3.1.2 XML-RDBMS Schema –Centric Approach:-

XML file is created by validating DTD or XML schema information. The RDBMS schema is reconstructed from the XML schema. XML tables are constructed from DTD; it used the data clustering concept for RDBMS table conversion. But this conversion work well for simple query execution but has problem of performance degradation for complex query evaluation Main challenge of this approach is selection of good XML schema and effective mapping XML-to-RDBMS and translation of XML queries into SQL. [34] Proposed transformation of XML queries into SQL queries [41] Proposal introduced XML storage by using DTD definition. This proposal implements different constraints such as key constraints and inclusion dependencies in the RDBMS after conversion of XML to RDBMS format. The query execution time is very high due to high number join operation. Several algorithms have been proposed for schema mapping by researchers, but the problem of data mapping has not been discussed thoroughly in the literature. In [45] addressed the problem of data mapping and defined an efficient and linear algorithm for mapping XML data to relational data. This proposal mapped XML data into relational table but it ignored ordered nature of XML elements and not dealt with the referential integrity constraints in data mapping algorithm. XTRON [36] presents an XML data

management system using a relational database as a repository which does not incur the modification of an RDBMS. Another proposal ODTDMap, [45] propose: (I) a lossless schema mapping algorithm to generate a database schema from a DTD, which made several improvements over existing algorithms, (II) two linear data mapping algorithms map ordered XML data to relational data The algorithms were efficient and well scalable with respect to the size of input documents. This work only considered 'string' data type in XML Schema. It needs to extend the work for other data type.

3.3. Native Storage and Query Processing:-

Another popular approach for storing XML file is Native database approach for storing XML database. NXD storage defines logical model for an XML document, it doesn't required any physical storage model. This model works well especially on handling huge amount of data. These data can be store and retrieve in their original structure. For mapping XML element additional processes is not required. But many important database features has to build from scratch such as user access management, transactional manager can adopted with minimal changes from current DBMS system.

The NXD storage classify into two main categories: text based storage, model-based storage. Text based storage store XML document in text form and provide some database transaction supporting accessing document. In model based storage well-designed XML API such as DOM [72] is used to retrieve whole original XML document. Though method is very simple but has two disadvantage first while processing query , document is completely read and parsed every time and second document is kept in memory while querying [3].

The native database is very good for storing small XML data and stores all components of the XML model intact. [42] Proposal, the data retrieval is faster than relational database because entire document is store together physically or use physical pointer between the parts of document. This mechanism allows document to be retrieve either without joins or with physical joins, both of which are faster than the logical joins used by relational database. This technique performs full text search and proximity search more effectively then RDBMs techniques [39].

Designing of new NXD storage is costly because Native storage does not have mature database management system. All database operation need to be developed from scratch. The different algorithms like query optimization, transaction management, storage management need to develop. It is difficult to adopt this newly developed NXD technique by previous developed

application because it is difficult to transfer the data from RDBMS to NXD database. The new algorithm needs to develop for it [34].

For answering queries XML need to traverse many individual element to element paths which required multiple index lookup [33] hence query execution time is also increase. XML queries are more complex because they contain regular path expressions [68]. Thus additional flexibility is needed in order to traverse data whose structure is irregular or partially unknown to the user. In the databases, complex search operation or aggregations operations are bulky, this operations are very difficult to achieve in Native databases. For evaluation of a single query, multiple index lookup is required. Another problem is redundant traversal of a XML tree for answering a single query. Other major problem is the query evaluation and intermediate query result is very high in NXD database. These problems are address by different proposals. These proposals are discuss below:-

Native storage proposals classify into three categories: navigational approach, join approach, hybrid approach. A navigational approach traverses the tree structure and tests whether a tree node satisfies the constraints specified by the path expression [16] [61].

The join approach is a list of XML tree nodes are selected .These nodes satisfy the query constraint associate with it and a pairwise join operation is performed on their structural relationships (e.g., parent-child, ancestor-descendant, etc.). The 'Join-approach' is scalable to large document and query by relying on the secondary storage to store intermediate results. A major disadvantage of the 'Join-approach' is the updating cost. Since it does not rely on navigation to discover the structural relationships between nodes, encodings of the structural relationship must be pre-computed and inserted into the indexes.

Third approach is a hybrid approach [6]; it is a combine of the navigational and join-based approaches. The key idea is to, first, navigate through the XML documents to obtain intermediate results that contain structural information, and then join the intermediate results based on the structural information. This avoids the update problem since the structural information is obtained dynamically. The hybrid approach also avoids the complexity of matching recursive documents which was noted as a problem in the navigational approach. The detail discussions of these proposals are as follows:-

3.3.1. Join Approach:-

The first proposal MPMGJN algorithm [13] proposed for answering containment queries. The research proves that the MPMGJN algorithm is faster than standard RDBMS join algorithms. This algorithm used hardware cache utilization for faster query performance. This approach answers all types of queries. The query result is retrieved by processing binary join operation, which increases the query performance cost. A different cost-based query optimization algorithm is required for reducing this cost.

Although the MPMGJN algorithm outperforms standard RDBMS join algorithms, but it can perform a lot of unnecessary computation and I/O for matching basic structural relationships, especially in the case of parent-child relationships (or, direct containment queries)[60] . The proposal developed two families of structural join algorithms for this task: 'tree-merge' and 'stack-tree'. The 'tree-merge' algorithms are a natural extension of traditional merge joins algorithm and the recently proposed multi-predicate merge joins operation on it. The stack-tree algorithms have no counterpart in traditional relational join processing. The stack-tree algorithms have worst-case I/O and CPU complexities.

These approaches solved simple path query easily but it is difficult to solve 'twig' query. Twig query is nothing but a set of binary (parent-child and ancestor-descendant) relationships between pairs of nodes, e.g., the parent-child relationships and the ancestor-descendant relationship. The twig query can solved by two steps :- (I) matching each of the binary structural relationships against the XML database, and (ii) "stitching" together these basic matches. The first step is solved by proposals [13] [60]. For the second step requires a 'joins' operation for identifying a good join ordering in a cost-based manner, taking selectivity and intermediate result size estimates into account. The basic limitation of [13] [60] approach for matching query twig patterns is that intermediate result sizes is very large. The proposal [47] reduced the intermediate query result size by introducing two algorithms: 'Path Stack' and 'PathMPMJ' algorithm. The 'Path stack' is generalizing structural join algorithm and 'PathMPMJ' is binary join algorithm. The proposal said that I/O and CPU optimal among all sequential algorithms that read the entire input, and has worst-case complexities linear in the sum of input and output sizes but independent of the sizes of intermediate results [47]. Another 'TwigStack 'algorithm refines 'PathStack' algorithm which provides the best result other than among all sequential algorithms that read the entire input.

A number of algorithms have been proposed to process a twig query holistically. Those algorithms are quite efficient for quires with only ancestor-descendant edges. But for the queries with mixed ancestor-descendant and parent-child edges, the previous approaches still may produce large intermediate results, even when the input and output size is more manageable. To overcome this limitation, [31] propose a novel holistic twig join algorithm, namely 'TwigStackList'. The main technique is to look-ahead read some elements in input data streams and cache limited number of them to lists in the main memory. The number of elements in any list is bounded by the length of the longest path in the XML document. Proposal claims that it reduced I/O cost and CPU cost also. 'TwigStackList' [31] method avoids generating large intermediate results which do not contribute to the final answer. The proposal also reduced CPU and I/O cost when twig patterns only have ancestor-descendant relationships.

Another important direction of XML query processing is to build structural indexes [114] [101] [97] over XML documents to avoid unnecessary scanning of source documents. XML structural indexing is a technique to partition XML documents. It is called as streaming scheme in the [66] paper. This paper develop a method to perform holistic twig pattern matching on XML documents partitioned using various streaming schemes. The method avoids unnecessary scanning of irrelevant portion of XML documents. More importantly, depending on different streaming schemes used, it can process a large class of twig patterns consisting of both ancestor descendant and parent-child relationships and avoid generating redundant intermediate results.

The holistic twig pattern matching [47] [66] has been developed in order to minimize the intermediate results, i.e. only to enumerate those root-to-leaf path matches that is in the final twig results. However, when the twig query contains parent-child relationship, these solutions may still generate large numbers of useless matches [47]. The paper [65] shows that the explicit join of individual root-to-leaf path matches or even the enumerations of these path matches are not necessary for processing twig queries.

Another challenge is to process XQuery is that there may be multiple path expressions in the FOR, LET, WHERE and RETURNS clauses, all with different semantics. The research proposal[65] introduced new concept of GTP (Generalized Tree Pattern) which avoid repetitive task in query execution and reduced intermediate result [65] propose a novel *hierarchical stack encoding* scheme to compactly represent the twig results. This scheme also reduces the complexity for twig query processing by assigning node_id to each XML node. This encoding

scheme is used in ‘Twig2Stack’ algorithm, [82] to our knowledge, this was the first GTP matching solution that is free of any post path-join, sort, duplicate elimination, and grouping operations. This algorithm lowers the query execution time and space overhead. It is better than TwigStack [65]. But the algorithm Twig2Stack takes much more time then ‘holistic matching’ [47] algorithm because it has complex stack manipulation and overhead for processing "non-productive" nodes in the stacks. It uses much more space because more non-productive nodes are being pushed onto stacks. These overheads also have in the two-phase algorithm [40]. This problem is address by a one-phase holistic twig evaluation algorithm [84] that outputs twig matches in their entireties without a later merge process. Instead of outputting individual path matches. This method holds the matching path temporarily. The process is continuing until entire twig matches are formed. The proposal used a new regional encoding schema for encoding XML node. This new regional coding schema represents the left, right positions and level number of the node respectively with additional information of Top_Branch_Node, Upper_Node and Lower_node and ClosePatternAncestor. This algorithm takes less time and space than [47] [84]. Another problem of “twig2stack” is time and space overhead. To decrease the time and space overhead, [4] proposed a non redundant compact XML (NRCX) storage for storing XML document. The stored data is queriable i.e. it does not require decompression. It maintains the document order. The size of structure index is small even for very large XML documents. This proposal reduced XML storage size and only one file is used for storing XML file. A single index look up identifies the index entry and retrieve the desire result and no additional join are required for answering total match queries. This technique is suitable for document centric XML document which do not contain attributes. This method process containment queries effectively but unable to processes the order sensitive query, partial matching and twig queries.

3.3.2. The Navigation Approach

In this approach XML data is stored in flat files, i.e. in the text file, without any indexes. The XML query is also represented as a flat file i.e. It is considered as a text file. The XML query processor actually scans the entire XML document sequentially element by element and if the matching query node found, it returns the query result as an output. The XML file reading and query reading are performed by the selective dissemination of information (SDI) application. The core component of SDI is the ‘document filtering’, which matches each XML document from the publisher with a collection of queries from the subscribers, to determine which sub

queries have at least one match D, rather than finding all matches of the sub queries in D. In this processes the XML data is available in the form of data streams without any specialized data structure. It is considered as a simple string. The entire tree has to be traversed node by node in the depth-first order. Each sub tree is stored on a separate disk page. The record of each node on a disk page can include the (tag, id) pairs of all children of that node, which records the information on the tree edges for facilitating forward navigation, or it can also include the (tag, id) pair of the parent of that node for facilitating backward navigation.

XML filtering systems aim to provide fast, on-the-fly matching of XML-encoded data to large numbers of query specifications containing constraints on both structure and content. It is now well accepted that approaches using event-based parsing and Finite State Machines (FSMs) can provide the basis for highly scalable structure-oriented XML filtering systems. The first proposal XFilter [2] was introduced in 2000. XFilter used a separate FSM per path query and a novel indexing mechanism to allow all of the FSMs to be executed simultaneously during the processing of a document. From this work the new method "YFilter" [77] was introduced which exploit commonality among queries by merging common prefixes of the query paths such that they are processed at most once. The resulting shared processing provides tremendous improvements in structure matching performance but complicates the handling of value-based predicates. All this streaming system has high CPU and I/O cost. An early XPath stream-querying systemXSQ [18], [19] incurs exponential (query) time complexity. Recently, some XQuery stream-querying systems have also been developed such as XSM [5], Raindrop [22], FlowGraph [73], and SAXON [43]

3.3.3. Hybrid Approach:-

Generally the backward or forward navigation is not effective, since each may access a large number of queries irrelevant to data nodes, which are scattered across a number of disk pages. Therefore, this operation cause high disk I/O cost. The query performance is often very poor since most of the elements that are irrelevant are also scanned. So additional processing is required for the identification of the required nodes. The other weakness of this approach is that it works well for single-node solutions (e.g. a set of nodes) but has difficulties to derive tuple solutions (a set of (A, B, C) tuples). This disadvantage is overcome by combining the native query processing with the navigation query processing.

A good performance is achieved if it involves both tree-based navigation and structural join algorithm. There are a number of options for how to combine tree-based navigation and structural joins based on information retrieval-style inverted lists, and that these options can have widely varying performance. A new algorithm "ZigZag Join" [1] introduced for designing query execution plan. This algorithm is the extension of the Multi-Predicate Merge Join (MPMGJN) algorithm. Another proposal next-of-kin (NoK) pattern matching [49] algorithm proposed a novel approach for efficiently evaluating path expressions by NoK pattern matching. The result of NoK pattern matching greatly reduces the number of structural joins. More effective algorithm was proposed by [73] for effective execution of XQuery over streaming data. The frame work developed a series of optimizations which transform XQuery queries so that they can be correctly executed with a single pass on the dataset. BlossomTree. [51] Algorithm combines the advantages of all above algorithm and avoids the disadvantages of the navigational and join-based approaches. The key idea is to, first, navigate through the XML documents to obtain intermediate results that contain structural information, and then join the intermediate results based on the structural information. In this way, it avoids the update problem since the structural information is obtained dynamically. And it avoids the complexity of matching recursive documents in the navigational approach.

For processing multiple queries on XML database [50] proposed index filtering algorithm. Index-Filter uses indexes built over the document tags to avoid processing large portions of the input document that are guaranteed not to be part of any match.

Similarly the recent proposal for multiple query execution is SIMP [7] which reduced redundant traversal and scalable organization of XML document.

3.4. Index Strategy used in XML:-

The XML data is mange either by RDBMS system or Native database system. Effective indexing technique is required for retrieving the XML data. Thus a proper indexing technique is required. But in the XML databases, designing of the XML indexing is a challenge for the database communities because XML file is in irregular structure and the lack of data-typing [58]. Researchers classify our indexing study into three aspects: 1) Type of Documents Index 2) Index residency and 3) Structural join indexing

XML is store into either document centric file or in data centric file. As per the data retrieval aspect different types of queries are used. Information retrieval technique is used for reading data from document centric file while path traversal and a particular node data retrieval technique are used for data centric application.

The main memory style of index defines how to store entire documents into main memory. These types of index have excellent response time for structural-join queries because they avoid expensive cost of I/O operations. Its implementation is straightforward and feasible for querying small XML document and inefficient for large XML document. Because loading entire XML file from secondary memory into main memory increased high I/O cost hence the need for disk-based algorithms is required. Now a day memory is not a problem thus disk base indexes can used memory as per their requirement but reduce query processing time is a major challenge for these index.

Alternatively, XML indexing algorithms can be evaluated on the basis of how the XML structural relationship is encoded in the index file [44] .These index map the document property like edge structure, values and their node relationships in the index using physical and logical address. These properties are mapped as per the query work load.

In RDBMS system DBA select the index on certain filed. This selection is depending upon frequently trigger queries, or some other criteria are used for selection of Indexes. In some research proposal similar technology is applied so researchers refer this index as selection index.

Detail discussion of all these index types are discussed below:-

3.4.1. Structural Summary Index:-

A structural summary is smaller version of an XML tree where all paths from root node to any leaf node in the actual XML tree are preserved. The main goal of a structural summary is to eliminate redundant structural information without losing structural constraints; that is the structural relationship between XML element such as P-C and A-D relationships. Data Guide, forward and backward similarities, inverted list etc data structures are used for creating the structural summary information.

It records the complete paths from root node till leaf nodes. These types of indexes are convenient for processing path-based queries; but not convenient for branch path queries.

Furthermore index strategy is best for answering the content based queries. However the index creation process is complex; and it is exponential to database size; and also expensive for those XML files which have high structural inconsistency.

As per our knowledge the first proposal [58] first techniques of XPath traversal proposed by [McHugh 1999]. It is a type of total summary index. The XML data is read from OEM and represented into the Data Guide', a new data structure. The data guide generates the path index that summarized all paths. The query path is searched in data guide first. If the query path exists in Data Guide, then the data is retrieved from the respective data path. This technique is suitable for answering the simple path queries. But this is not suitable for answering the A-D and the twig queries. The additional processing is required for answering content-based query. Sometime for answering some queries, Data Guide is required to traverse a complete original data source. For the irregular XML file, Data Guide is much larger than the original XML file. To overcome this drawback [58] introduced the new concept 'Strong Data Guide' for storing XML file. But the index size and updating cost is very high.

For reducing index size the new concept of bi simulation was introduced by index family: 1-index, 2-index T-index [97]. The 1-index supports only for simple path query, 2-index supports multiple path queries, T- index also support same and build appropriate template for query in advance. The 1-index structure supports only "Total Matching" queries. 2-index executes 'Total Matching' queries, 'Partial Matching Queries' though additional computation is required. The researchers observed that most of the information in the database has fixed pre-defined structure and only certain components are irregular. Therefore, rather than indexing all paths, it is better to index only certain paths of the tree that has high percentage of accessing. T-index reduces the size of the index. It ensures that every node in the index appears only once in the index. But still the size of index is large for a highly unstructured data. This approach is unable to solve the problem of 'Partial Matching Queries' and twig queries as well as A-D types of queries. And dynamic updates are not possible.

The problem of large index size is reduced by [101]. The A(K) index path that has a length that is less than 'K' hence it is not appropriate for answering query which has length greater than 'K'. The parameter k controls the convergence and the size of A (K)-index. If K is too small, the index cannot support long path expression accurately. If K is long query result is more accurate, but the index size is large. This index is good for short queries; but it cannot provide an accurate

answer for longer queries. It supports dynamic update. It supports only Path Query for single P-C relationship. It does not support answering twig queries.

All above two problems: high index size and dynamic update are undertaken by D (k) index [114] The D (k) index is process adoptive ability to adjust the index structure according current query load. It can support frequently used path expression by adjusting its structure according to the query workload. Each index node v in D (k)-index has two attributes: - One denotes the local similarity requirement on v, and other denotes v.extent which is the set of data nodes associated with v. The proposal based on the external memory cost model. Hence it takes maximum time for query processing. D (k) index only consider regular path expression in the proposal but in real life structure is more complicated than the regular path expression.

Another external memory index is index fabric [8] which has a B-tree like structure which uses 'tries' as data structure for creating an index. It uses the layered-based paging strategy to index large data. The index structure is stored on disk, and divided, into multiple blocks of approximately equal size. Each block holds small sub-tries. The tries of the lower levels are referenced by higher levels tries. This index is suitable for answering 'Partial Matching' queries. Either the 'Temporal Path' or the 'Refinement Path' is created for answering the 'Partial Matching Queries'. The temporal path creation is not a dynamic process; it needs to be determined prior to the index creation and loading time. Therefore, it is time consuming process and not applicable for dynamic XML file. These indexes are disk based indexes, and hence they take time for query processing.

[10] Suggest the proposal on query workload. It doesn't keep all paths starting from root. It utilized frequently used path by applying sequential pattern mapping technique. . The APEX index consists of two structures, graph structure (GAPEX) and hash tree structure (HAPEX). The GAPEX represents the structural summary of XML data and a hash tree structure (HAPEX) consists of all incoming label paths to the respective nodes. The hash tree is used to find nodes of the structure graph for a given label path. This proposal answers well for shorts queries but cannot answer 'twig' queries effectively. Index structure considered only the frequent path queries and neglect other types of queries.

This method doesn't maintain all paths starting from the root node, it utilizes only frequently used paths, by applying sequential pattern mining. The APEX index consists of two structures, graph structure (GAPEX) and hash tree structure (HAPEX). The GAPEX represents the

structural summary of XML data and a hash tree structure (HAPEX) consists of all incoming label paths to the respective nodes. Each label in the data graph is kept as a node in the HAPEX known as 'hnode'. Each 'hnode' points to either a node of GAPEX or another hash table. Similarly, each node in GAPEX (known as xnode) maps the entry of a 'hnode' of HAPEX. An extent i.e. a set of edges in the data graph is assigned to each 'xnode.' The 'Apex' index provides a good answer for short queries. It can answer P-C, A-D and total match queries; but cannot answer 'twig' query effectively.

XML data is rich in structure; this calls for advanced indexing techniques in order to facilitate efficient query processing. 'Forward and Backward' index [32]. Address the lack-of-scalability and lack-of-efficiency issues associated with the in-memory F&B Index. It is a disk-based F&B Index that features several good clustering properties. Those properties not only enable efficient tree traversal, but also give rise to several novel query processing methods based on disk scan and segment based join. The disk-based F&B Index has several salient features, including cache-friendliness behavior and good scalability. It also significantly outperforms alternative tree-based or join-based query processing systems. Index size often tends to be too large and no supporting updating index needs to be regenerated.

Thus to speed up path expression queries over the graph, a variety of structural indexes have been proposed. A (k)-index introduces the concept of local bisimilarity for partitioning, allowing the trade-off between index size and query answering power. However, all index nodes in A(k)-index have the same local similarity k, which cannot take advantage of the fact that a workload may contain path expressions of different lengths, or that different parts of the data graph may have different local similarity requirements. To overcome these limitations, researchers propose M (k) - and M*(k)-indexes [98]. The basic M (k)-index is workload-aware: Like the previously proposed D (k)-index, it allows different index nodes to have different local similarity requirements, providing finer partitioning only for parts of the data graph targeted by longer path expressions. Unlike D (k) - index, M (k)-index is never over-refined for irrelevant index or data nodes. However, the workload-aware feature still incurs over refinement due to over-qualified parent index nodes. Moreover, fine partitions penalize the performance of short path expressions. To solve these problems, the M*(k)-index is used. An M*(k)-index consists of a collection of indexes whose nodes are organized in a partition hierarchy, allowing successively coarser partitioning information to co-exist with the finest partitioning information required

The paper presents the concept of minimal infrequent structures (MIS), [71] which are infrequent structures in XML data, whose substructures are all frequent. Indexing the MIS of a document collection comes at several benefits compared to using path indexes. First, the proposed method indexes not only paths, but also arbitrary useful structures of high selectivity. Second, it is space-efficient, requiring only a limited amount of storage. Third, since the MIS are infrequent, the index prunes large amounts of data fast. Fourth, due to its lightweight nature, it can be used independently, or in combination with other indexing and query processing techniques. Finally, not only does it provide a fast means of query evaluation, but it can also be used to spot queries

This research combine IRST [80] with structural index and propose IRST (k, l)-index, an adaptive structural index that can support branching path queries efficiently. The Inter-Relevant Successive Trees (IRST) is an efficient model for storing and indexing massive full-text queries. It has good properties, i.e., single-chain, compressibility which bring out a series of advantages, such as multifunctional query, full-text regeneration, pattern mining, and small expansion ratio Since IRST has high performance and extensibility, it is extended to the XML data graph and propose the IRST (k, l)-index, an adaptive structural index that can support branching and simple path queries efficiently. The research proposal support branching path queries efficiently. The IRST (*k, l*)-index performs efficiently in terms of construction time, space consumption, and query performance for branching and simple paths.

SIMP [7] approach try to reduced redundant traversal of query execution, and almost all structural queries are solved without traversing the original XML tree. The proposal describes the XML file with Node view and Structural Summary representation. Node view includes a hash entry and a split-by-label structure. Hash entry points to each node in the split by label structure. Simple queries are evaluated on structure queries without visiting XML document. Node view is used to filter out queries that lead to empty results. This method is efficient but still have some disadvantage :- (1) the method support for the queries those have common prefix part. The method is not explored for the queries that have common suffix part. (2)The method execute queries of type (/, //,*) only remaining queries – order queries, contentment queries and branch path queries are not explore.

3.4.2. Selectivity Index:-

These index techniques are implemented from the RDBMS literature. In RDBMS literature the indexes are created on certain fields which are selected by Database Administrator (DBA). The field of indexing is selected on different criteria like on frequently triggered XML queries, the fields which satisfy certain SQL queries. The same concept is implemented in XML. T-index index [68], index fabric [8] has proposal in which frequently queries templates are created and index for querying XML data.

It summarizes all frequently used paths, or only subsets representing most common paths, or path templates corresponding to the frequent queries. These indexes are capable of handling partial queries as well as twig queries successfully. But the additional processing cost is required for answering content-based queries.

3.4.3. Structural Join Index:-

It built the index on the attributes and the elements with a particular name or identity. These indexes return elements based on their structural relationships (ancestor–descendant or parent–child). With the help of these node query processor, returns different nodes whose content satisfies a given condition. After returning all possible nodes in the query they are joined by using structural join algorithm. Researchers classify this index into four categories: (1) node indexing, (2) path encoding and (3) sequence based indexing and (4) feature based indexing technique.

3.4.3.1. Node Indexing Technique:-

This is a best method for indexing XML information. Each node in the XML tree is assigned a unique number. This node_id represents Ancestor/Descendant or Parent/Child relationships. [40] Suggest that the designing of node_id structure solved the problem of reconstruction and decision. From the literature review for our study researchers classify the 'Node Index' into three types: Sub tree labeling schema, Prefix labeling Schema, Multiplicative Labeling Schema

In Sub tree Labeling Schema, each tree nodes are labeled by their node positions in XML tree. This position is obtained by traversing the complete tree in a specific order. Sub tree Labeling Schema is classified into two types: Interval Encoding and Regional Encoding. Interval

Encoding XML tree is traversed in either preorder or post order manner and generates a unique number for each node. This node-id can be accelerating query execution fast. In Regional Encoding schema**,** the entire XML tree is divided into some regions. Each region is assigned an identity of < startid, endid> number. This type of schema is good supports text based database storage.

As per our knowledge [55] introduced first labeling schema. Each XML node is labeled by ((First Number, Second Number). The first number is 'preordered tree traversal' number and second number is 'Post Order tree traversal' sequence. This number represents structural relationships between XML nodes. They can identify A-D relationship easily but it is difficult for identification of P-C relationship. This schema has high updating cost because after insertion or deletion of new node required to recalculate XML tree numbers.

For reducing updating cost [57] suggest new coding method, each XML nodes are represented by a pair <order, size>. The 'order' is a preorder tree traversal number and its size is any arbitrary integer larger than total number of current descendents of the nodes. The order attribute determines the relationships between nodes. As per the researchers the size parameter avoids relabeling cost. This parameter determines the reserve memory space. As soon as memory space is filled, entire re-labeling of the tree nodes is required. But the determination of the 'size' integer is a difficult task. Thus updating cost is one of the major problems in labeling schema. The proposal [9] tries to reduce it. But instead of that it is more focus on defining containment relationships between XML nodes. For XML node_id representation, it adds one additional parameter 'level' which determines the depth of the tree. In query processing operation node selection id perform on selection of matching parameters of ((started, endid, level) between two nodes. This proposal effectively answers all types of query but introduced new problem of multiple joins operations in query execution. So it increases query processing cost. Still updating cost is high, after addition of new node and deletion of new node required entire relabeling.

For reducing updating cost and query performance cost XML node are relabeled by the combination of parent node and self node. This style is called Prefix labeling Schema. Instead of representing node_id in number format it represent it into string. [69] Represent the node_id by concatenating the ancestor string and string of current node. (Each String-id (node-id) is the concatenation of the path-id from the root node till the last node) e.g. 1123 means third child of second child of first child of root node. This representation leads the problem of storage space

hence it is not applicable for large volume of the database. Additional cost is required for computing the containment relationship. This schema partially supports dynamic updates. When new nodes are added at the end of each level of the tree, then new node-id can be added. But if the new node insertion is at the end of the tree, then a complete re-labeling is required for that tree level. Another simple proposal for assigning node_id information to the XML element is proposed by [52]. The proposal use a 'dot' separator delimiter is used for assigned node_id to each XML node. The XML node label is a combination of 'parentid. nodeid'. i.e. 1.2.3 is third child of second child of first child of root node. The delimiter determines the structural relationships between different node_id. But this still dynamic updating problem is not solved. And label size is varying. It is directly proportional to the depth of the tree.

These problems are address by [17] the node label are assign by combination of '0' and '1' string. The Label size is increased as per the depth of the tree. The "Clue" technique is used for reducing the label size. . It estimates the maximum and minimum number of descendent nodes in the future insertion. But practically, 'Clue' implementation is impossible. The reduction of label size is still open problem for node indexing technique. The proposal made the provision for addition of the new node data. No relabeling is required but there is no provision for reutilization of label of deleted node. Once the label is assign then it cannot be used by the system even though the node is deleted from the list. This increased the label size. Thus the reutilization of existing deleted label is another issue of the research.

[53] Suggest new method for reducing updating cost. The key idea of this research is utilized only odd numbers for assigning to node_ids. And all even numbers are kept aside for assigning to the updating new nodes. As per our knowledge this is considered to be the first schema which allowed arbitrary updates without changing any existing label. It was implemented by Microsoft® SQL server™. . But the query execution time is high because additional query processing operators are required for deciding the prefix level of the nodes. The Continuous Fraction Encoding (CFE) Code [79] is assign to XML node for reduction of the updating cost. CEF code completely avoids re-labeling. A new CFE node can be inserted between any two consecutive CFE nodes. But once the node is deleted from the list, it cannot be reused for labeling code. CEF code identifies containment relationship very well but cannot support twig query evaluation. The node label is too large so increased label_id A proper indexing

mechanism is required for better query performance. Reutilization of deleted label node is proposed in [27] It define P-C and A-D relationship. But the label size problem is still open.

The type of node labeling schema assigns a number to each node by using some mathematical property. Relationships between nodes can be computed, based on some arithmetic properties of the node labels**.**

Prime numbers [74] are assigning to each XML nodes. The numbers can be assigned either by 'Top-down approach' or 'Bottom up approach'. In Bottom up approach a prime number is assigned to each leaf node. Then, for each subsequent level, the parent's label becomes the product of their children labels. In 'Top-down approach', the first prime number is assigned to root node '1'; and the next number is assigned to the next node is the product of parent_id and the subsequent prime number. This method is good for dynamic updates. But have the label size problem; each prime number can only be used once for assigning to a tree. The size of the label increases as it reaches the bottom of the tree. The size storage of the label has direct impact on the performance of query processing. Novel idea is proposed by [12] QED. The XML tree nodes are represented by the combination of four numbers 0,1,2,3. Thus the label length is always less than four digits. Interpretation of node labels is complex; a specialized program is required for its interpretation. This increased query processing cost. The method effectively handles containment queries but cannot processes 'twig' query, as well as ordered query. [11] Proposed two method for numbering XML nodes: Variable Length Encoding "VCDBS" and Fixed Length Encoding "FCDBS". In FCDBS algorithm assigns node_id to static XML tree while "VCDBS" is used for dynamic updating. [12] And [11] proposal decreased the size of the node label but they required special algorithm for node interpretation.

A new technique "Compact index tree" (Ctree) [4] introduced for indexing XML data. Ctree provides concise path summaries at its group level and detailed child-parent relationships in arrays at its element levels. The array in a group provides direct mapping from elements to their parents. Such a fast child-to-parent access is essential for efficient branching query processing. The Ctree-based query processing method enables early pruning of a large search space. Ctree is able to capture one-to-one parent child relationships (regular groups) and have monotonic relationships, which can be used to speed up query evaluation. In addition, instead of using global IDs, the group-based element reference which facilitates stepwise early pruning, efficient value processing, heterogeneous values differentiation, and efficient XML data updating. Thus

Ctree a structure index such that it alone can answer both single-path and branching queries without consulting the XML data. The values in XML documents are usually heterogeneous, including many types of data such as date, number, and text. These values are index into inverted list which help for answering value predicate queries.

3.4.3.2. Path Encoding Indexes: -

This type of index have idea of creating a path summary of XML data to speed up the processes of query evaluation. These index stores all path information starting from root node to any arbitrary XML node and indexes are created on these paths. This technique is implemented in RDBMS storage and native storage.

1) Path Encoding Approach in RDBMS Storage
2) Path Encoding Approach in Native Storage

a) Path Encoding Approach in RDBMS Storage Strategy :-

Many techniques have been proposed to store XML data in RDBMS in order to leverage advantages of relational database engine in storing, querying, indexing and updating data. RDBMS path encoding approach stores all possible paths from root node till leaf node of XML tree. Many different tables are created for this storage XML contain and maintaining document order relationship. Researchers have already discuss these techniques in relational database storage 'Path Materialization '(ref page no and para no). [81][26][76][78]

These relations are used to speed up XML queries over a relational database engine, reduced the storage space and avoid expensive structural joins that are required to evaluate branch path queries. But this technology has major disadvantage of recursive self joins even for simple path expressions. Updating XML data dramatically increases the updating processes for the mapping data.

b) Path Encoding Approach in Native Storage :-

A path summary for an XML document is simply a list of all distinct paths that basically build-up the actual XML document. A collection of all path summaries for an XML document or a set of XML documents are linked together in a tree-like graph called summary graph [80]. This summary may be total matching path of XML file or partial matching path of XML file. It is

implemented by computer native language. This implementation is Path-Encoding approach in native storage researchers have already discuss this implementation in type of selective indexes.
In native path indexing proposals are Index family [80], A (k)-index [59], D (k)-index [59], APEX [56] etc. These indexes are used the bi-similarity principal and build effective graph structure. The index structure is adjusted as per the query workload. They evaluate the query fast but have the problem of large index size and index updating cost is high. These indexes are fully concentrated only on the structural part of the XML file but no attention on the content of the XML file (i.e. value). Thus these indexes are inefficient for processing content based queries.

3.4.3.3. Sequence based Index: -

Sequence indexes transform XML documents and queries into structure-encoded sequences. Answering a query requires sequence string matching between the encoded sequences of the data and the query. This eliminates the need for joins to evaluate twig queries.
ViST [24] index structure encodes the XML tree and the query into structural-encoding sequences. The structural encoding sequence is a sequence in which XML tree is represented into two parts. The first part contains element tag and second part contains the root path of the parent node of the node list in the first part. For encoding node_id the tree is traverse in top-down fashion. As a result, for deep and large XML documents, the size of the index becomes a problem as it does not scale well with an increase in data size because the top elements have to be included within the sequence of the newly inserted elements. As the paths in XML data get longer, the sequence length will increase and hence the size of the index will increase exponentially in the size of data. The proposal listed another problem called the false-positives problem. This problem is noted when a branch node has multiple identical child nodes. It is resolved by disassembling the query tree at the branch into multiple trees, and using join operations to combine their result. This solution is definitely expensive, since it involves additional join operations.
ViST[24] is top-down transformation approach which has weakness in the query processing because it results in a large number of nodes (paths) being examined during commonly occurring non-contiguous tags names. This problem overcomes by PRIX [37] approach. So it transfers the tree in bottom-up fashion which plays a crucial role in reducing the query processing time.

PRIX is based on Prufer sequences. To illustrate how a Prufer sequence is used let's consider the following figure which denote a XML graph tree. The letters inside the node circles represent the tag types (names) and the numbers shown beside the nodes represent the post-order numbering of the tree. The index size and false-positives problem is reduced by this method. The size of a PRIX grows linearly in the total length of the sequences stored in it because an increase in the path length will result in a sequence addition which is equal to the amount of the increase.PRIX uses a complex four-phase refinement process to deal with the false-positives and the false-negatives that are associated with sequence index schemes. Basically, PRIX overcomes the false-positives problem by using document by document post-processing which is time consuming process.

In this approach XML documents and branching queries are represented as sequence, and subsequent matching provides the query answers. These approaches don't support queries with selection condition over nodes. The Sequence approach has the following limitations: 1) it supports only the ordered twig queries. 2) It may involve a large number of index investigation and index-probe operations might result in high random disk I/O costs. 3) It might repeatedly visit many data nodes unnecessarily.

The comparative study of different proposal are listed into annexure-I.

3.5. The Outcome of the literature review: -

From the literature review the following conclusion are noted :-

3.5.1. XML Indexing Structure:-

1. Structural Summary Index:-

- In Structural Summary Index the major problem is the index size of an XML file. Small size of index is good to fit into memory. Once index are resident into memory then query execution time is fast.
- Structural Summary index is a smaller version of an XML tree where all paths from the root node to any leaf node in the actual XML tree are preserved in the summary tree. Hence XML queries are executed on this tree instead of on complete tree. But sometime for query execution required to traverse complete XML tree instead of traversing structural summary information
- These indexes are not capable for executing 'twig' and 'partial matching' queries.

2. **Selective Index**:-

- Selective indexes are created on certain frequently asked query. But for other type of queries cannot take the advantage of this index.
- It is very difficult to select which data fragment are needed for creation of the index, in literature different data mining algorithm are used for this purpose but still it is upcoming area of research.
- These indexes are good for answering 'partial' matching and 'twig' queries successfully but additional processing cost is required for answering content-based queries.
- Selective Index has the problem of index size, it is still high. It is grows exponentially as per the document.
- Index updating cost is high. As XML document structure is change it needs to update the index structure.

3. **Structural Join** :-

- **Node Indexing**: -

  Each XML tag of the XML file is represented by node_id. These node_id represents structural relationships between XML nodes. It is very important property for maintaining XML node relationship. These nodes represent the query optimization plan for query execution. Researchers notice two important structures of node_ids: (1) Range Label Schema (2) Prefix labeling Schema (3) Multiplicative Node Schema. Researchers note our observation of these node-indexing is as follows:-

  i) Range Label Schema:-

  (1) These node_ids are evaluate the contentment relationship queries easily i.e. P-C queries but takes more time for evaluating A-D relationship. This label schema easily determines sibling relationship, Order sensitive queries. But these node_id have difficult in solving descendent axis, following –sibling, preceding-sibling queries.

  (2) The previous research proposals of range label schema don't consider the XML node order. As XML is hierarchical order sensitive data, it is important to maintain these order information. But this drawback is overcome by some of the current research proposal.

(3) The node updating cost is very high. When new labels are added either entire XML files nodes are need to re-label. To overcome this drawback in some research proposal some unused numbers are keep aside for future insertion, this un-utilized number increased numbering cost.

(4) Generally these label ids are not orthogonal.

(5) The size of Node_ids are fixed can fit into memory so that query execution time may fast.

ii) pre-labeling schema:-

(1) The structural relationship is identified by using certain computation on node label. Sometime these computation cost is high because it increases query execution time also.

(2) This labeling schema performs dynamic updates easily; a new node can easily add or deleted from XML tree. Relabeling is not required.

(3) The label length is extensive. It is increased as depth of the tree is increased. Sometime node_ids are impossible for reading. In some proposal these label length is decreases by additional computational cost.

(4) These labels are Orthogonal (i.e. Orthogonal means adopting another labeling scheme.)

(5) The label storage size is less because instead of storing the numbers in integer it is store into character data. Thus character data required only one byte for storage while integer data required 2-bytes for storage.

(6) In this node_id it is difficult to determine the order of the node. Determination of order the node is very important for query execution because XML data is a tree structure data. Thus this numbering schema cannot solved order sensitive queries but twig queries can easily solved by this schema as compare with range labeling schema.

- **Path Indexing** :-

  (1) Path Indexing Structure :- These index is classify into two types : (1) XML-RDBMS Path indexing (2) XML-Native path Index

  (2) All graph indexing schemes provide a complete answer for both path queries and twig queries. They do not require extra joins to evaluate the path queries but required additional cost for answering Twig queries.

  (3) Expensive computational complexity required to alter the index structure during the query-evaluation process which may result in slowing the query-evaluation process itself.

  (4) More storage space is required to keep the queries and/or the changes in the document's structure.

  (5) They are more concentrate on frequently trigger queries and ignores other queries which might involve heavy comparison operation

  (6) These indexes approach is effective in handling Parent-child relationship queries but inefficient in for handle path expressions containing ancestor-descendant queries, partial matching queries.

  (7) The major drawback: - the path summaries only XML data file paths. They never check the content of the XML nodes. (I.e. value of the node) hence fails to support selection conditions over internal tree nodes. Or on content type of queries.

  (8) It is still suffering the problem of high Index Size.

  (9) Updating Node cost is high Insertion/deletion of any new node required to reconstruct the index file

- **Sequence based Indexing :-**

  This approach doesn't support queries with selection condition over nodes. The Sequence approach has the following limitations: 1) it supports only the ordered twig queries. 2) It may involve a large number of index investigation and index-probe operations might result in high random disk I/O costs. 3) It might repeatedly visit many data nodes unnecessarily.

3.5.2. XML to RDBMS Storage: -

In RDBMS approach XML data is converted into tables and stored into relational database. All XML queries are transferred into SQL format, and existing query optimizer plan is used for evaluating queries. Major problem of this approach are listed below:-

- XML data is stored into RDBMS format with multiple tables. Thus for query execution more number of join operations are required.
- XML data is hierarchical ordered data. It is very difficult to maintain the order and hierarchical nature of XML into relational table.
- After storing the XML information into RDBMs format, it cannot execute equijoin operation on it. The inner join and outer join operation can easily perform on it.
- In some research proposal the XML data is maintain into a single table. It increases self join operation for query execution.
- All types of queries can solve easily. But not effectively evaluate twig and order sensitive queries.
- XML-RDBMS mapping is classified into two types: - (1) Schema less mapping (2) Schema Centric Mapping.
- Schema less mapping processes is also divided into two types: 1) Static Mapping and 2) Dynamic Mapping. In Static mapping processes there is possibility of loss of information of XML file. After mapping the information into the relational table it is difficult to convert back to the XML file. I.e. reconstruction of XML file is not possible.
- In dynamic mapping the main idea is to keep the original document untouched, so there is no need to reconstruct it again. But this approach suffers the problem of connecting with the previous existing data in the table. And specialized technique is required for translating XML query text into structured text operators.
- Schema Centric Storage Technique need database reconstruction when there is any change the XML schema, which is very expensive. The selection of a good relational schema selection from XML schema (or DTD) is difficult. XML-to-relational mapping on this selected RDBMs schema is complicated. A specialized technique is required for translating XML queries into SQL queries.

3.5.3. XML to Native Storage: -

In the native approach, XML data is stored on disk in the form of inverted lists, sequences, or trees and native algorithms are developed to further improve XML twig-query performance. However, in the native approach, many important RDBMS components such as storage management, access methods, query processing and optimization, and concurrency control and recovery, have to be built from scratch. These algorithm must care for

- Index size should small so entire XML file is kept into memory.
- Query repose time is directly proportional to I/O. High I/O the query execution time is high; low I/O query response time is low. In some proposal due to high I/O cost, query execution time is increased.
- In query evaluation the intermediate result is very high which increased storage space as well as query execution time. Thus reduction in intermediate result is required
- In native implementation equijoin operation are effectively implemented but other feature like storage management, access method, query processing and optimization and concurrency and recovery have to be built from scratch.

3.6. Summary and Conclusion:-

In concluding the chapter, researchers observed that for effective XML storage and index proposed method must consider structural join and determine a good tradeoff between the size of index and its answering power. The relational database management systems are promising media to store and retrieve XML data, but the current mapping approaches to map XML data into relational tables are still immature. The study noted that the path indexing structure is the best option for query evaluation. It not only reduced the index size by eliminating redundant path data but also improves the query evaluation processes by linking the indexed data nodes to their original structure. Researchers believe that if hierarchical structure is maintain with summarized path index then it is more effective for answering all types of query. This hierarchical relationship is maintained by defining a new node_id structure. This node_id structure is also facilitating the relational storage of XML data.

References :-

1 A. Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A.N. Rao, F. Tian, S. Viglas, Y. Wang, J.F. Naughton, and D.J. DeWitt, "Mixed Mode XML Query Processing," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

2 ALTINEL, M., AKSOY, D., BABY, T., FRANKLIN, M.J., SHAPIRO, W., AND ZDONIK, S.B. 1999. DBIS-Toolkit:Adaptable middleware for large scale data delivery. In Proc. of SIGMOD 1999. ACM Press, New York, NY, USA, 544-546.

3 An XML Indexing Structure with Relative Region Coordinate, In proc. of International Conference on Data Engineering (ICDE): 313-320.

4 Atul D. Raut, Dr. M. Atique Efficient Querying of Root to Leaf Xpath Queries for XML Documents International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 1, July 2012 ISSN: 2277-3754 ISO 9001:2008 Certified

5 B. Ludascher, P. Mukhopadhyay, and Y. Papakonstantinou, "A Transducer-Based XML Query Processor," Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), 2002.

6 B.M. Monjurul Alom, Frans Henskens, Michael Hannaford DRXQP: A Dynamic Relational XML Query Processor ; 2010 Seventh International Conference on Information Technology 978-0-7695-3984-3/10 $26.00 © 2010 IEEE

7 Bo Zhang , Zhihua Geng , Aoying Zhou SIMP: Efficient XML Structural Index for Multiple Query Processing The Ninth International Conference on Web-Age Information Management 978-0-7695-3185-4/08 $25.00 © 2008 IEEE DOI 10.1109/WAIM.2008.25

8 Brian F. Cooper ,Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, Moshe Shadmon A Fast Index for Semistructured Data Proceedings of the 27th VLDB conference, Roma, Italy, 2001

9 C Zhang, J Naughton, D DeWitt, Q Luo G Lohman " On Support Containment Queries in Relational Database Management Systems" in Proc. Of ACM SIGMOD , pp 425-436, 2001, pp 425-36

10 C. Chung, J. Min, and K. Shim. APEX: An Adaptive Path Index for XML Data. In proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pages 121-132, 2002.

11 C. Li, T.W. Ling, and M. Hu, "Efficient Processing of Updates in Dynamic XML Data," Proc. Int'l Conf. Data Eng. (ICDE), pp. 13-22, 2006

12 Changqing Li Tok and Wang Ling "QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates" CIKM'05, October 31-November 4, 2005, Bremen, Germany. ACM 1-59593-140-6/05/0010

13 Chun Zhang, Je_rey Naughton, David DeWitt, Qiong Luo , Guy Lohman On Supporting Containment Queries in Relational Database Management Systems , SIGMOD ’01 Santa Barbara, California USA Copyright 2001 ACM 089791886/ 97/05 pp 425-36

14 D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML via XML Fragments. In the proceedings of the 26th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 151-158, 2003.

15 D. Florescu and D. Kossmann, “Storing and Querying XML Data Using an RDMBS,” IEEE Data Eng. Bull., vol. 22, pp. 27-34, 1999.

16 D.D. Chamberlin, “XQuery: An XML Query Language,” IBM Systems J., vol. 41, no. 4, 2002.

17 E. Cohen, H Kaplan and T. Milo, “Labeling Dynamic XML Trees”, in Proceedings of the ACM SIGMOD-SIGACT-SIGART, 2002, pp. 272-81

18 F. Peng and S.S. Chawathe, “XPath Queries on Streaming Data,” Proc. 22nd ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’03), 2003

19 F. Peng and S.S. Chawathe, “XSQ: A Streaming XPath Engine,” ACM Trans. Database Systems, vol. 30, pp. 577-623, 2005.

20 F. Tian, D.J. DeWitt, J. Chen, and C. Zhang, “The Design and Performance Evaluation of Alternative XML Storage Strategies,” SIGMOD Record, vol. 31, no. 1, pp. 5-10, 2002.

21 H. He and J. Yang, “Multiresolution indexing of XML for frequent queries”, In Proc. of the 20th Intl. Conf. on Data Engineering, Boston, April 2004, pp. 683–694

22 H. Su, E.A. Rundensteiner, and M. Mani, “Semantic Query Optimization for XQuery over XML Streams,” Proc. 31st Int’l Conf. Very Large Data Bases (VLDB ’05), 2005.

23 H. Wang and X. Meng, “On the Sequencing of Tree Structures for XML Indexing,” Proc. 21st IEEE Int’l Conf. Data Eng. (ICDE ’05), 2005

24 H. Wang, S. Park, W. Fan, and P.S. Yu, “ViST: A Dynamic Index Method for Querying XML Data by Tree Structures,” Proc. 29th ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’03), 2003

25 H. Zhang, and F. W. Tompa, “Querying XML Documents by Dynamic Shredding,” in DocEng’04, Milwaukee: Wisconsin: USA, 2004

26 Haifeng Jiangy Hongjun Luy Wei Wangy Jerey Xu Yuz Path Materialization Revisited: An Efficient Storage Model for XML Data (AICE2000), Canberra. Conferences in Research and Practice in Information Technology, Vol. 1

27 Hye-Kyeong Ko and SangKeun Lee A Binary String Approach for Update in Dynamic Ordered XML Data IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 22, NO. 4, APRIL 2010

28 Ibrahim Dweib , Ayman Awadi and Joan Lu MAXDOR: Mapping XML Document into Relational Database The Open Information Systems Journal, 2009, 3, 108-122

29 J. Chu-Carroll, J. Prager, K. Czuba, D. Ferrucci, and P. Duboue. Semantic Search via XML Fragments: A High-Precision Approach to IR. In the proceedings of the 29th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 445-452, 2006

30 J. Kamps, M. Marx, M. Rijke, and B. Sigurbjörnsson. Best-Match Querying from Document-Centric XML. In proceedings of the 7th International Workshop on the Web and Databases (with ACM SIGMOD/PODS 2004), Paris, France, pages 55-60, 2004.

31 J. Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In CIKM, pages 533–542, 2004.

32 J. Lu, T. W. Ling, C. Y. Chan, and T. Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In Proceedings of VLDB,pages 193–204, 2005.

33 J. McHugh and J. Widom, "Query Optimization for XML," in VLDB Edinburgh, Scotland, 1999.

34 J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. F. Naughton, and I. Tatarinov. A General Techniques for Querying XML Documents using a Relational Database System. SIGMOD Record, 30(3):20{26,2001.

35 J. Shanmugasundaram,H. Gang, K. Tufte, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and opportunities. In proceedings of the 25th international conference on Very Large Data Bases, Edinburgh, Scotland, pages 302-314, 1999

36 J.-K. Min, C.-H. Lee, and C.-W. Chung, “XTRON: An XML data management system using relational database,” Information and Software Technology, vol. 50, pp. 462-479, 2008.

37 K. Beyer, R. J. Cochrane, V. Josifovski, J. Kleewein, G. Lapis, G. Lohman, B. Lyle, F. O¨ zcan, H. Pirahesh, N. Seemann, T. Truong, B. Van der Linden, B. Vickery, and C. Zhang. System RX: One part relational, one part XML n SIGMOD, 2005.

38 L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In the proceedings of 2003 ACM SIGMOD international conference on Management of data, California, USA, pages 16-27, 2003.

39 Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In CIKM, pages 533–542, 2004.

40 M Al-Nadawi, Dr Siobhan North, Dr Barry Eaglestone Indexing XML Databases: "Classifications, Problems Identification and a New Approach" , The University of Sheffield Department of Computer Science technical report 15th November, 2007

41 M. Mani and D. Lee. XML to Relational Conversion using Theory of Regular Tree Grammars. In Proceedings of the 28th VLDB Conference, Hong Kong, China, 2000.

42 Matthias Nicola , Bert van der Linden "Native XML Support in DB2 Univeral Database " Proceedings of the 31st VLDB Conference , Trondheim , Norway,2005

43 Michael Kay "Ten Reason Why Saxon XQuery is Fast" Bulletine of the IEEE Computer Scociety Technical Committee on Data Engineering 2008

44 Mohammed Al-Badawi, Dr. Siobhán North, Dr. Barry Eaglestone , research memorandum Indexing XML Databases: Classifications, Problems Identification and a New Approach, 15th November, 2007 in The University of Sheffield Department of Computer Science

45 Mustafa Atay , Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi Efficient schema-based XML-to-Relational data mapping www.elsevier.com/locate/infosys

46 Mustafa Atay, Yezhou Sun, Dapeng Liu, Shiyong Lu, Farshad Fotouhi MAPPING XML DATA TO RELATIONAL DATA: A DOM-BASED APPROACH arXiv:1010.1746v1 [cs.DB] 8 Oct 2010

47 N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," Proc. 21st ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '02), 2002

48 N. Fuhr, and K. Groβjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In the Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, Louisiana, USA, pages 172-180, 2001.

49 N. Zhang, V. Kacholia, and M.T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML,"Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004.

50 Nicolas Bruno, Luis Gravano, Nick Koudas, Divesh Srivastava Navigation- vs. Index-Based XML Multi-Query Processing Proceedings of the 19th International Conference on Data Engineering (ICDE'03) 1063-6382/03 $ 17.00 © 2003 IEEE

51 Ning Zhang, Shishir K. Agrawal, M. Tamer ¨Ozsu BlossomTree: Evaluating XPaths in FLWOR Expressions" Technical Report CS-2004-58 November 2004

52 Online Computer Library Center, 'Introduction to the Dewey Decimal Classification", http://www.oclc.org/dewey/

53 P. O'Neil, E. O'Neils , S. Pal , L . Cseri, G Schaller and N. Westbury, "ORDPATHS:Insert-Friendly XML Nodes Labels", in Proceedings of the AMC SIGMOD, 2004, pp 903-8

54 P. Rao and B. Moon, "PRIX: Indexing and Querying XML Using Prufer Sequences," Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004

55 P.F. Dietz, "Maintaining Order in a linked List" , in Proceedings of the ACM Symposium of Theory of Computing, 1982, pp 122-7

56 Q. Chen, A. Lim, and K. Ong. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data. In proceedings of the 2003 ACM SIGMOD international conference on Management of data, CA, USA, pages 134-144, 2003.

57 Q.Li and B. Moon , " Indexing and Querying XML Data for Regular Path Expressions", in Proceedings of the VLDB, 2001 ,pp 361-70

58 R. Goldman, and J. Widom, "Data Guides : Enabling Query Formulation and Optimization in Semistructured Databases", Proceedings of the VLDB, 1997, pp. 436-445.

59 R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In proceedings of the 18th international conference on Data Engineering, Washington DC, USA, pages 129-141, 2002.

60 S. A1-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of the IEEE International Conference on Data Engineering, 2002.

61 S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In Proceedings of the 6th annual ACM international information and data management, Washington DC, USA, pages 31-38, 2004workshop on Web

62 S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

63 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, "Indexing XML Data Stored in a Relational Database," Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04), 2004.

64 Samini Subramaniam , Su-Cheng Haw , Poo Kuan Hoong Mapping and Labeling XML Data For Dynamic Update, Second International Conference on Computer Research and Development , 978-0-7695-4043-6/10 $26.00 © 2010 IEEE DOI 10.1109/ICCRD.2010.173

65 Songting Chen , HuaGang Li , Junichi Tatemura ,WangPin Hsiung , Divyakant Agrawal , K. Selc¸uk Candan Twig2Stack: Bottomup Processing of Generalized TreePattern Queries over XML Documents VLDB '06, September 1215, 2006, Seoul, Korea.

66 T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In SIGMOD, pages 455–466, 2005.

67 T. Milo, and D. Suciu, "Index Structures for Path Expression", Proceedings of the ICDT, 1999, pp. 277-295.

68 T. Milo, and D. Suciu. Index Structures for Path Expressions. In proceedings of the 7th international conference on Data Technology, Volume 1540/1998, Jerusalem, pages 277-295, 1999

69 W.E. Kimber , "HyTime and SGML: Understanding the HyTime HYQ Query Language" ,http://ftp2.de.freebsd.org/pub/sgml/ifi.uio.no/HyTime/HyQ. Kimber

70 W3C Website. XML Query Language (XQL). Online:http://www.w3.org/TandS/QL/QL98/pp/xql.html, Access on: 07/10/2007

71 Wang Lian, Nikos Mamoulis, David Wai-lok Cheung, Member, IEEE Computer Society, and S.M. Yiu Indexing Useful Structural Patterns for XML Query Processing IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 7, JULY 2005

72 Whitmer R., DOM Working Group : Document Object Model (DOM) Level 3 XPATH Specification [2003]

73 X. Li and G. Agrawal, "Efficient Evaluation of XQuery over Streaming Data," Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05), 2005

74 Xiaodong Wu et "A Prime Number Labeling Scheme for Dynamic Order XML Trees" in 20th International Conference on Data Engineering (ICDE04) 2004 IEEE pp 66-78

75 Xiaojie Yuan , Xiangyu Hu, Dongxing Wu, Haiwei Zhang+ and Xin Lian' XML Data Storage and Query Optimization in Relational Database by XPath Processing Model' , © 2013 ACADEMY PUBLISHER

76 Y. Chen, S.B Davidson, and Y. Zheng "BLAS: An Efficient XPath Processing System", in Proceeding of the ACM SIGMOD, 2004, pp47-58

77 YANLEI DIAO, MICHAEL J. FRANKLIN, HAO ZHANG,PETER M. FISCHER Path Sharing and Predicate Evaluation for High-Performance XML Filtering published in ACM TODS, December 2003

78 Yi Chen , Susan B. Davidson, Yifeng Zheng "A bi-labeling based XPath processing system" Information Systems 35, 2010, doi: 10.1016/j.is.2009.05. ACM

79 Yi Jiang , Zhi min Zeng, Dong Zhan Zhang "An Efficient Encoding and Labeling Based Upon Continued Fraction for Dynamic XML Data", in World Congress on Software Engineering , IEEE DOI 10.1109/WCSE. 2009.150, PP 324-328

80 Yingjie Fan , Shuyun Wang , Chenghong Zhang Haibing Ma4, Yunfa Hu An Efficient Structural Index for Branching Path Queries The Ninth International Conference on Web-Age Information Management , IEEE 2008 pp119-126

81 YoshiKawa, M. and Amagasa, T. (2001). XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology.

82 Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos. From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. In Proceedings of VLDB, pages 237–248, 2003

83 Z. Liu, and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In proceedings of the 2007 ACM SIGMOD international conference on Management of Data, Beijing, China, pages 329-340, 2007

84 Zhewei Jiang, Cheng Luo, Wen-Chi Hou, Qiang Zhu, and Dunren Che Efficient Processing of XML Twig Pattern : A Novel One-Phase Holistic Solution

Chapter 4

Research Methodology

4.1 Introduction

Over the past decade, the expansion of internet and complex web application such as ecommerce, e-learning require data storage of different format. Since February 1998, the eXtensible Mark-up Language (XML) has become the standard medium for data representation and exchange over the Web [2]. Due to its simplicity and user friendliness, use of XML technology is increased day by day.

It is used as International Standard for document exchange. It is vendor independent and not tied up with any company or with any application. The name 'eXtensible' suggests that it is not restricted with fixed tag like HTML. The company or developer adds their own tag as per their need. Another important characteristic of XML is reusability, instead of creating data from scratch; XML reuses the exiting data and build the new structure from it. Another important characteristic is compatibility, it is easy to apply different style sheets to an XML document to manipulate content for different audiences or output to different documents.

Its utilization is not only restricted to document representation and manipulation but also used as functional language as well as data manipulation language. XML is capable of storing different company documents like product manual, company work process etc are created by XML. It works as functional or programming language also. XML also manipulate data and works as database management system. For example the web content like user navigation, maintenance of user profiles etc are managed by XML. These data management is nothing but the database management. Once the XML is used as database management system, it must follow the features of DBMS like query optimization, indexing, transaction management etc. Thus, a proper mechanism is needed for this management. This thesis addresses the query optimization and indexing of XML. Hence, the main objective of the thesis is determining effective storage management and index structure for XML data.

4.2 Importance of the Study:-

XML is an important component of web application. It defines the standard framework for e-business so that business communication is easy. For example, the e-commerce web site promotes their products for sell. Web site extracts the different product from different company database and preset it for sell. Their data format is different so it is very difficult to merge this information for further processing. But this is achieved through XML. XML collect these different formats and combined together and display the information into a web enable format. XML has another property of extensibility so the new records can inserted as per the need. Additional insertion cost or rebuilding cost is avoided by the XML technology. One of the important examples is EDI. The EDI is Electronic data interchange community, which was used by the large company for selling and buying their product. This service was not affordable by the small company due to high paying cost and as large company have major stock holder of it hence they have their say for changing their standers which also affect small company business. These standards are replaced by XML. XML based e-commerce is penetrate down even to small companies. Most of the people agree that XML is easy to work with more extensible

Now days, the business are need to communicate each other. Therefore the data is transfer from not only one web site to another website but also from one system to another system, from one business to each other e.g. Enterprise Application Integration. This integration application is within the company itself means accounting system is integrated with inventory system, inventory is associated with billing system. These communication is performed the XML protocol only.

XML is used not only data representation but also used for document representation like product manuals are represented using XML. It also used for metadata representation. E.g. system configuration data, the workflow procedure in the company like production cycle of the product is explain in XML. It represents serializing complex data into non-relational data. It integrates with other system.

In software application contains structure and hierarchical information. These information is very difficult to represent by any RDBMS system e.g. users preference for website. Their likes, their

navigational flow etc. cannot store by RDBMS system but XML is capable of storing this information. As mention above workflow of any production system is unable to store by RDBMS system but it is easily stored in XML technology.

Other important characteristic, of XML is multi-language support. The different languages systems can communicate to each other .e.g. 'Chines' language website can communicate to any 'Marathi language 'web' site. The translation between these two sites is performed with XML support.

XML is also used as content management system. Content is unit of digital information. It can be text, images, graphics, sound etc. that is likely managed in an electronic format. Content Management is effectively the management of the content. These content re-use saves companies money and makes authors more efficient. XML supports and promotes these sorts of processes by being flexible and modular. XML allows you to separate content from format. The formatting of the XML document is inside a separate style sheet. This separation allows easily maintaining and updating formatting as needs change. It is easy to maintain a consistent style for all documents when the content is separate from the formatting.

As internet evolved and it's potential for e-business is cleared then W3C developed XML language for performing the data exchange between e-businesses. There is a strong momentum for XML to be an important "next-generation" language for E-commerce. And several companies have been started to developed XML based solutions. All above mention characteristics suggest the significance of XML in software.

4.3 Scope of the study:-

As the XML utilization are increased. It is used by all three types of communities-'Document type of Community', 'Data type of Community' and 'Functional communities' are interested to use XML. The 'Document Community' is using XML for document storage. 'Data Community' is using XML for data storage and database application. The functional Community is using XML for programming purpose. As need of communities are different, the different data retrieval or different types of queries are needed. For 'Document Community required IR techniques for retrieving data. Functional communities are using all programming constrains and data communities are using data bases queries.

From 1990, onwards different query languages were developed. The different query language and their characteristics and advantages are explained into chapter2 page no28. The XQuery is languages which support all types of community. Therefore the research scope is limited to XQuery language.

This research limits its scope to 'Data Community'. The 'Data Community' is interested for database functionality. The DBMS functionality is 'Query Processing ', 'Transaction Management', 'Database Security' etc. This research scope limited to the query optimization techniques. In query optimization 'indexing' is an important component for effective query evaluation. The research scope is limited to indexing techniques. Thus the research scope is identification of effective indexing techniques for XML query execution and identification of a proper XML storage technique.

4.4 Problem Statement of the Research

Considering the above issues, the researcher defined the problem statement of the research. :-

Designing a flexible and efficient data storage structural for XML file and a query optimization algorithm on it. This query optimization technique includes designing of efficient index structure. This indexing structure is such that it helps in execution of the all types of query.

This new framework includes the following features:-

- Assigning a unique identity to each XML node.
- The XML node identifier represents P-C, A-D, and sibling relationship. So, that is can answer all types of query like P-C, A-D queries, Partial match and Content- query by single lookup.
- This framework must avoid redundant traversal of XML tree for answering the all types of XML query.
- This index structure must reduce disk I/O cost and provide high-quality query performance resul**t.**

4.5 Objective of the Research

The objective of the research is:-

- Recognize the importance of XML in software development.
- Identification of effective storage method for XML storage and query processing.
- Designing a new index structure for XML file so that it reduced redundant traversal of XML file.
- Designing an efficient algorithm for query processing.

4.6 Statement of Hypothesis:-

1 Instead of using XML in isolation, better results can be obtained by using combo model which includes RDBMS and HTML.
2 Existing model of XML does have some weak points.
3 XML-RDBMS storage is always better than XML-Native storage and XML-Navigational storage.
4 There is no difference between query processing cost of XML databases and RDBMS database.
5 XML index size is directly proportional to query processing cost and effective organization and data retrieval of XML data file.

H_1 Instead of using XML in isolation better results can be obtained by using combo model which includes RDBMS and HTML.

XML is an important component of today's technology because XML has extensible, self-descriptive and application independent features. The extensibility means the data can be added or deleted from files easily. XML cannot consider backward compatibility issues so someone can add the new node into XML file; without any modification of previous data. XML is self-defining; it describes the structure and meaning of the data content that it holds. The application independence means XML can be used to exchange data among the different computer systems irrespective of the platforms on which those systems were running on. All these features suggest that XML is used in today's e-business as well as in different e-documents easily. This

hypothesis studies the XML utilization in different application and how it is used in different e-business and XML is only sufficient for performing all business transaction as well as representation

To study this hypothesis, the case study method was followed. Different case study was selected from different company portal and identified their bottle neck problem in the business and their solution towards that problem. The study also performed on sample data collected from different software users.

H_2 Existing model of XML does have some weak points

The literature suggests XML utilization in the different business. It is specially used for data transformation, data communication and data representation. The subsequent list mentions the utilization of XML:-

1. The application required to store menu data or some combo box information.
2. It is used for developing the database driven websites.
3. In image gallery application, XML is used as the data file (xml) for storing the names and location of the images.
4. For shopping application, it is used to store the product details and a purchase detail of the customer as well as it is also used for performing e-payment.
5. In travel applications, XML is used as a gateway for different booking transaction.
6. The web services such as Weather services, Currency rates service etc. are using the XML language as a programming language as well as database for supplying information to process for further transaction.
7. The developers use the XML data files to generate the dynamic content by applying different Style sheets.
8. XML is also used to develop the content management systems.
9. Many companies are using XML files for writing the documents. Then program like DocBook can be used to generate the required documents.
10. Many software development frameworks are using XML files to store the configuration data for the application. For example, JPA uses persistence.xml file for configuring the JPA environment.
11. Many companies are using XML for electronic data exchanges for transferring their business records from one web site to another.

12. XML is used for transferring data information between two web sites.

13. XML is also used very easily in the application where database and flat files are difficult.

Since XML data is used for different purposes, there may be additional support required for manipulation, organization of data. This hypothesis checked the same. For the study, researchers select the different website and study how these companies store this information for representation of the data.

H_3 XML-RDBMS storage is always better than XML-Native storage and XML-Navigational storage.

XML is used by different communities: database community, functional community and document community. The document communities use XML for storing and retrieving data information. The programming communities are using XML for programming purpose. The database communities are using XML for the database.

To satisfy this requirement, XML needs to store the application data effectively and efficiently. In literature, three techniques are used for it. They are: XML-RDBMS and XML native storage and XML navigational storage.

In XML-RDBMS storage, XML data is converted into relational table in such a way that the storage maintains the P-C relationship and A-D relationship. The major advantage of this storage is it can use existing RDBMS like query optimization, indexing, data security is applicable to it. In this storage, XML queries are first converted into SQL queries, and then existing query optimization technique is used for query evaluation.

In XML-Native database storage, a specialized storage technique is derived for XML storage. It stores XML data by using different data structure like link list, stack, and multiple links lists etc. The additional data conversion cost is not required; it automatically maintains P-C, A-D and sibling relationship. All database features like normalization, query optimizations are needed to derive from the scratch. XML queries are also stored using different data structure and specialized query optimization algorithm is executed on it.

In XML navigational storage, XML data file is arrived as data stream without associated index; entire XML tree needs to traverse node by node in depth wise manner. XML queries are also traversed as a data stream. When desired match is found, the results are displayed.

This hypothesis tests the best XML storage methodology. The investigation is performed on query execution speed, index techniques and disk utilization cost. The three different products: Saxon, IBM DB2 and eXit are used for study.

H_4 There is no difference between query processing cost of XML databases and RDBMS database

The main goal of the query execution is retrieval of desired data with minimum time. For this purpose, proper query optimization plan is required. The optimization plan is nothing but writing the blueprint of physical and logical query execution plan for effective retrievals of data from the query and query optimizer selects the most efficient plan from it. The query execution plan calculates the query cost. The query costing parameters are 'Data Organization', 'Query Execution time', 'Index size'. This hypothesis checks the cost of query processing in RDBMS system as well as in XML system. It also studies the different impact of different factors on query execution.

The experiment is performed on IBM DB2 and SAXON for XML query processing. In IBM DB2, the data is stored into RDBMS formant. In SAXON, the XML file is text file and java Saxon package executed for query execution.

H_5 XML index size is directly proportional to data organization of XML file and query processing cost.

In query execution, indexing plays an important role. Indexing is nothing but an organization of data in such a way that the searching value can be retrieved data effectively with minimum time. These indexes are created on some data filed. These data filed is nothing but search key value in the query.

XML is a hierarchical order, recursive nature of data. The data searching and data retrieval is very difficult. These operations can perform effectively, if the data is organized in effective manner, this hypothesis tests how the XML data is organized and studies the different XML-indexing techniques on it. This hypothesis also checks the impact of index size on XML storage and query execution.

To test this hypothesis, the different index types are selected (node index, path index and combination of node index + path index). Researchers perform the experiment with different size of XML files i.e. XML file contains 10 records, 100 records and 1000 records and same index structure. It measures performance i.e. query execution time.

4.7 Research Methodology :-

This research study is limited to identification and designing a framework for the best XML storage and data retrieval technique. It utilizes both primary and secondary data. The primary data is collected through questionnaire, experiment, case study analysis and content analysis. . The scope of research is limited; the survey is undertaken by obtaining a purposive and quota sample. The description of the research methodology required for the process of obtaining a sample as well as the nature and size of sample should be adequately explained. Probability quota, stratified sampling technique were used. The selection of respondents was involved based on the important characteristics under study such as XML known users, software developers and software tester etc

4.7.1 Secondary Data:-

XML satisfied the need of all three communities: document community, data community and functional community. It is used for data representation, data community is used XML for data storage and query evaluation. For functional community it is used as programming language. Hence all community has interest in XML development. All these communities had performed extensive research in their own area. One of the common problems between them is designing of a query optimization techniques for data retrieval from XML database. Therefore, this thesis is focus on indexing and query processing techniques for data retrieval from XML database. The secondary data was collected from different website, different articles, periodical and the different research articles. The secondary data is collected analysis.

4.7.2 Primary Data :-

This research is for designing new indexing and query processing technique for XML. For our study, the researchers selected following methods for data collection:-

(1) Questionnaire: - XML is an important constituent in web software development. This method was used for collecting the opinion of different software experts.

(2) Experimental Research: - This method was adopted for testing the XML storage and query evolution techniques.

(3) Case Study Method: - This is one of the important methods for research. The data was collected from the different company cases. The analysis was done on identification of the problem, the XML solution towards that problem.

(4) Content Analysis: - It consists of analyzing the contents of documentary material. In this research, XML utilization in different business was analyzed by using this method.

The primary data collected by the researcher is explained in the following manner.

4.7.3 Questionnaire: -

The method was used for collection of different data from different software users. The main focus from this method was analyzing the XML utilization from the different user's perspective. Small questionnaire was set and distributed to software user. The samples were selected as Industry users and non industry users. Non industry users are computer teacher. From the industry the experts were classed as 'Software developer', 'Software tester', 'other category '(they may be in maintenance of product or on bench people). The sample size is as follows:-

Table 4.1 Respondent Sample

	Software Role	No of sample Collected
1	Computer Teacher	40
2	Software Development	470
3	Software Testing	580
4	Support Team	112
5	Other in software Company	136
	Total	**1338**

4.7.4 Experimental Research: -

The laboratory research was conducted for studying XML storage and indexing method.

(1) Software used for experiment. :- Table 4.2list the software used for experiment

Table 4.2 Software List for Experiment

Sr No	Software used	XML type Storage	Index Strategy
1	IBM DB2	Native Storage- Coarse-Grained (special data type .xml is used for storing XML file)	Path Index
2	IBM DB2	Fine-Grained (RDBMS Storage)	B+ index is used
3	SAXON	Native Storage – Navigation approach for storage	Not possible to create Index

(2) Data Set for experiment:-

XML data can store into three types of file: Document-centric, Data-centric, Mix-mode (combination of Data centric and Document centric application). In Document-centric XML documents store text information e.g. user's manuals, static web pages, and marketing flyers or brochures. They are characterized by loose, irregular structure. Their structural sequence is crucial. For storing this type of data, content management system tools which are used commercially. The available tools are SyCOMAX, Content@ and Frontier, etc. Data-centric documents are used for data exchange and transport medium. They are usually highly structured and marked up with XML tags. Data-oriented paradigm has focused only on data stored information e.g. web service return; current stock price, stock information, Railway reservation etc.Third XML storage is 'Mix-mode' which is a combination of data-centric and document-centric storage. Mix-mode data is preferred to store into native database [3]. For the experiment, the following types of files were considered: -

Table 4.3: Input Data Files for Experiment

	File Type	**Input Data information**	**Depth of File**	**Breadth of File**
Data Set-I	Data Centric File	Set1	Constant	Varying
		Set2	Varying	Varying
Data Set-II	Mix Mode File	Set3	Constant	Varying
		Set4	Varying	Varying

1. Data Centric File :- Two files are considered
    a. CDINFO.XML:- Depth of the file is four and six files with 10 nodes, 100 nodes, 1000 nodes, 10,000 nodes, 50,000 nodes and 100000 nodes.
    b. WHETHER.XML:- Depth of the file is ten and six files with 10 nodes, 100 nodes, 1000 nodes, 10,000 nodes, 50,000 nodes and 100000 nodes.
2. Mix-mode File: - These files are combination of data centric as well as document centric.
    a. Book.XML: - Depth of the file is four and six files with 10 nodes, 100 nodes, 1000 nodes, 10,000 nodes, 50,000 nodes and 100000 nodes.
    b. Chap.XML:- Depth of the file is ten and six files with 10 nodes, 100 nodes, 1000 nodes, 10,000 nodes, 50,000 nodes and 100000 nodes.

The detail Storage Structure of above files is listed into following table 4.4 and detail file structure is explained into annexure –II.

Table 4.4 XML File Structure

Dataset	Number of node at 1st Level (Assume root level no 0)	Max fan-out (maximum node at a particular level)	Depth of the file	Total number of nodes
SET 1.1 :- Data Centric File : CD.XML				
SET 1.1.1	10 - nodes	60	4	197
SET 1.1.2	100 - nodes	600	4	703
SET 1.1.3	1000 - nodes	6000	4	6301
SET 1.1.4	10000 - nodes	59952	4	69944
SET 1.1.5	50000 - nodes	300000	4	350140
SET 1.1.6	100000 – nodes	599994	4	699994
SET 1.2 Data Centric File :- Whether .XML				
SET 1.2.1	10 - nodes	150	12	483
SET 1.2.2	100 - nodes	1575	12	5041
SET 1.2.3	1000 - nodes	15075	12	48241
SET 1.2.4	10000 – nodes	150000	12	471006
SET 1.2.5	50000 - nodes	749880	12	2349530
SET 1.2.6	100000 – nodes	1500045	12	5000145
SET 2.1 Mix mode File :- Chp.xml				
SET 2.1.1	10 - nodes	40	5	71
SET 2.1.2	100 - nodes	402	5	699
SET 3.1.3	1000 - nodes	3920	5	6861
SET 3.1.4	10000 - nodes	39200	5	68601
SET 3.1.5	50000 - nodes	196000	5	343001
SET 3.1.6	100000 - nodes	392000	5	686001
SET 2.1 Mix mode File :- MXDB.xml				
SET 2.2.1	10 - nodes	40	10	181

SET 2.2.2	100 - nodes	380	10	1711
SET 2.2.3	1000 - nodes	3800	10	17101
SET 2.2.4	10,000 - nodes	38000	10	171001
SET 2.2.5	50000 - nodes	190000	10	855001
SET 2.2.6	100000 – nodes	380000	10	1710001

(3) Query Set for Experiment:-

The XML queries are classified as: containment queries, order queries and node queries or content queries [19]. The query syntax is classified into two types: (1) Tree Structure queries (2) Containment Query. (3) Twigs Queries (4) Containment queries.

Table 4.5 Types of XML Query

Sr No	Query Type	
1	**Tree Traversal**	Following
		Following – sibling
		Preceding
		Preceding - sibling
2	**Containment Query**	Child axis query
		Self axis query
		Descendent axis query
3	**Twig**	FLOWR- Clause - return small XML tree
4	**Content matching query**	FLOWR- Clause - match the content of the query

The following eighteen queries are selected for the experiments which satisfy all above mentioned requirements. The Syntax of the queries is listed into Table 4.6 :-

Table 4.6 Query Set for Experiment

Q1	select a complete tree	for $c in doc("CD.XML")/CATALOG/CD return $c
Q2	Select a particular tag of tree	for $c1 in doc("CD.XML")/CATALOG/CD return <item> {$c1/TITLE}
Q3	Following	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/following::*}
Q4	Following-sibling Axis	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/following-sibling::*}
Q5	Following-sibling Axis	for $c1 in doc("CD.XML")/CATALOG/CD[1] return {$c1/following-sibling::*}
Q6	Preceding Axis	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/preceding::*}
Q7	Preceding-sibling Axis	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/preceding-sibling::*}
Q8	Preceding-sibling Axis	for $c1 in doc("CD.XML")/CATALOG/CD[1] return{$c1/preceding-sibling::*}
Q9	Self- axis	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/self::node()}
Q10	Descendent Axis	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/descendant::*}
Q11	Child node	for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/child::*}
Q12	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR=1985 return $c1
Q13	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR>1985 return $c1
Q14	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR<1985 return $c1
Q15	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR<1985 return $c1/TITLE
Q16	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR=1985 return $c1/TITLE
Q17	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR=1985 return $c1/YEAR
Q18	FLOWR	for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR<1985 return $c1/YEAR

(4) Software and Hardware Requirement:-

- Hardware Configuration :- Intel Pentium (R) Dual Core CPU with 2.69GHz CPU, 2.79 GB RAM and 40 GB hard disk and Windows Operating System.
- Software :- For relational database implementation researchers use IBM DB2 Pure XML 9.0, Java for PESS algorithm and c++ language for XML node_id creation
- Parameter Set UP:-
  - Navigation Approach :SAXON
    - JDK1.6.0
    - Heap Size
    - Saxon9he
  - For DB2 Pure XML researchers set the following parameters:
    - Page Size : 4 KB
    - Buffer Pooling
    - DB2 Pure XML: - The indexes are set on XML files 'path encoding' sequence and secondary index on node_id values.
  - Researchers select JDK 1.6.0 and visual C++ for the experiment. Additional parameter setting is not required for it. Only researchers change Java Heap memory for reading XML files.

(5) Measurement Parameter :-

The Measurement parameters are:-

1) **XML Storage Measurement**: - XML storage is measure on maintains of the structural relationships between XML nodes. XML data is represented into tree structure format, therefore the maintenance of XML structure and actual stored data information are two important components of query execution. Under this section the researcher measure the two parameters (1) XML storage structure. (2) Data retrieval technique.
2) **XML query measurement** :- As per the [6] [19] [7] proposal's guidelines the following parameters are listed for query evaluation:-
   - Query evaluation speed: - This criterion refers to the time required to evaluate a query in comparison to different XML storage and different query traversal.
   - Disk traffic: -It refers to the amount of data that an index requires to evaluate a query, i.e. the size of pages which it fetches from disk.

- Disk space:-The amount of disk space required for storing the XML data.
- Scalability: - The size of documents stored in databases constantly grows and the costs for storage space increases. Index structures therefore should not only to be able to handle large amount of data, but also to scale up the small documents.
- Index updates: - Refers to the time which it takes to update an index when updating a document. Thereby, updates include insertion, deletions and modifications of documents.

(6) Algorithms for Measurement :-

a. Measurement variables for IBM DB2:-

1) **Query Evaluation Speed: -** Researchers have written following program for query evaluation time measurement. For each query, it is executed 100 times and takes the average of response time.

Algorithm 4.1:- Query Measurement Time (IBM DB2)
1. Load Data base driver 2. Establish the Connection 3. Create Query statement 4. Read the start time of the query 5. Read the end time of the query 6. Repeat the steps 4 and step 5 ten times 7. Take the average of query execution time 8. Repeat the process for each query.

Psudocode 4.1 :- Query Execution Time Measurement

```
class cdq1
{  public static void main(String argv[])
  {       //  load driver
       // create connection
      // create  statement object
     // create  query statement
      for(int k = 0; k < n; k++)
        {  t1=(new java.sql.Timestamp(System.currentTimeMillis()));
          String query="XQUERY for $y in db2-
          fn:xmlcolumn('CD.CDINFO')/CATALOG/CD    where  $y/YEAR =1985 return
          $y/YEAR";
            ResultSet rs = stmt.executeQuery(query);
                // retrieve and display the result from the xquery
              while (rs.next())
                        String data=rs.getString(1);
                // Close the result set
                 rs.close();
                  // Close the statement object
                   stmt.close();
             t2=(new java.sql.Timestamp(System.currentTimeMillis()));
            total[k]  = t2 - t1;
                  }//end of for statement
          system.out.println(total(k) /k/);
  } // end of main
 }//end of class cdq1
```

2) Disk Space Measurement :-

- Disk space measurement in Navigation Approach :
  - Size of text file
  - Memory required for constructing a node structure.
- Disk space measurement in RDBMS Approach
  - Read Table space
  - Read Page Size
  - Query on Estimate Size of database provide statistic of table
  - Read database size as well as index size
- Following formula is used for disk space measurement :-

Disk size of the file = (total number of rows * average row length) * (total number of Pages * Page Size)

3) **Disk traffic Measurement** : -

The query execution plan provides the 'cardinality', 'selectivity', and 'total buffer pool pages' used in query evaluation, CPU cost, I/O instruction cost. The disk traffic is nothing but total number of buffer pages and the I/O cost required for query execution for the particular cardinality, and selectivity with the specific CPU cost.

4) **Scalability Measurement: -**

For the experiment we used total five file with growing database i.e. from 10 records till 100000 records. Index is also defined for this entire file. Hence the parameter check index structure handles large amount of data.

5) **Index updates**:-

The new record is added, deleted or modified into the file, the index structure is capable to perform all these changes into index file.

b. Measurement Variables for SAXON

1. **SAXON Query Execution Speed: -** The algorithm is same as mentioned in psudocode 4.2. The time is measured by using java time library

Psudocode 4.2 :- Query Execution Time Measurement on SAXON

```
class cdq1
  {  public static void main(String argv[])
    { for (int k = 0; k < 10 ; k++)
       { t1=(new java.sql.Timestamp(System.currentTimeMillis()));
         java  net.sf.saxon.Query  [options]  query  filename
         t2=(new java.sql.Timestamp(System.currentTimeMillis()));
         total[k]  = t2 - t1;
         }//end of for statement
        system.out.println(total(k) /k/);
      } // end of main
   }// end of  class cdq1
```

2. SAXON : Disk Traffic Measurement

- Navigation Approach :SAXON-
  - Saxon is not a database product. It cannot afford luxury of data storage and memory residential indexing.
  - XML file is read by SAX parser and Nodes are created on fly.
  - Total number of node reading is depending upon java heap size
  - I/O cost or Disk traffic depends upon heap. e.g. If heap memory is 25 bytes and file size 100 bytes file is sliced into four and I/O cost is four .

3. SAXON: Disk space measurement

Saxon is not the database product hence the disk space is calculated by amount of data –space required for storing XML file i.e. text file. SAXON is not database product hence remaining two parameters are not needed for testing it.

4.8 Case Study Method:-

Managing data into different business is a big challenge in today's e-business. The business data interact to the different organization not only for information exchange but also reuse of the existing data in the organization in different format. In the data information exchange, the data incompatibility problem is arising. This incompatibility problem was solved by XML. It is a structure document, its aim is to separate presentation, structure and meaning from actual document. Therefore the documents can easily understand. It is easily interchange between businesses. It can reuse in multiple situation instead of creating it from scratch.

To test these characteristics researchers study the different cases of different company. These company adopted XML techniques in their organization. Researcher studies it with reference to following points - the existing business problems and their XML solution. Why companies need to use the XML technology? Study the XML utilization for different purpose. Once the data is shifted then does sold technology is needed.

4.9 Content Analysis:-

Business has urged to communicate each other not only for email, fax but also for the data entry, data interpretation and electronic transactions. Previously, EDI is used for it. But EDI depends on proprietary networks and X12 document descriptions. It is third party proprietary. Large company feels free to modify it as they pleased. This EDI is replacing by XML. An XML document can carry the description of its contents in document header section. The receiving system, read the header and handles the data, it is not depends upon the third party. Different organizations are involved in writing the 'vocabulary 'and designing 'framework' for standardization. Different web services have been designed using XML data. For the study, the following business framework was selected:-

- E-commerce :- XML used for e-transaction, electronic data exchange,
- Financial: - e.g. Financial report creation, electronic exchange of securities transactions, used for representation of structured financial products.
- Government :- Tax XML includes a vocabulary of terms, a repository of artifacts including XML templates, documents exchanged for tax compliance

- Astronomy e.g. astronomical data, such as images, spectra, tables, and sky atlases.
- Food e.g. Meat & Poultry Data Standards Organization
- Research Lab :- XML data for representation of research content
- Legal e.g. used for Electronic Court Filing, e-Contracts, e-Notary

The XML Business frameworks with examples are listed in table 4.7:-

Table 4.7 XML Business framework example

Sr No	**Business Domain**	**Examples of XML utilization**
1	E-commerce	Online Shopping
		Electronic Payment s
		Online Auctions
		Internet Banking
		Online Ticket Booking
2	Financial	Financial company data representation
3	Government	Government Policy data representation
4	Legal	XML is used for representation of legal matter
5	Research	Utilization of XML in research ,like Space science (NASA), Basic science document representation (Math , Physics, Chemistry etc)
6	Computer science	XML data utilization in Computer Science
7	Publication	XML utilization in Publication
8	Food and Agricultural	XML utilization in food Industry and agricultural domain
9	Entertainment	XML used in entertainment
10	Engineering	Utilization of XML in Engineering
11	Pharmacy	Pharmacy company
12	Heath care	Utilization of XML in healthcare industry
13	Other	Utilization of XML in space , Artificial Intelligence, Music etc

4.9.1 Hypothesis testing techniques :-

The researcher has collected primary data in the field work and through the laboratory experiment. The said data is properly analyzed with statistical techniques. In addition to this, necessary graph and charts have also been prepared to support the analysis of the data wherever necessary.

4.10 Summary and Conclusion

XML utilization is increased day by day and now it becomes the standard media for data exchange on the internet. One of the dimensions of utilization it is a database product. Hence effective data storage mechanism and indexing strategy is required. Due to XML characteristics it is retrench challenge for designing perfect indexing schema for effective storage.

This chapter explores the research objective and hypothesis of the research. It also discussed the prerequisites of the experiment. The input, Hardware, Software, Platform used for experiment as well as software used for experiment.

References:-

1 Abraham Silberschatz, Henry Korth, S. Sudarshan "Database System Concept" McGraw-Hill Education 27-Jan-2010

2 Barbara Catania and Anna Maddalena, Athena Vakali " XML Document Indexes: A Classification SEPTEMBER • OCTOBER 2005 Published by the IEEE Computer Society 1089-7801/05 © 2005 IEEE IEEE INTERNET COMPUTING

3 Byron Choi , Mary Fern´andez , Jerome Simeon "The XQuery Formal Semantics: A Foundation for Implementation and Optimization" May 31, 2002

4 C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. 20th ACM SIGMOD Int'l Conf.Management of Data (SIGMOD '01), 2001

5 Chulho Ahn, Quing Li, Ramez Elmasri, Shalli Prabhakar, Niroj Manandhar, Do Youn Kim " A Survey of Three Types of XML Indexing Techniques" ACM Transactions on Computational Logic, Vol. 37, No. 4, 12 2005, Pages 1 -24.

6 Gang Gou and Rada Chirkova "Efficiently Querying Large XML Data Repositories: A Survey" IEEE Transactions On Knowledge And Data Engineering, Vol. 19, No. 10, October 2007

7 H. Wang, S. Park, W. Fan, and P.S. Yu, "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures," Proc. 29th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03), 2003.

8 J. McHugh and J. Widom, "Query Optimization for XML," Proc. 25th Int'l Conf. Very Large Data Bases (VLDB '99), 1999.

9 James Keogh " J2EE: the complete reference" McGraw-Hill/Osborne, 06-Sep-2002 -

10 John O'Donahue " Java Database Programming Bible" ISBN:0764549243 John Wiley & Sons © 2002

11 Justin Zobel, Alistair Mo®at, and Kotagiri Ramamohanarao. Guidelines for Presentation and Comparison of Indexing Techniques. ACM SIGMOD Record, 25(3):10-15, 1996.

12 K. Beyer, R. J. Cochrane, V. Josifovski, J. Kleewein, G. Lapis, G. Lohman, B. Lyle, F. O¨ zcan, H. Pirahesh, N. Seemann, T. Truong, B. Van der Linden, B. Vickery, and C. Zhang. System RX: One part relational, one part XML n SIGMOD, 2005.

13 Kamil Toman "Storing And Indexing XML Data" ,2002

14 M Al-Nadawi, Dr Siobhan North, Dr Barry Eaglestone Indexing XML Databases: "Classifications, Problems Identification and a New Approach" , The University of

15 Mag.a Katharina "Flexible and Selective Indexing in XML Databases" ,Johannes Kepler UniversitÄat Linz, Altenberger Stra¼e 69, 4040 Linz, Austria ,August 2008

16 Matthias Nicola , Bert van der Linden "Native XML Support in DB2 Univeral Database " Proceedings of the 31st VLDB Conference , Trondheim , Norway,2005

17 Michael Kay "Ten Reason Why Saxon XQuery is Fast" Bulletine of the IEEE Computer Scociety Technical Committee on Data Engineering 2008

18 Michael Kay "XSLT and Xpath Optimizatin " "http:// saxon.sf.net"

19 Mohammad Samir, Patrick Martin "XML Structural Indexes " Technical report No. 2009-560 June 2009 School of Computing Queen's University Kingston, Ontario, Canada K7L3N6

20 N. Zhang, V. Kacholia, and M.T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML," Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004

21 Ntima Mabanza "Analyzing the Impact of XML Storage Models on the Performance of Native XML Database Systems – A Case Study" 978-0-7695-3984-3/10 © 2010 IEEE DOI 10.1109/ITNG.2010.207

22 S. A1-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of the IEEE International Conference on Data Engineering, 2002.

23 S. Abiteboul, P. Buneman, and D. Suci. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, ISBN: 1-55860-622-X, 2000

24 Steven A. Demurjian, David K. Hsiao, David K. Hsiao,Douglas S. Kerr , Robert C. Tekampe , and Robert J. Watson. "Performance measurement methodologies for systems. Proceedings of the 1985 ACM annual conference on database the range of computing: mid-80's perspective: mid-80's perspective, Denver, Colorado, United States, 1985, Pages: 16-28

25 Su Cheng Haw, and G. S. V. Radha Krishna Rao "Query Optimization Techniques for XML Databases " World Academy of Science, Engineering and Technology 22 2006

26 Su-Cheng Haw, Chien-Sing Lee "Evolution of Structural Path Indexing Techniques in XML Databases: A Survey and Open Discussion" ISBN 978-89-5519-136-3 Feb. 17-20, 2008 ICACT 2008

27 Xiaojie Yuan Xin Wang Chenying Wang Efficient XPath Evaluation Using a Structural Summary Index, 978-0-7695-3336-0/08 © 2008 IEEE DOI

CHAPTER 5

DATA PRESENTATION, ANALYSIS AND INTERPRETATION

5.1 Introduction

The research studied the data storage and data retrieval techniques for XML. The study was divided into two parts. In first part, the researcher studied to identify the XML utilization in software industry and in next part, the research studied XML storage and query execution policy.

In Part-1, the researchers studied the utilization of XML in different company. The study included identification of existing company problem and the solution towards that problem. In part-2, the study was performed for XML storage and XML query execution policy. This included existing XML storage and query execution policy. An experimental research technique was followed for the second part of study. The study was performed on following constituents:-

Table 5.1 Selection of Sample

Sr No	Constituents	No of Sample Points
Part-I	**XML - Utilization into Business Framework**	**36**
	Sample data collected from IT development and computer teacher	**1338**
Part-II	**XML File**	**24 files**
	XML Queries (XQuery)	**18 Queries**

The data was collected from different company web site and different business framework was study. For experiment research 24 XML files were selected. These files were broadly classified into two types: data-centric and Mix-mode. The Mix-mode type of files is a combination of data centric and document centric files. For each category it was again classified into two types. The fist category the files were extended breadth-wise and in second category the files were extended depth-wise. Total six files were selected for each category i.e. in first file 10 records, second files contains 100 records ,third file contain 100 records ,

forth file contains 10000 records, fifth file contain 50,000 records, sixth file contain 100000 records. For data analysis statistical parameters and graphics have been used wherever necessary and useful. The analysis is grouped in two-parts as follows:-

- **Presentation and Analysis of Data-Part-I: XML utilization in Business Framework:-**

XML is a key point for web business therefore different XML frameworks have been studied for understanding the XML uses in business as well as in different applications. The data was also collected from software experts to analyses their opinion on XML uses in business framework. The questionnaire was distributed 2000 peoples out of which only 1338 peoples responded back. The computer teachers and software industry experts were selected for study. These experts are playing different role in Software Company like software developers, software testers, software maintenance and other roles playing in the software industry like trainee, trainer and technical writer etc The thesis is focused on XML utilization in software hence the opinion of Software Developers were very important for it . The analysis was carried out on following criteria:-

1 The analysis of different company data for examines the role of XML, HTML and RDBMS technology in e- businesses.

2 Analysis of data from the IT Experts.

3 Analysis of utilization of XML in e- Business.

- **Presentation and Analysis of Part II: XML storage and query execution policy (Experimental Study).**

In this part the researcher studied the existing XML storage and XML query exaction technique. The study was divided into two parts:-

a. **Serve Method:-**The serve method was followed for studying XML storage and Query execution. The last fourteen years (1998-till today) research papers (IEEE and ACM) on XML storage and query execution strategy were analyzed. Different XML books were studied for understanding query execution policy and different storage methodology. The thesis is more

focused on XML-RDBMS storage and XML native storage strategy. The study was performed on following parameters:-

- ❖ XML storage Strategy
- ❖ Query execution algorithm
- ❖ Indexing strategy

b. Comparative study of three Commercial Products:-

Three commercial XML products (1) IBM DB2.9 (2) SAXON (3) eXit were selected for studying XML storage, query execution and XML indexing technique. The IBM DB2.9 is used for XML-RDBMS, XML-Native .The software eXit is also used for XML-Native storage, and SAXON is used XML-Navigational storage. The products were analysis on following criteria:-

1 XML storage

2 XML Query execution Plan

3 Index technique.

c. The Experimental results on Query execution and storage strategy :-

From the literature server researchers identified XML is stored into three type's storage: XML-RDBMS, XML-Native and Navigational storage. To test these storage and query execution strategy researcher test these storage and query execution on "IBM DB2.9" and "SAXON". IBM DB2.9 was used for testing 'XML-RDBMS' and 'XML-Native' storage, SAXON is used for "XML-Navigational" storage. The storage measurement parameter and algorithm was explained in chapter 4, page no- 94 . The detail structure of the XML files were describe into chapter 4, page no:91 The following analysis criteria were used for study:-

1) Query Execution Time
    a. Query Execution Time measurement : SAXON and IBMDB2.9
    b. Comparative Study of Query execution time of SAXON and IBM 2.9
    c. Comparative study of Query execution time of XML-RDBMS storage , XML-Native Storage
2) Disk Space Measurement
3) Disk Traffic Measurement
4) Study of Query execution Plan for XML-RDBMS and XML-Native

5) Study of Index creation time measurement of different XML files.

5.2 Presentation and Analysis of Data-Part-I: XML utilization in Business Framework

5.2.1 The analysis of different company data for examines the role of XML, HTML and RDBMS technology in e- businesses.:-

From 1990 onwards computer literacy increased into society, different companies changed their business into computer. The computer software was developed as per their business need. The computer software maintains business data effectively and efficiently. The desired data can be retrieved and reported successfully. The business terminology changed and it was replaced with the terms 'e-commerce', 'e-ticket', 'e-booking' 'EDI' etc. The performance of companies was also increased, they launched their own website. Hence more customers were communicating to it and become more customer focused.

As the business grows, the different websites need to communicate to each other to perform a complete business transaction. The business needs to interact not only with their own website or own company software but also with other company server and different database. Hence it is needed to establish a global standard for communication and establish business standard. A single business transaction is needed to communicate with different server and different database. Different companies were using different database products. Hence the communications were difficult.

This communication was effectively managed by new technology 'XML'. It was used as defector standard for data exchange between different application and web server. So it becomes an important component in the e-business. Due to its extensibility, flexibly, portability and user-friendly nature its utilization is increased. It was also used as effective communicator as well as database model for storing data. XML satisfies the need of all community users:-'database user, programmer, and documents search community.

The company may not require additional investment into database product. Since it is open standard, it is easily available and cost effective. No further investment is required for maintenance and purchasing the product. Therefore, most of the companies had been interested in investment on it.

The observations of company and XML utilization into their business are summarized as

	CASE	Why XML	Remark
Case 1	Expense Management System	The data is stored into XML fromat; XML store the the expenses of each employee	XML is used as extensibility characteristics
	System stored the expense of the emplyee	Expenses report is generated from the employee records	RDBMS can also used for creating and storing the information or generating the records but it doesnot have the characteristics as extensibility hence whenever new data is added , ir need to reconstruct the complete table
	Hence proper storage mechanium is required for storing the data	One of the iportatnt property of XML is extensibility, hence record can extends as per the user requirement	For fixed format record RDBMS is used instead of XML
Case 2	**Pharma Companey**	XML as a ability to handle a structrural and semi structural drugs related infromation	XML - extensibility and flexibility
	Deliver new drugs to different companey	XML has abilty to interact with different server; hence pharam IT solution utilised it for interoperatibility of clinicl care(healthcare) and clinical research (pharma)	XML - Interoperability
	For delivery the companey face the compatibility problem, different server need to interact to each other.	XML is usetilised to stored the meta data inforamtion about the product, (e.g. applicant, product, submission date ...), and links to the files with the actual submission information.	XML is used as meta data for representation of another data
		XML is used for:-	XML used for content management for pharma product
		Rerpesenting common technical document	
		Advisory Enets Reporting	
		Documenting - Drug development Process	
		Product Information Management System	
		Product Labeling inforamtion	
		RDBMS - used for stoing all secared data	
		HTML is used for web site display	

Analysis - XML replace RDBMS / HTML			
	CASE	**Why XML**	**Remark**
Case 3	XML and Aastra SIP Telephones	IT is an internet phone which is used by the employee of the organization who donot have PC	XML is used for diplay the desired information on interent
	Phone API which access different informtion by connecting different portal	The phone provide connectiviy to business process and people and provide the information of whether , can access organization data	In diffferent application desired data is embeded into XML records
		The phone can be utilized in different organizaion . E.g in education it is used for scheduling classes, notifying school closing , attendance etc. In travel / hotel , making dining reservaion , requesting houskeeing, check on flight etc	Still back office store the data in RDBMs format
		These functionalit is achive by using XML enable	
		XML technology is used for web display The display and keypad on Aastra SIP phones can be used to easily access and display data from the XML applications	
		Aastra SIP Phones run over a data network infrastructure, software developers can just as easily create integrated voice and data applications for Aastra SIP Phones using XML as they can for any Internet application.	

Analysis - XML replace RDBMS			
	CASE	**Why XML**	**Remark**
Case 4	Discussion forma is "Does XML replace RDBMS"	XML is not a database	XML is used for moving data between different database
	Discssion does XML replace RDBMS	Relational databases are proven technology with more than 20 years of implementation experience. They are solid, stable, useful products.	XML is used for configuration
		XML is a very useful technology for moving data betweendifferent databases or between databases and other programs. However, it is not itself a database. Don't use it like one.“	XML is used for serilizing non complex objects in a non-relational format .
		When your data volume is small and you don't have to do complex queries against it. XML files are good for things like storing configuration or document templates.	XML is not scalable data
		storing large amounts of XML files for a datasource is not very scalable but it makes it easier to move the data around. XML can also be used to serialize complex objects in a non-relational format, which could eliminate the need for an ORM if you can serialize/deserialize your objects directly from the XML	RDBMS is scalable database and can manage concurrancy in daabase . XML is unable to manage concurrancy and scalability
		The web site applications need to connect different applciation with differet database . SQL statemnes are need to modify as per the database tables/ structure , It is very difficult to organize this SQL statement . One of the solution is XML file. XML file embeded these SQL statement Each XML file contains the set of SQL statements for one of the dozen databases. The code determines which database is accessed for the logged-in user, and retrieves the appropriate SQL from the specified XML file.	XML is sued for document template but proprer SQL queries are required for retriving the desire information .
			XML is more suited as data interchange format than as data storage format

Analysis - XML replace RDBMS / HTML			
	CASE	**Why XML**	**Remark**
Case 5	Kola air	XML is used for	The web site prefer to use the XML for
	Koala specialize in information management for the aerospace and aviation industries. More specific they provide the service in aircraft manufacturing,in flight operations,and in aviation maintenance, repair & overhaul (MRO) ,in providing aeronautical information.	Aircraf documents are represented by using XML	Managing large document
		For docemnting Aeronautical Information Publication repoert	Manage complex document
		Taining , operational manual and producton manual are represented by usng XML	Mangeing frequenly updated
		transalting the documents (production / operational) into different lanuage	Reuing exisitng XML content in the publicaion
			XML help in document customization
			XML is used fr miantaining and publishing information in different languages
			XML is used as a tool for translating the document
			This canot be done by RDBMS
			RDBMS system can be used for managing docuemetn informationand product informaion
Case6	Health care	The hospital has , staff of more than 2,000 physicians handles more than 1 million clinic visits and 80,000 hospital visits per year	It include a document repository and metaddata repository ,pairing medical record files with DB2 database
	The UCLA Health System is one such organization: a multi-hospital healthcare provider with a diversity of clinical and healthcare applications	A healthcare system of that size must process large amounts of data on a daily basis, including medical records updates; lab results; MRI, computed tomography (CT), and electron beam CT angiography images; admission, discharge, and transfer data; and pharmacy orders	The system suggest t it used RDBMS with XML
		The main requirement of the hospital al data stored securely and reliably. The data must easily search , transmitted and must be organized by different employee spread across mulitple locations.	The case suggests the XML utilization with RDBMS database .
		Sysem connect with document manageemnt system of PODs , the Clinical Campus,	

Summary of the Study: - The study summarized into following table 5.2 as:-

Table5.2 Summary of XML Utilization in different Company

	RDBMS Used	**HTML Used**	**XML is used as**
Case 1	Yes	Yes	Maitaining EMPlyee records
Case 2	Yes	Yes	XML meta representation
			Document representation
			Content Manaegment
			Interoperatibility
Case 3	Yes	Yes	Interoperatbility
			Database
			Content Manegemnt
Case 4	100% RDBMS used for data storage	Used for web designing	XML is used for moving data between different database
			XML is used for configuration
			XML is used for serilizing non complex objects in a non-relational format .
			XML is not scalable data
			RDBMS is scalable database and can manage concurrancy in daabase . XML is unable to manage concurrancy and scalability
			XML is sued for document template but proprer SQL queries are required for retriving the desire information .
			XML is more suited as data interchange format than as data storage format
Case 5	Yes	--	Document Management
			Web Content Management
Case 6	Yes	--	XML used as Databse
Case 7	Yes	Yes	XML as Metadata
			XML - document and data representation
			Wencontent Management
Case 8	Yes	Yes	Used in e-mail system
			Data transformation from ne system to another System
Case 9	Yes	Yes	Used for data transfer
			Used for mulitple language interpretation
Case 10	Yes	Yes	Business Workflow explanation

The conclusion of the case study is listed in table 5.3

Table5.3: Data Analysis: XML Utilization in different Company

Different uses of XML technology		
Sr No	XML Used	Percentage uses
1	XML used as Database	30%
2	XML is used for Contentment Management	30%
3	XML is used for inter-operability	60%
4	XML is used for Document representation	20%
5	XML is used for Metadata representations	40%
6	XML used for Language translation	10%

RDBMS systems are still used into company along with XML	
	Respondent
YES	100%
No	0%

HMTL is used into web site along with XML technology	
	Respondent
YES	99%
No	1%

5.2.2 Analysis of data from the IT Experts:-

For the sampling, the Data is collected from internet. The researcher asked three questions to respective users. The questions are:-

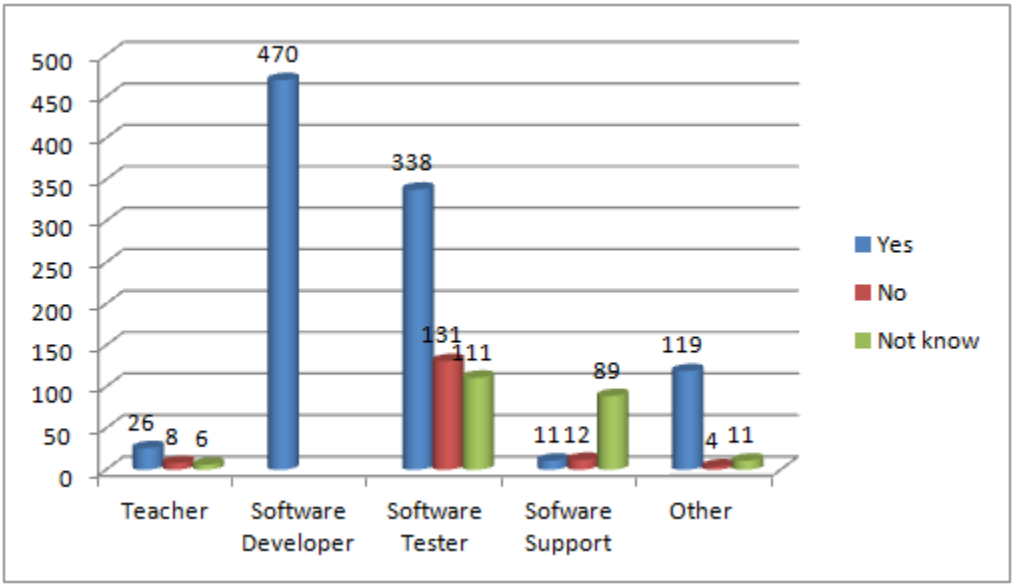
I. Is XML is sufficient for only representation of business framework?

The table 5.3 demonstrates that 72.16% population agrees the XML sufficient of representation of business framework; 11.06% said that it is not an important component. 16.24% people passed neutral comments on it.

Table 5.4 Distribution of sample of XML Utilization in Business Framework

Occupation	**Strictly Agree**	**Neutral**	**Strictly Disagree**
Teacher	26 (65%)	6 (15%)	8 (20%)
Software Developer	470 (100%)	-	-
Software Tester	338 (58.27%)	111 (19.13%)	131 (22.58%)
Software Support	11 (9.82%)	89 (79.46%)	12 (10.71%)
Other	119 (88.80%)	11 (8.20%)	4 (2.98%)

Figures in bracket indicates Percentages



Graph 5.1: Utilization of XML in Business Framework

Above graph 5.1 suggests that from the computer teacher community 65% agree that XML is an important component in web software development as well as in e-business. 20% people are not agreeing with it; 15% passed no comments on it. From the software development community 100% people are agreeing XML importance, 58.28% of software testers suggest XML as an important component in software development, 22.59% are not agreeing with it 19.44% passed no comments on it. The 79.56% software maintenance persons passed no

comments on it. 9.82% agreed and 10.71% did not agree on it. From 'Other' 88.81% agreed XML is important in development, 2.9 disagreed and 8.21 did not respond to the question.

The assessment of XML utilization in business framework researchers follows 3-pointer Likert scale, having categories like strongly agree, neutral, strongly disagree. The table shows the ranking results. It is seen that highest average value is assign to software developers which is 3 followed by others 2. 89. The next weighted value is assigned to 'Teacher' which is 2.45, Software Tester is 2.36 and Software Support team the value is 1.99. Since the assumption is focus on software developer point of view, their opinion must consider and it is in ranking of 1.

Occupation	Strictly Agree	Neutral	Strictly Disagree	Total	Wt. Avg.	Ranking
Software Developer	470			470	3	**1**
Other	119	11	4	136	2.82	**2**
Teacher	26	6	8	40	2.45	**3**
Software Tester	338	111	131	580	2.36	**4**
Software Support	11	89	12	112	1.99	**5**

II. **XML can be used as: Database, Documentation and Content management System**

The following table5.5 suggested 16.89% people advice utilization of XML as Database tools. 20.03% recommended XML utilization as document purpose, 47.16 % counseled it can be used for all purpose and 7.55% hasn't provide their advice.

Table5.5 XML can be used as:

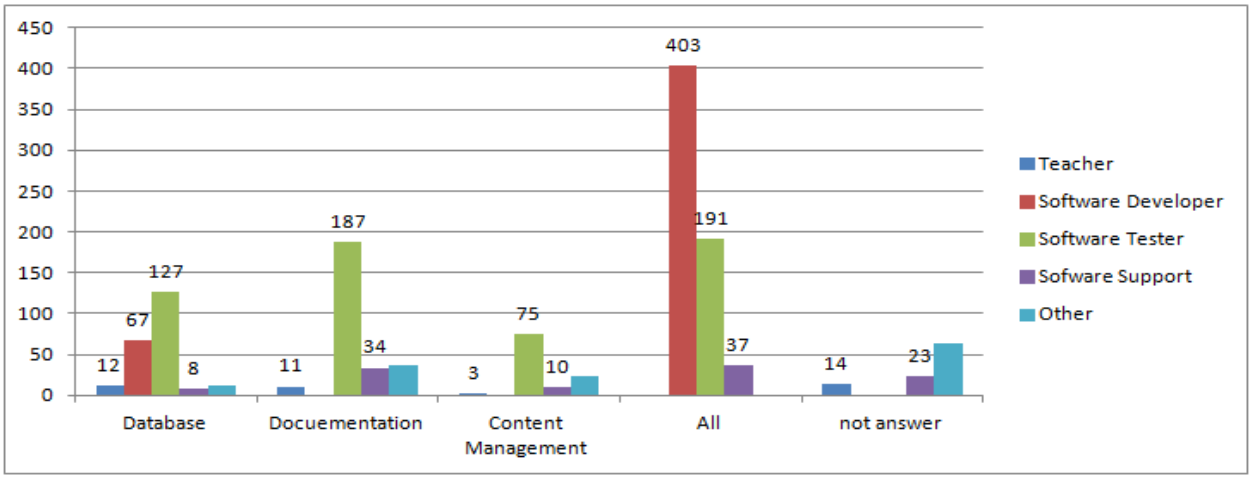
Database, Documentation and Content management System

Occupation	Database	Documentation	Content Management	All	not answer
Teacher	12 (30%)	11 (27.5%)	3 (7.5%)	-	14 (35%)
Software Developer	67 (14.26%)			403 (85.74%)	
Software Tester	127 (21.90%)	187 (32.24%)	75 (12.93%)	191 (32.93)	
Software Support	8 (7.4%)	34 (30.34%)	10 (8.92%)	37 (33.03%)	23 (20.53%)
Other	12 (8.82%)	36 (26.47%)	24 (17.64%)		64 (47.05%)

Figures in bracket indicates Percentages

Table5.4 XML can be used as: Database, Documentation and Content management System

Following graph5.2 displays the detailed XML utilization:-



Graph 5.2 XML can be used as: Database, Documentation and Content management System

30% Teacher suggested it was used as database, 27.5% suggested it was used as documents purpose, 7.5 advised it as content management and 35% didn't answer it. Software developer

suggesed utilization of XML is for database (14.26%) and 85.74 advised to used XML for all purpose. The 21.90% software tester suggested XML utilization as database, 32.24 % for documentation, 12.93% for content management and 32.93% used it for all purpose.

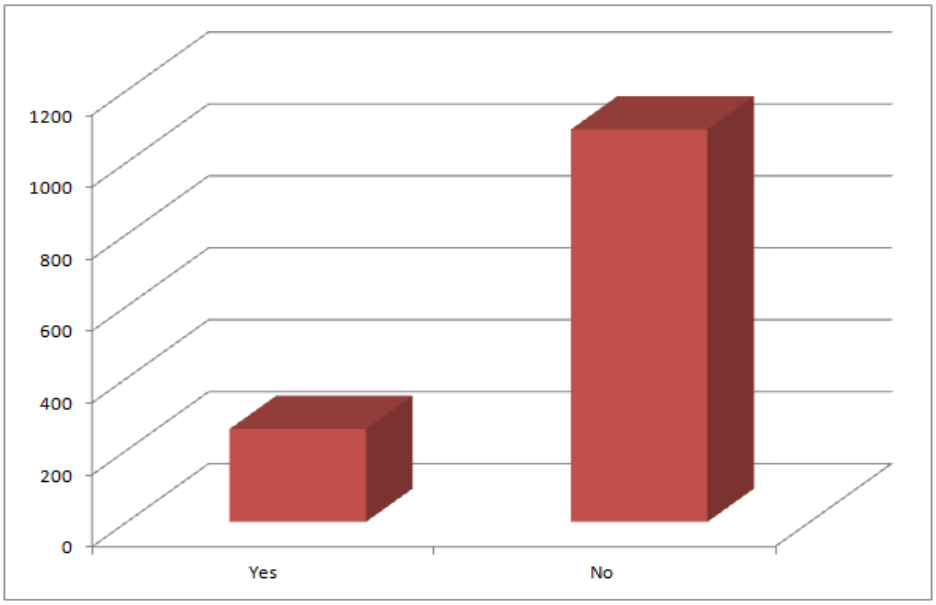
XML cannot replace RDBMS/ HTML

Table5.6 suggests 3.51% people agreed XML replaces RDBMS / HTML. 81.54 people advised XML couldn't replace RDBMS or HTML. All these tools play different roles in software development. 14.95% does not answer any query. The community wise detailed analysis is presented into graph5.3

Table 5.6: XML cannot replace RDBMS/ HTML

Sr No	XML replace RDBMs/HMTL	No of Respondent
1	YES	855 (83.74%)
2	No	166 (16.26%)
	Total	1021 (100%)

Figures in bracket indicates Percentages



Graph5.3 XML cannot replace RDBMS/ HTML

XML cannot replace RDBMS or HTML. They play different roles in software development.

5.2.3 Analysis of utilization of XML in e- Business

The utilization of computer in the business increased. The utilization of XML in the business is as follows:-

- XML used for communication of business transaction between the companies:-
  - E-business XML is backbone: The Business needs urgent communication. It is either done by Fax or email. All these mode of communication is done by manual intervention. XML is used to initiate this activity automatically. In previous years, EDI was used for the business communication. It describes ANSI X 12 standards for communication but it didn't satisfy all business transaction communication need. EDI transaction was costly. EDI depends on proprietary networks and X12 document descriptions, large companies feel quite free to modify any way they please. So only large companies utilized it. XML replaces this EDI standard. XML is pen standard, and cost effective, hence small companies are also interested in the investment.
  - Many companies are using XML for electronic data exchanges: - Another mode of business communication is data transfer from one business to another business; XML is also used for this transaction. XML is capable of handing different database efficiently. It communicates this information to the different sites.
- XML is used for representation of complex document: - The product or service information is very complex. A representation of this information is easily done by XML. XML is capable of adding dynamic information, so as change in services/product can easily be updated into XML. Many companies are also using XML files to write the documents like operational manual and product procedure manual, user guide etc.
- The businesses are increased globally. Hence the company information is transferred into different languages across the globe. XML is very helpful when software is publishing in more than one language from the same source. By storing content in XML source files, transaction cost is reduced. Then the data is created into several languages by just clicking a button.
- XML is used as metadata: - The metadata is data about data. XML is used for representation of metadata information.

- Many software development frameworks are using XML files to store the configuration data for the application. For example, JPA uses persistence.xml file for configuring the JPA environment, financial data like electronic exchange of securities transactions, the structured financial products.
- Data development is another use of XML. The program can use the XML data files for business processing.
- The XML data files can be used to generate dynamic continent by applying different style sheets.
- Database development with the XML is another good use of XML.
- It is used for content management

All above summit insists that XML is important in e-business domain. All the business need is not only satisfied by only XML technology but it needs additional support from other techniques.

To test this, XML data was collected form the different business to study their business framework. The analysis of the study is represented into following table 5.6:-

❖ **Analysis of utilization of XML in IT Business :-**

Table5.7:Extension of XML lanugage

XML - Utilization into Business Framework				
Sr No	**Industry**	**Extention of XML**	**Description**	**Organization**
1	Food XML	mpXML	XML specification for exchanging business information among all segments and entities in the meat and poultry supply and marketing chain	Meat & Poultry Data Standards Organization (mpXML).
2	Advertising XML	adXML(XML for Advertising)	"is an international, open standards organization, aiming to define an advertising XML schema for both on-line and off-line media."	The AdsML Consortium.
3	Astronomy XML	FITSML	XML specification for astronomical data, such as images, spectra, tables, and sky atlases.	NASA.
4	Building XML	oBIX	The Open Building Information Exchange specification that will enable enterprise applications to communicate with mechanical and electrical systems in buildings.	OASIS
5	Chemistry XML	Chemical Markup Language (CML)	XML specification covering macromolecular sequences to inorganic molecules and quantum chemistry.	Open Applications Group, Inc
6	Construction XML	Architecture Description Markup Language (ADML):	ADML is based on ACME, an architecture description language. ADML adds to ACME, a standardized representation the ability to define links to objects outside the architecture	The Open Group.
7	Education XML	Schools Interoperability Framework (SIF)	An XML specification for data sharing among schools, kindergarten through 12th grade.	SIF Association
8	Finance XML	eXtensible Business Reporting Language (XBRL)	XML specification that describes financial information for public and private companies and other organizations. They have created XML taxonomies. Since financial reporting varies by country, the taxonomies vary by country	XBRL International
		Financial Information eXchange (FIX)	XML specification for the real-time electronic exchange of securities transactions	FIX Protocol Organization (FPL)
		Financial products Markup Language (FpML)	XML specification for swaps, derivatives, and structured financial products.	International Swaps and Derivatives Association (ISDA
		Interactive Financial Exchange (IFX)	XML specification for electronic bill presentment and payment, business to business payments, business to business banking	Interactive Financial eXchange Forum (IFX Forum).
		Market Data Definition Language (MDDL):	XML specification to enable interchange of data necessary to account for, to analyze, and to trade instruments of the world's financial markets	Software & Information Industry Association (SIIA)
		Open Financial Exchange (OFX)	XML specification for the electronic exchange of financial data between financial institutions, businesses and consumers via the Internet	Open Financial Exchange (OFX) Consortium.
		SWIFTStandards:	SWIFTStandards develops business standards to support transactions in the financial markets for payments, securities, treasury and trade services.	SWIFT
9	Reserch Lab	Research Information eXchange Markup	XML specification to tag any piece of research content, in any form or media with enough detail for end users to	Research Information eXchange Markup Language (RIXML).org

XML - Utilization into Business Framework				
Sr No	**Industry**	**Extention of XML**	**Description**	**Organization**
10	Government XML	Election Markup Language (EML)	XML specification for the structured interchange of data among hardware, software, and service providers	OASIS
		The National Information Exchange Model (NIEM)	It is designed to develop, disseminate and support enterprise-wide information exchange standards and processes that enable jurisdictions to effectively share critical information in emergency situations	National Information Exchange Model (NIEM
		NIEM builds on the Global Justice XML Data Model (GJXDM).		
		Tax XML	Tax XML includes a vocabulary of terms, a repository of artifacts including XML templates, documents exchanged for tax compliance	OASIS
11	Human Resources XML	HR-XML	XML specification designed to enable e-business and the automation of human resources-related data exchanges.	HR-XML Consortium
12	Instruments XML	Instrument Markup Language (IML)	XML specification that applies to virtually any kind of instrument that can be controlled by a computer.	NASA
		ACORD XML for Life Insurance	XML specification that addresses real-time requirements by defining business transactions that include both request and response messages for Personal Lines, Commercial Lines, Specialty Lines, Surety, Claims, and Accounting transactions	ACORD
		ACORD XML for Property and Casualty Insurance		
		ACORD XML for		
13	Legal XML	Global Justice XML Data	XML standards to create legal documents	OASIS
		LegalXML has		
		LegalXML Electronic Court		
		LegalXML eContracts		
		LegalXML eNotary		
		LegalXML Integrated Justice		
		LegalXML Legislative Documents		
		LegalXML Online Dispute Resolution (OdrXML):		
		LegalXML Legal Transcripts		
		LegalXML Subscriber Data Handover Interface (SDHI)		
14	Manufacturing XML	Planning and Scheduling Language on XML Specification (PSLX):	XML message standards for information about planning and scheduling in manufacturing industries	
		papiNe	XML standard for paper industry	
		Production Planning and Scheduling (PPS		
15	News XML	News Industry Text Format (NITF	XML specification for the content and structure of news articles.	IPTC website.
		NewsML	International Press Telecommunications Council	
		Publishing Requirements for Industry Standard Markup (PRISM):		IDEAlliance
		SportsML	XML data for sport data	SportsML

XML - Utilization into Business Framework				
Sr No	Industry	Extention of XML	Description	Organization
25	Telephone XML	Call Control eXtensible Markup Language	Call Control eXtensible Markup Language (CCXML) is the language that provides telephony call control for scripting languages such as VoiceXML.	
26	Media Control	Media Server Control Markup Language	The Media Server Control Markup Language (MSCML) is an XML markup language. MSCML is used in conjunction with Session Initiation Protocol (SIP) to provide two broad classes of functionality	
27		Media Server Markup Language	MSML is used to allow media servers to extend support on SIP implementations to include functionality previously available only in the Media Gateway Control Protocol (MGCP)	
28		Synchrnized Mltimedia Integration Language	XML document type for describing mulimedia presentations	
29	Mathematics	Math Markup Language	Markup language for mathematics	
30	Chemical Industry	Chemical Markup Languag	XML based language used for desribing molecular information	
31	Sports-XML	GolfML	The purpose of GolfML is to provide a XML-based file format for the exchange of golf-related data.	
			The markup language will focus on:	
			1. The description of golf courses, including geo-spatial data	
			2. The golfer's logbook, with scorecards and statistics	
32	Space	SML (Spacecraft Markup Language)	The purpose of Spacecraft Markup Language (SML) is to extend XML to provide a standardized representation to be used in the space domain	
			"Spacecraft Markup Language (SML) is an extension of the Extensible Markup Language (XML) providing the Space Community with a standard definition of XML tags and concepts of structure to allow the definition of spacecraft and other support data objects.	
33	Music	MML(Music Markup Language)	Music Markup Language (MML) is an attempt to mark music objects and events with an XML-based language. Marking such objects should enable managing music documents for various purposes, ranging from music theory and notation to practical performance.	
34	Book	Bank Internet Payment System	Sponsored by a consortium of leading technology banks, BIPS will focus on encouraging companies to initiate a variety of payment transactions via the Internet through increased system security.	
35	Engineering	Engineering: LandXML	Landmark development Language	
36	Artifcial Intelligence	AIML: Artificial Intelligence Markup Language	AIML (Artificial Intelligence Markup Language) is an XML-compliant language	

From the above table 5.6XML is used in every industry. It is used for the document content representation; it is used for transformation of data from one machine to another machine or from one network to another network, from one software system to another software system. To transfer data from one system to another different web services are used, these web services are also represent the information in XML. The web service describes a standard way of integrating web-based application. Web services share business logic, data and processes through a programmatic interface across network. These communication performed by XML. The following Table 5.8 listed some of the important web services.

Table5.8: XML Protocol for data transformation

	XML Protocol			
Sr No	Protocol	Organization	Architect	Website reference
	Generic XML protocols	provide a mechanism to represent arbitrary data.		
1	XML-RPC	Userland	Dave Winer (userland), Bob Atkinson (microsoft), Mohsen Al-Ghosein (microsoft), Don Box (developmentor)	http://xmlrpc.scripting.com/spec.html
	SOAP	Ariba, Inc.	Don Box, DevelopMentor	http://www.w3.org/TR/soap/
		Commerce One, Inc.	David Ehnebuske, IBM	
		Compaq Computer Corporation	Gopal Kakivaya, Microsoft	
		DevelopMentor, Inc.	Andrew Layman, Microsoft	
		Hewlett Packard Company	Noah Mendelsohn, Lotus Development Corp.	
		International Business Machines Corporation	Henrik Frystyk Nielsen, Microsoft	
		IONA Technologies	Satish Thatte, Microsoft	
		Lotus Development Corporation	Dave Winer, UserLand Software, Inc.	
		Microsoft Corporation		
		SAP AG		
3	WDDX	UserLand Software Inc.		
		Allaire	Simeon Simeonov (Allaire), Nate Weiss (independent/JavaScript), Spike Washburn (Allaire/Java), Rasmus Lerdorf (PHP3), Scott Guelich (Scripted.com/Perl)	
4	ebXML	United Nations framework of the Economic and Social Council, the United Nations Economic Commission for Europe (UNECE)		
5	BizTalk	Microsoft	Satish Thatte	http://msdn.microsoft.com/en-us/data/bb190600.aspx
6	Wf-XML mailing list	Workflow Management Coalition	WfMC Working Group 4	
7	HL7	Health Level Seven is the Application Protocol for Electronic Data Exchange in Healthcare Environments.		

	XML Protocol			
Sr No	Protocol	Organization	Architect	Website reference
	Generic XML protocols	provide a mechanism to represent arbitrary data.		
8	OFX	Open Financial Exchange (OFX) allows financial institutions to communicate with one another and their clients about account transactions	Microsoft	Harness the power of XML to Open Financial Exchange files Enabling XML parsing for non-XML OFX files by Colin Beckingham, Researcher, Freelance 17 March 2009 http://www.ibm.com/developerworks/library/x-ofxv2/
9	web service-related XML protocols	protocols with a fixed grammar targeting a particular application		
10	WSDL		Ariba	Erik Christensen (Microsoft)
			International Business Machines Corporation	Francisco Curbera (IBM)
			Microsoft	Greg Meredith (Microsoft)
				Sanjiva Weerawarana (IBM)
11	WIDL		webMethods	Phillip Merrick (webMethods)
				Charles Allen (webMethods)
12	UDDI		ARIBA	
13	DISCO	Microsoft		
14	NASSL		IBM	Francisco Curbera
				Sanjiva Weerawarana
				Matthew J. Duftler
15	ICE	GCA	Neil Webber, Vignette Corporation	
			Conleth O'Connell, Vignette Corporation	
			Bruce Hunt, Adobe Systems, Inc.	
			Rick Levine, Sun Microsystems, Inc.	
			Laird Popkin, Sothebys.com	
16	IOTP	Commerce One	David Burdett	
17	eCo	not reviewed yet		
18	XMOP	not reviewed yet		
19	XIOP	Financial Toolsmiths AB	Anders W. Tell (Financial	
20	P3P	W3C	Lorrie Cranor (AT&T),	
			Marc Langheinrich (ETH Zurich),	
			Massimo Marchiori (W3C/MIT),	
			Martin Presler-Marshall (IBM), Joseph Reagle (W3C/MIT)	
21	CCPP	W3C	Franklin Reynolds (Nokia Research Center)	
			Johan Hjelm (W3C/Ericsson)	
			Spencer Dawkins (Nortel)	
			Sandeep Singhal (IBM)	
22	E-Speak	hp		

Summary and Conclusions:-

From the above analysis, the common uses of XML are noticed

- Web development: - XML is used as CMS in web development.
- Documentation: - XML is easy user friendly; it is easy to understand for human. Most of the company suggest their documentation in XML format
- Database development: - XML is simply written in such a way to allow machines to read it and make it accessible to a database. It is used for transfer of data from one machine to another machine irrespective of database used in the application.
- XML cannot use as RDBMs or HTML.
- XML is used for witting protocol and web service
- XML is used for writing business documentation and the language is extended as per the business need.

5.3 Presentation and Analysis of Part II: XML storage and query execution policy.

5.3.1 Serve Method : XML storage Strategy

Introduction:-

There are three main alternatives of storing XML document. The first approach is mapping XML data into relational database system (RDBMS), second approach is specialized data management system known as native database (NXD). This thesis is focused on XML-RDBMS approach for data storage and data retrieval techniques. The study was performed on same. XML data organization into different relational tables were difficult task because XML organizes data in a hierarchical, object-oriented and multidimensional way in the form of a tree with arbitrary depth and width [1] while traditional relational database management system consists of table for data storage. It stores the data into rows and columns thus it has difficulty in storage of such tree structure, recursive and order data. The following table5.9 discusses the major difference between XML and RDBMS data.

Table5.9: XML and RDBM Comparision

XML	RDBMS
Hierarchical Structure	Data is storeed into different tables
Elements are in order	No straightforward strategy for representation of ordered data.
Elements are nested	Automic cell values
Able to adopt to the changes i.e A new node can be added or deleted without prior permission	It is enable to adopt to the changes randomly
Schema is optional	Schema is required
The direct retrieval of data is possible	Join operation is nessesary for retreval of data
Diffciluty in concurrent execution	Concurrancy mechanism is well esatablished
Difficult to implement database features like security , transaction management etc	Mature query optimization and security, transaction management , normalization etc already present

In lierature five types are suggested for XML-RDBMS storage. It is stored as String in which XML data is stored as string value (clob / blob) in tuples in relational storage. This is simple and easy approach for storage and good for small XML file. It considered all data as string format and so cannot capable of storing the other data type hence it cannot process all types of queries and difficult to manage the large XML file.

XML data is stored as "Tree Representation" style, different tables are used for storage and these tables are linked with each other with additional attribute. The XML file structure is defined by DTD or XML Schema or it may not be defined by the files. Therefore tree representation has two types:' Schema Centric Approach' and 'Schema Less approach'.

Another method for storage is "Publishing and shredding". This technique is suitable for XML enable database. The standard XML shedding command is executed and the file is converted into tables. Each XML element is mapped into the table row and each column of that row is sub element of the XML element. An XML view of relational data can be treated

as virtual XML document, and XML queries can be executed against the virtual XML document. XML queries are executed on this virtual view. A special 'select XML element clause is used for execution of query. This clause executes P-C and A-D types of query but unable to execute tree traversal queries like following sibling, preceding, etc.

From 2009, the commercial database vendor DB2, oracle introduced new data type ".xml" for storing the file in tree structure format only. Each document id is stored as a tuple into the table. Different types in indexes are created on it. But large XML files are unable to store in this fashion. It is needed to be shredded horizontally and stored it into different row thus different records are created for this; it is called as medium grain storage.

From these storage types we observed the new data type maintains tree structure, recursion relationship by RDBMS storage hence XML file structure is maintained. The clob/ blob storage is not useful for querying data hence we find 'Schema Centric ' and 'Schema less' are the research areas for identification of problem and their solution. The detailed study of these approaches were listed into chapter3 page no 44 till page 47.

5.3.1.1 Serve Method : Query Execution:-

XML queries are classified into different types: containment queries are used for retrieval of structural relationship data. Order queries are used for retrieve the XML tree order data like: forward matching, reverse matching, Total sixteen different axes are used for retrieve the data. The twig query used to retrieve a small XML tree. The content matching queries retrieve the content of XML tree nodes. In total matching queries the complete XML path is specified for data retrieval and in partial matching queries the queries path is matched in between of the complete query. The following table listed the different RDBMS proposal along with query execution capability.

Table5.10 XML-RDBMS Approach towards Query Processing

XML-RDBMS Approach									
Techniques	**Query Processing Cost**	**Dynamic Updates**	**Query Execution**						
			Containment Queries		**Order Queries**	**Twig Queries**	**Content Queries**	**Toal matc hign**	**Partial Matchig n**
			P-C	**A-D**					
EDGE Approach									
Edge Approach	High	High	Yes	No	No	No	Yes	Yes	No
Binary Approach	High	High	Yes	Yes	Yes	No	Yes	Yes	No
Universal Approach	High	High	No	No	No	No	Yes	Yes	No
NODE Appraoch									
Extension of Inverted List	High	High	Yes	Yes	Yes	No	Yes	Yes	No
s-XML Model	Moderate	Possble	Yes	Yes	Yes	No	Yes	Yes	No
MAXDOR	Moderate	Possble/ Partialy	Yes	Yes	Yes	Yes	Yes	Yes	No
Path Materialization									
XPARENT	High	No	Yes	Yes	Yes	No	Yes	Yes	No
XREL	High	No	Yes	Yes	Yes	No	Yes	Yes	No
Reverse Path Approach	High	Possle	Yes	Yes	Yes	Yes	Yes	Yes	Yes
BLASÉ	Low	Possible	Yes	Yes	Yes	Yes	Yes	Yes	Yes

From the above table 5.10 the "Path Materialization approach" is the best solution for query execution. The query execution time is less as compared to other approach because it required less join operation. The new inserted and deleted records are effectively stored into database without reconstruction of database. So reconstruction cost is less. The "BLAS" is the best proposal from all PM approach. It not only reduced dynamic cost but also reverse conversation is possible i.e. it is possible to convert RDBMS tables into the XML file. Label arithmetic of BLAS effectively solved queries with specified predicate value. The detail comparison between XML-RDBMS , Node_id technique is discussed into annexure-I.

5.3.2 Serve Method : XML Index strategy :-

In query execution, index plays an important role. It is used for fast retrieval of data from XML storage. In XML, index designing is a very difficult task because it has recursive structure and hierarchical relationships. XML indexes are classified into four category, they are listed in table 5.11:-

Table 5.11 XML index Strategy

Sr No	Index Type	
1	Structural Summary	XML file is converted into small summary format and indexes are created on it.
2	Structural Join	These indexes are created on the structural summary information. i.e on P-C , A-D relationship.
3	Selective Index	These types of indexes are like RDBMs index. The DBA select the attributes on which the indexes are created. E.g they are created on frequently asked queried path.
4	Value Index	These indexes are created on attribute value as well as node value.

Since this thesis is more focus on XML-RDBMS storage. The XML- RDBMS proposals indexes are listed table 5.12:-

Table 5.12 XML-RDBMS –index strategy

Sr No	**Proposal**	**Index**
1	Edge Approach	Cluster index is prebuilt on Label and additional index is on value attribute.
2	Binary Approach	Index is on value attribute
3	Universal Approach	Cluster index is prebuilt on Label
4	Extension of Inverted List	----
5	s-XML Model	Index on node_id value
6	MAXDOR	Index on node_id value
7	XPARENT	B+ tree index on path string
8	Reverse Path Approach	B+ tree index on path string
9	BLASÉ	B+ tree index on path string

5.4 Comparative study of three Commercial Products:-

From the theoretical background, researcher studies the three commercial products for XML storage, Query execution and Indexing strategy:

(1) IBM DB2

(2) Saxon

(3) eXit

The comparative studies of the products are listed below:-

XML three Product Comparision Study

	IBM dB2	SAXON	eXit
Storage	Three Types of Storage	The SAXON is an open source XML query processor developed by Michael Kay. It is not a database product	A unique number is assigned to each XML node; these node_ids maintain the hierarchical relationship among the XML nodes. The 'level-m' numbering schema is used for
	External File Storage		
	XML data is sored in externla file		
	XML data file is stored by using user defined data type	**Three Compnents can be used in SAXON**	
	Realtional Database storage	**tree constructor** :- XML docuement is parsed and constrct XML tree in memeory	It store XML information into four files: Collection.dbx, DOM.dbx, element.dbx, words.dbx
	Partial Shreading :- Only a subset of elements or attributes from each incoming documents are shreadded into relational table	**tree navigator** :- This class is used desired XML node in the XML tree	Collection.dbx:- For every relationship in an XML file a object is created . The 'collection_id' is a unique identifier for each hierarchical relationship.
	Hybride Shreading :- It is combination of relatinal and native database storage	**The style sheet compiler** :- It analyses the XML document prior to execution	dom.dbx:- For each document node, node_id is created and its address is stored into data pages.
	Native XML sorage:-	**The style sheet interpreter**: - This is the core component of the saxon	element .dbx:- The 'name_id' is a unique node identifier for each element and attribute node
	Special type ".xml" is used for storing XML file	**The *outputter* component**:- It is used to control the output process	words.dbx:- This is nothing but the collection of text node information
	Classify in three types:-		
	Coarse-grain :- Two columns table is used for storing the XML data. The first column stores the identification of XML document, and the second column stores XML document itself.		
	Medium Grain :- The XML document is very large. It cannot be stored into a row, so the document is sliced and each row of the table stores a slice of the document. This storage is called as medium grain storage		
	Fine Grain :-Individual components of the XML document are stored in columns of multiple tables of the database. It is nothing but the RDBMS storage.		

Summary and Conclusion:-

Researcher had reviewed different XML techniques for processing XML Query. These techniques are categories them into two types: - XML to Relational approach and XML to native approach.

Representing XML data into relational table format is very difficult, because XML is hierarchical, recursive and order data. Due to these features, it is difficult to store it into RDBMS format. The different proposal suggests the different techniques for storage of data. Once the data is converted into RDBMS format, it can take the advantage of RDBMS feature like query optimization, indexing etc.

In native database, XML data is stored into inverted list or tree or in multiple link list/ stack. Hence the data is stored into different data structure; the query equip-joint operations are evaluated easily. It reduced 'join' operation also. But while executing query the intermediate results are very high which increased query processing cost. Most of the proposal is developed for XPath query execution but XQuery optimization is still in progress. Another database facility like concurrent execution, indexing need to be developed from the scratch.

Researchers observed that if the data is more hierarchical format it always prefers to store the information into Native format. If the data is more secured and structured format, operated on multi user environment, it is always preferred to store it into RDBMS format. The combination of RDBMS storage and Native query execution method is good trade-off.

5.5 The Experimental results on Query execution and storage strategy

5.5.1 Query Execution Time Measurement of IBM DB2.9 and SAXON:-

Measurement of Query Execution time: - For the laboratory research two product were selected:- IBM DB2.9 and SAXON . The Experimental detail, its hardware/software requirement, the measurement algorithms was explained in chapter 4 page no94 Experimental Parameter and query data set was explain in chapter 4 page no 95.

Total 24 XML files and eighteen queries were considered as input. Broadly the files are classified into four. The Data Set1.1, the depth of the XML file is four and the breath of the file is increased. This dataset contain six more files with 10 records, 100 records, 1000 records, 10000 records, 50000 records and 100000 records. The structure is presented into chapter 4 page 91. In second Dataset1.2, the breadth of the file is tenth level and depth of the

file is increased. This data set also contain six more files with 10 records, 100 records, 1000 records, 10000 records , 50000 records and 100000 records. The Dataset2.1 and Dataset 2.2 contains the mixed mode types of files. The mix-mode type of file is a combination of data and document contains. The Dataset 2.1 has depth 4 and breadth is increasing and the dataset 2.2 has depth ten and breadth is increasing. These dataset further classified into six more files with 10 records, 100 records, 1000 records, 10000 records, 50000 records and 100000 records. The query execution time is recorded as follows:-

Experiment Result for Data Set1:-Set 1.1

Input Data file:- the experiment input file is in for dataset 1.1 The Detail structure is explain in annexure-II

XML file Structure for Data Set 1.1

Dataset	Number of node at 1st Level (Assume root level no 0)	Max fan-out (maximum node at a particular level)	Depth of the file	Total number of nodes
SET 1.1 :- Data Centric File : CD.XML				
SET 1.1.1	10 - nodes	60	4	197
SET 1.1.2	100 - nodes	600	4	703
SET 1.1.3	1000 - nodes	6000	4	6301
SET 1.1.4	10000 - nodes	59952	4	69944
SET 1.1.5	50000 - nodes	300000	4	350140
SET 1.1.6	100000 – nodes	599994	4	699994

- SAXON : Query Execution time and Graph are listed below:-

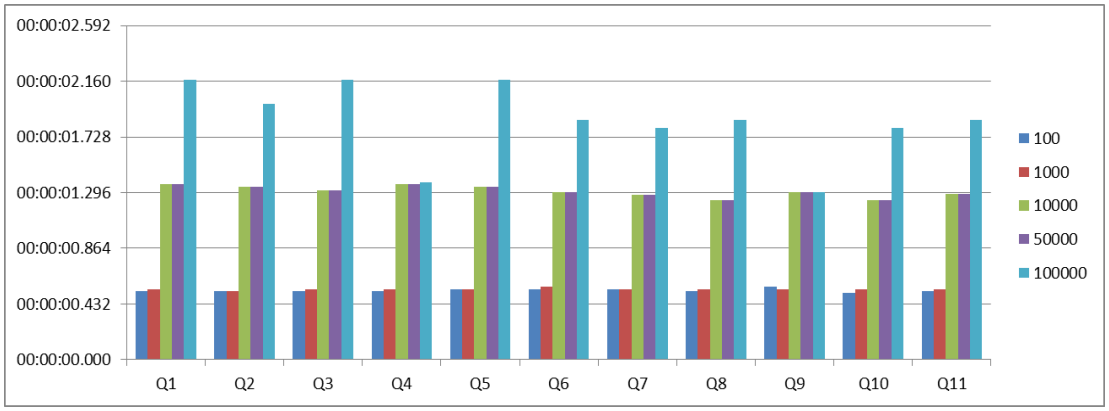
Table 5.13 SAXON: Query Execution Timing for Data Set 1.1

Query	100(Records)	1000(records)	10000(Records)	50000(Records)	100000 (Records)
Q1	00:00:00.531	00:00:00.547	00:00:01.359	00:00:01.000	00:00:02.172
Q2	00:00:00.531	00:00:00.531	00:00:01.344	00:00:01.344	00:00:01.984
Q3	00:00:00.531	00:00:00.547	00:00:01.313	00:00:01.313	00:00:02.172
Q4	00:00:00.532	00:00:00.547	00:00:01.359	00:00:01.359	00:00:01.375
Q5	00:00:00.547	00:00:00.547	00:00:01.344	00:00:01.344	00:00:02.172
Q6	00:00:00.547	00:00:00.563	00:00:01.297	00:00:01.297	00:00:01.859
Q7	00:00:00.547	00:00:00.547	00:00:01.281	00:00:01.281	00:00:01.797
Q8	00:00:00.531	00:00:00.547	00:00:01.234	00:00:01.234	00:00:01.860
Q9	00:00:00.563	00:00:00.547	00:00:01.297	00:00:01.297	00:00:01.297
Q10	00:00:00.516	00:00:00.547	00:00:01.235	00:00:01.235	00:00:01.797
Q11	00:00:00.531	00:00:00.547	00:00:01.282	00:00:01.282	00:00:01.859
Q12	00:00:00.547	00:00:00.578	00:00:01.313	00:00:01.313	00:00:02.000
Q13	00:00:00.563	00:00:00.563	00:00:01.422	00:00:01.422	00:00:02.250
Q14	00:00:00.531	00:00:00.578	00:00:01.375	00:00:01.375	00:00:01.937
Q15	00:00:00.547	00:00:00.578	00:00:01.297	00:00:01.297	00:00:01.937
Q16	00:00:00.547	00:00:00.563	00:00:01.344	00:00:01.344	00:00:02.000
Q17	00:00:00.547	00:00:00.547	00:00:01.375	00:00:01.375	00:00:01.985
Q18	00:00:00.547	00:00:00.563	00:00:01.250	00:00:01.250	00:00:01.813

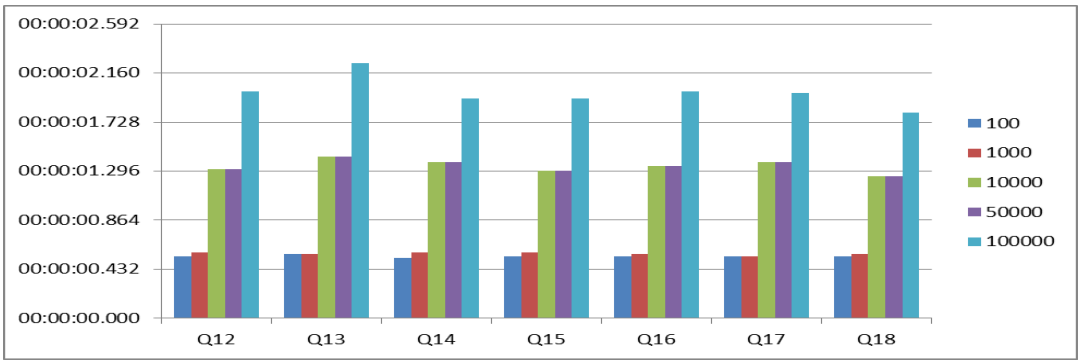
Results of SAXON:-

Graph5.4: SAXON: Query Execution for Data Set 1.1

1. SAXON: Query Execution For Data Set 1.1 (Query 1 - Query 11- Axis Type of Query)



SAXON: Query Execution For Data Set 1.1 (Query 12 - Query 18-FLOWER Query)



- IBM DB2.9: XML-Native Storage :- Query Execution time and Graph are listed below:-

Table 5.14 IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 1.1

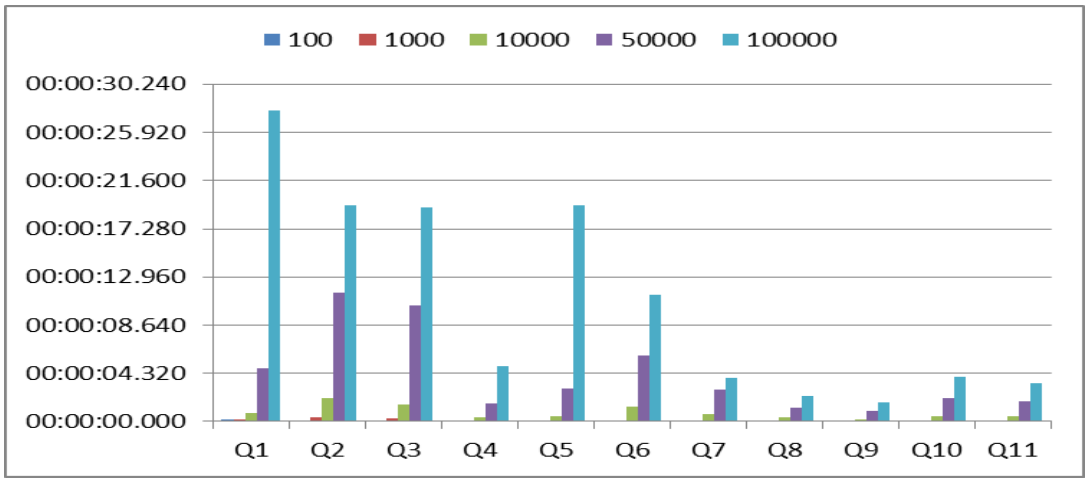
Query	100	1000	10000	50000	100000
Q1	00:00:00.172	00:00:00.203	00:00:00.781	00:00:04.781	00:00:27.859
Q2	00:00:00.109	00:00:00.359	00:00:02.078	00:00:11.594	00:00:19.344
Q3	00:00:00.125	00:00:00.250	00:00:01.563	00:00:10.375	00:00:19.234
Q4	00:00:00.078	00:00:00.094	00:00:00.375	00:00:01.578	00:00:04.953
Q5	00:00:00.094	00:00:00.141	00:00:00.454	00:00:02.922	00:00:19.360
Q6	00:00:00.094	00:00:00.093	00:00:01.375	00:00:05.906	00:00:11.375
Q7	00:00:00.063	00:00:00.140	00:00:00.703	00:00:02.891	00:00:03.922
Q8	00:00:00.063	00:00:00.094	00:00:00.344	00:00:01.234	00:00:02.266
Q9	00:00:00.063	00:00:00.079	00:00:00.234	00:00:00.922	00:00:01.735
Q10	00:00:00.063	00:00:00.110	00:00:00.515	00:00:02.078	00:00:04.016
Q11	00:00:00.063	00:00:00.109	00:00:00.484	00:00:01.843	00:00:03.453
Q12	00:00:00.078	00:00:00.265	00:00:01.359	00:00:09.109	00:00:16.719
Q13	00:00:00.157	00:00:00.484	00:00:02.078	00:00:09.328	00:00:18.781
Q14	00:00:00.078	00:00:00.250	00:00:01.359	00:00:05.844	00:00:11.250
Q15	00:00:00.079	00:00:00.250	00:00:01.281	00:00:05.500	00:00:10.515
Q16	00:00:00.094	00:00:00.359	00:00:01.937	00:00:08.718	00:00:16.641
Q17	00:00:00.079	00:00:00.281	00:00:01.281	00:00:05.438	00:00:10.422
Q18	00:00:00.078	00:00:00.250	00:00:01.328	00:00:05.469	00:00:10.704

Result of IBM DB2.9:-

Graph5.5: IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 1.1

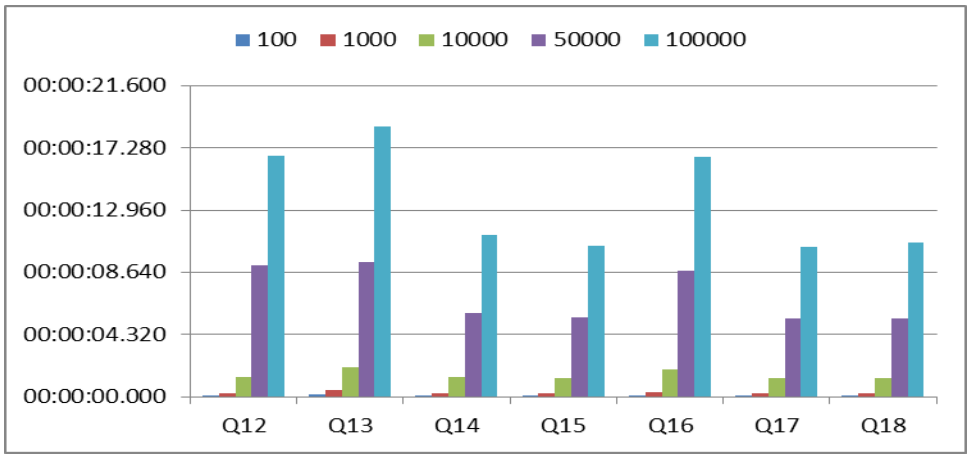
IBMDB2.9 Native Storage: Query Execution For Data Set 1.1

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 Native Storage: Query Execution For Data Set 1.1

(Query 12 - Query 18-FLOWER Query)



- IBM DB2.9: XML-Native Storage :- Query Execution time and Graph are listed below:-

Table 5.16 IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 1.1

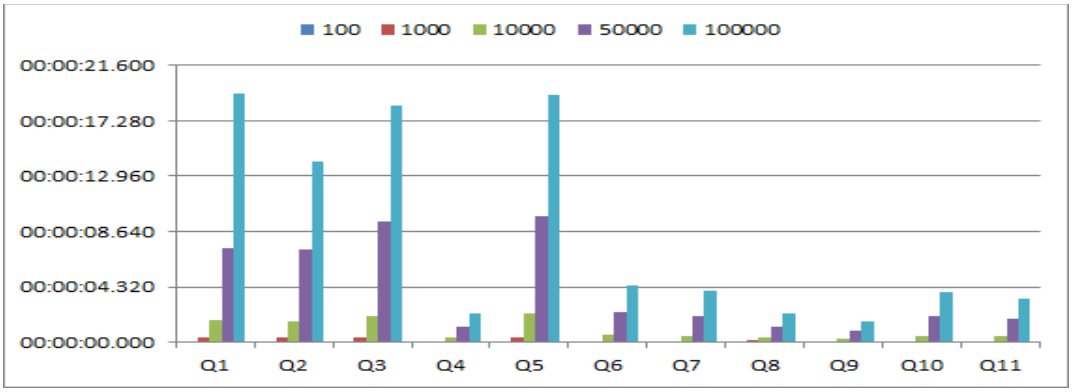
Query	10	100	1000	10000	50000	100000
Q1	00:00:00.062	00:00:00.109	00:00:00.344	00:00:01.734	00:00:07.360	00:00:19.359
Q2	00:00:00.062	00:00:00.094	00:00:00.344	00:00:01.578	00:00:07.187	00:00:14.140
Q3	00:00:00.047	00:00:00.094	00:00:00.375	00:00:02.062	00:00:09.437	00:00:18.515
Q4	00:00:00.062	00:00:00.063	00:00:00.078	00:00:00.360	00:00:01.188	00:00:02.297
Q5	00:00:00.063	00:00:00.109	00:00:00.391	00:00:02.234	00:00:09.812	00:00:19.344
Q6	00:00:00.063	00:00:00.094	00:00:00.110	00:00:00.547	00:00:02.359	00:00:04.406
Q7	00:00:00.063	00:00:00.094	00:00:00.110	00:00:00.453	00:00:02.032	00:00:03.984
Q8	00:00:00.063	00:00:00.094	00:00:00.125	00:00:00.359	00:00:01.203	00:00:02.250
Q9	00:00:00.063	00:00:00.063	00:00:00.078	00:00:00.219	00:00:00.860	00:00:01.625
Q10	00:00:00.063	00:00:00.063	00:00:00.109	00:00:00.500	00:00:02.047	00:00:03.907
Q11	00:00:00.062	00:00:00.047	00:00:00.094	00:00:00.437	00:00:01.828	00:00:03.375
Q12	00:00:00.063	00:00:00.079	00:00:00.250	00:00:01.360	00:00:05.687	00:00:11.062
Q13	00:00:00.063	00:00:00.078	00:00:00.250	00:00:01.407	00:00:05.765	00:00:11.250
Q14	00:00:00.062	00:00:00.062	00:00:00.093	00:00:00.359	00:00:05.781	00:00:11.187
Q15	00:00:00.063	00:00:00.079	00:00:00.250	00:00:01.328	00:00:05.422	00:00:10.704
Q16	00:00:00.062	00:00:00.078	00:00:00.250	00:00:01.281	00:00:05.500	00:00:10.546
Q17	00:00:00.063	00:00:00.078	00:00:00.234	00:00:01.265	00:00:05.422	00:00:10.391
Q18	00:00:00.063	00:00:00.078	00:00:00.250	00:00:01.328	00:00:05.391	00:00:10.187

Result of IBM DB2.9 XML-RDBMs Storage :-

Graph5.6: DB2- Query Execution: XML-RDBMS Storage for Data Set 1.1

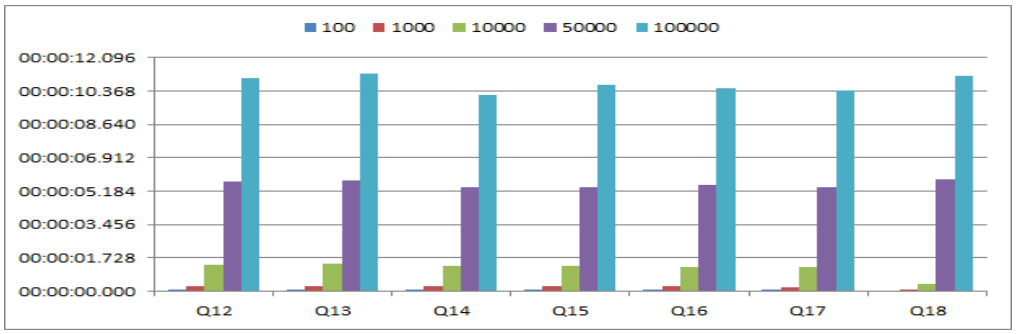
IBMDB2.9 XML-RDBMs Storage: Query Execution For Data Set 1.1

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 XML-RDBMS Storage: Query Execution For Data Set 1.1

(Query 12 - Query 18-FLOWER Query)



5.5.1.1 Experiment Result for Data Set1: Set 1.2:-

Input Data file:- the experiment input file is in table 5.13 for dataset 1.2, the file structure is presented in annexure-II

XML file Structure for Data Set 1.2

Dataset	Number of node at 1st Level (Assume root level no 0)	Max fan-out (maximum node at a particular level)	Depth of the file	Total number of nodes
SET 1.2 Data Centric File :- Whether .XML				
SET 1.2.1	10 - nodes	150	12	483
SET 1.2.2	100 - nodes	1575	12	5041
SET 1.2.3	1000 - nodes	15075	12	48241
SET 1.2.4	10000 – nodes	150000	12	471006
SET 1.2.5	50000 - nodes	749880	12	2349530
SET 1.2.6	100000 – nodes	1500045	12	5000145

- SAXON : Query Execution time and Graph are listed below:-

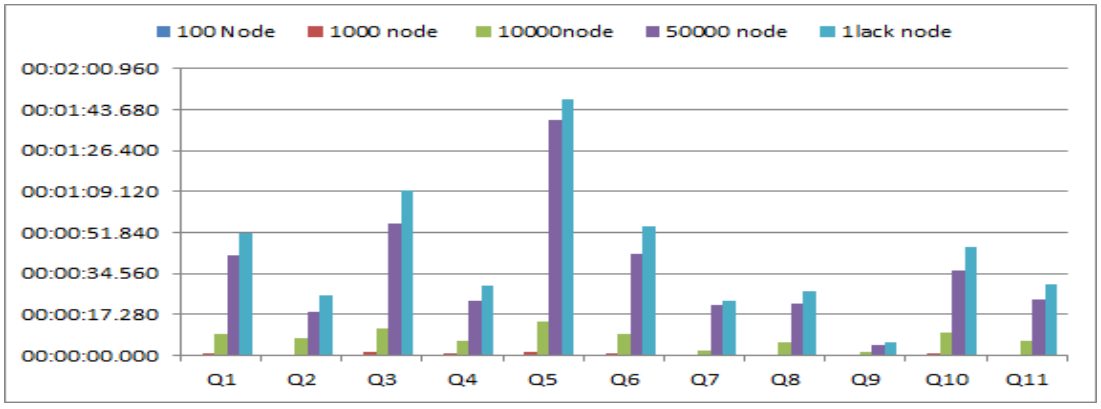
Table 5.17 SAXON: Query Execution Timing for Data Set 1.2

Query	10 node	100 Node	1000 node	10000node	50000 node	1lack node
Q1	00:00:00.062	00:00:00.188	00:00:00.672	00:00:09.312	00:00:42.141	00:00:51.453
Q2	00:00:00.328	00:00:00.141	00:00:00.625	00:00:07.063	00:00:18.375	00:00:25.438
Q3	00:00:00.078	00:00:00.297	00:00:01.625	00:00:11.422	00:00:55.688	00:01:09.766
Q4	00:00:00.156	00:00:00.141	00:00:00.672	00:00:06.141	00:00:23.156	00:00:29.297
Q5	00:00:00.078	00:00:00.187	00:00:01.609	00:00:14.078	00:01:39.203	00:01:48.359
Q6	00:00:00.063	00:00:00.188	00:00:00.812	00:00:09.156	00:00:43.016	00:00:54.438
Q7	00:00:00.063	00:00:00.062	00:00:00.265	00:00:02.015	00:00:21.047	00:00:23.062
Q8	00:00:00.063	00:00:00.125	00:00:00.547	00:00:05.453	00:00:21.750	00:00:27.203
Q9	00:00:00.047	00:00:00.063	00:00:00.157	00:00:01.234	00:00:04.172	00:00:05.406
Q10	00:00:00.078	00:00:00.000	00:00:00.985	00:00:09.860	00:00:36.061	00:00:45.921
Q11	00:00:00.047	00:00:00.125	00:00:00.594	00:00:06.156	00:00:23.782	00:00:29.938
Q12	00:00:00.047	00:00:00.141	00:00:00.219	00:00:03.640	00:00:05.813	00:00:09.453
Q13	00:00:00.062	00:00:00.063	00:00:00.453	00:00:04.296	00:00:28.843	00:00:30.765
Q14	00:00:00.078	00:00:00.094	00:00:00.437	00:00:03.578	00:00:12.859	00:00:16.437
Q15	00:00:00.047	00:00:00.140	00:00:00.297	00:00:01.922	00:00:05.000	00:00:09.296
Q16	00:00:00.046	00:00:00.078	00:00:00.313	00:00:02.234	00:00:07.610	00:00:09.844
Q17	00:00:00.047	00:00:00.062	00:00:00.422	00:00:01.515	00:00:18.735	00:00:20.219
Q18	00:00:00.047	00:00:00.062	00:00:00.171	00:00:01.484	00:00:05.079	00:00:06.594

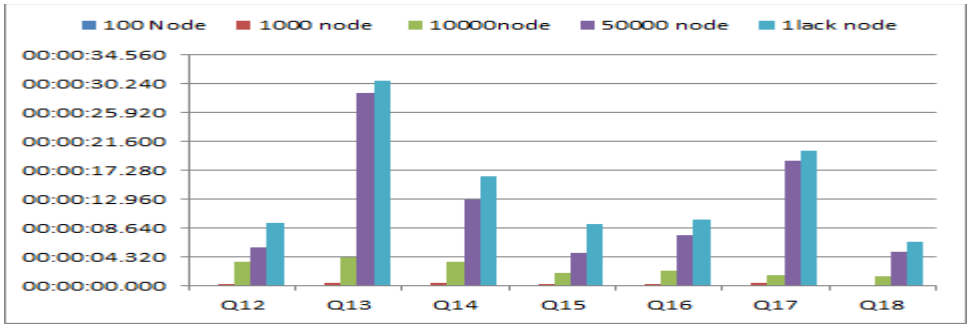
Result of SAON:-

Graph5.7: SAXON: Query Execution Storage for Data Set 1.2

SAXON: Query Execution For Data Set 1.2 (Query 1 - Query 11- Axis Type of Query)



SAXON: Query Execution For Data Set 1.2 (Query 12 - Query 18-FLOWER Query)



- IBM DB2.9 XML-Native Storage Query Execution time and Graph are listed below:-

Table 5.18 IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 1.2

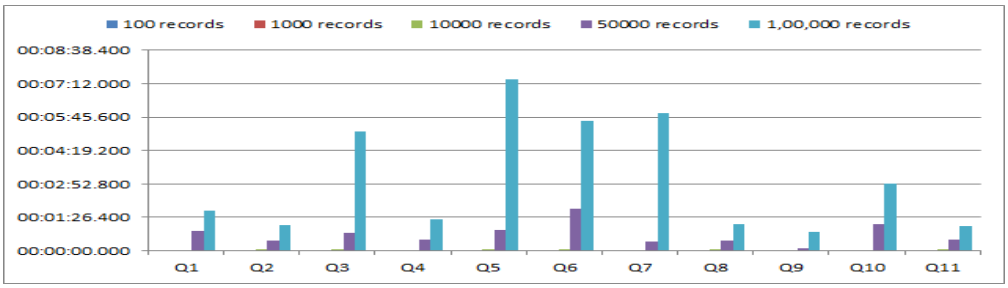
	10 record	100 records	1000 records	10000 records	50000 records	1,00,000 records
Q1	00:00:00.062	00:00:00.250	00:00:01.610	00:00:00.015	00:00:51.453	00:01:44.484
Q2	00:00:00.125	00:00:00.360	00:00:01.219	00:00:03.938	00:00:25.438	00:01:06.938
Q3	00:00:00.094	00:00:00.125	00:00:00.500	00:00:03.797	00:00:45.921	00:05:09.672
Q4	00:00:00.109	00:00:00.125	00:00:00.344	00:00:02.188	00:00:29.297	00:01:20.203
Q5	00:00:00.093	00:00:00.125	00:00:00.454	00:00:03.047	00:00:54.438	00:07:24.094
Q6	00:00:00.094	00:00:00.110	00:00:00.484	00:00:03.562	00:01:48.359	00:05:35.187
Q7	00:00:00.094	00:00:00.125	00:00:00.500	00:00:02.156	00:00:23.062	00:05:55.984
Q8	00:00:00.078	00:00:00.141	00:00:00.313	00:00:03.062	00:00:27.203	00:01:08.000
Q9	00:00:00.078	00:00:00.094	00:00:00.532	00:00:01.328	00:00:05.406	00:00:48.828
Q10	00:00:00.079	00:00:00.109	00:00:00.422	00:00:02.500	00:01:09.766	00:02:53.297
Q11	00:00:00.109	00:00:00.313	00:00:01.656	00:00:04.047	00:00:29.938	00:01:03.141
Q12	00:00:00.688	00:00:00.219	00:00:00.703	00:00:08.094	00:00:09.453	00:01:01.516
Q13	00:00:00.110	00:00:00.125	00:00:00.532	00:00:03.656	00:00:16.437	00:01:20.156
Q14	00:00:00.094	00:00:00.094	00:00:00.343	00:00:08.203	00:00:30.765	00:01:05.563
Q15	00:00:00.078	00:00:00.093	00:00:00.281	00:00:01.719	00:00:06.594	00:00:48.234
Q16	00:00:00.094	00:00:00.110	00:00:00.438	00:00:02.828	00:00:09.844	00:00:54.515
Q17	00:00:00.078	00:00:00.094	00:00:00.250	00:00:01.453	00:00:20.219	00:00:50.203
Q18	00:00:00.297	00:00:00.110	00:00:00.250	00:00:01.453	00:00:09.296	00:00:48.890

Result of IBM DB2.9

Graph5.8: DB2.9- XML Native Storage Query Execution for Data Set 1.2

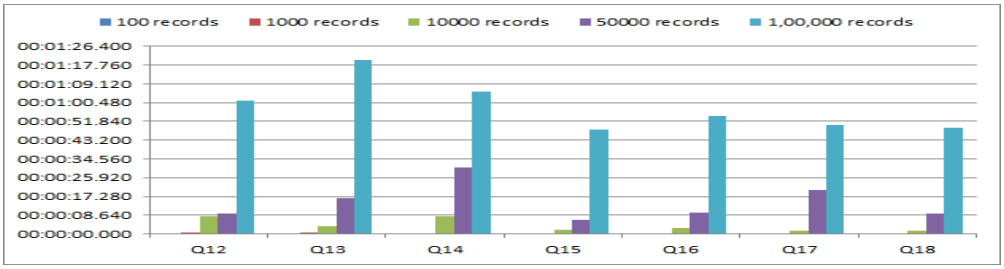
IBMDB2.9 Native Storage: Query Execution For Data Set 1.2

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 Native Storage: Query Execution For Data Set 1.2

(Query 12 - Query 18-FLOWER Query)



- IBM DB2.9 XML-RDBMs Storage Query execution time and graph are listed below

Table 5.19 IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 1.2

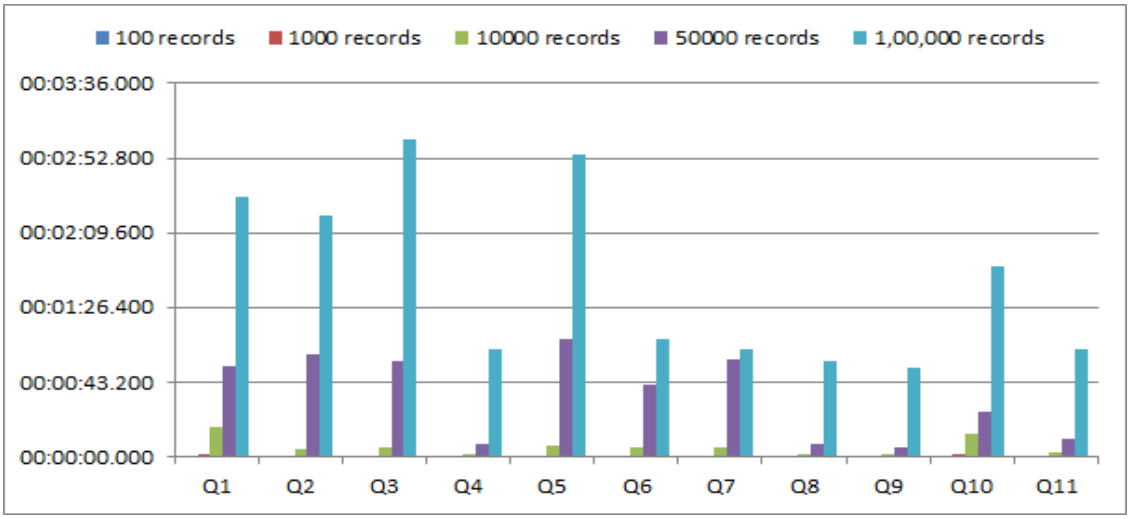
	10 record	100 records	1000 records	10000 records	50000 records	1,00,000 records
Q1	00:00:00.078	00:00:00.297	00:00:01.140	00:00:17.250	00:00:52.672	00:02:30.672
Q2	00:00:00.078	00:00:00.125	00:00:00.891	00:00:04.110	00:00:59.407	00:02:19.500
Q3	00:00:00.078	00:00:00.156	00:00:00.641	00:00:05.407	00:00:55.125	00:03:03.578
Q4	00:00:00.062	00:00:00.063	00:00:00.188	00:00:01.703	00:00:07.703	00:01:01.844
Q5	00:00:00.062	00:00:00.141	00:00:00.657	00:00:06.157	00:01:07.641	00:02:54.329
Q6	00:00:00.062	00:00:00.125	00:00:00.688	00:00:05.094	00:00:42.125	00:01:08.000
Q7	00:00:00.062	00:00:00.109	00:00:00.672	00:00:05.032	00:00:56.641	00:01:01.844
Q8	00:00:00.047	00:00:00.063	00:00:00.188	00:00:01.672	00:00:07.515	00:00:54.953
Q9	00:00:00.047	00:00:00.063	00:00:00.156	00:00:01.187	00:00:05.172	00:00:51.125
Q10	00:00:00.094	00:00:00.266	00:00:01.594	00:00:13.703	00:00:25.984	00:01:50.313
Q11	00:00:00.047	00:00:00.078	00:00:00.328	00:00:02.250	00:00:10.453	00:01:02.531
Q12	00:00:00.063	00:00:00.078	00:00:00.281	00:00:02.078	00:00:09.687	00:00:58.359
Q13	00:00:00.063	00:00:00.125	00:00:00.453	00:00:03.609	00:00:17.047	00:01:19.968
Q14	00:00:00.063	00:00:00.078	00:00:00.266	00:00:02.047	00:00:12.329	00:01:08.266
Q15	00:00:00.063	00:00:00.078	00:00:00.313	00:00:02.203	00:00:09.922	00:00:43.547
Q16	00:00:00.046	00:00:00.079	00:00:00.313	00:00:02.235	00:00:16.187	00:00:50.344
Q17	00:00:00.047	00:00:00.063	00:00:00.187	00:00:01.625	00:00:06.922	00:00:50.641
Q18	00:00:00.047	00:00:00.062	00:00:00.188	00:00:01.609	00:00:06.922	00:00:39.454

Result of IBM DB2.9

Graph5.9: DB2.9- XML-RDBMs Storage Query Execution for Data Set 1.2

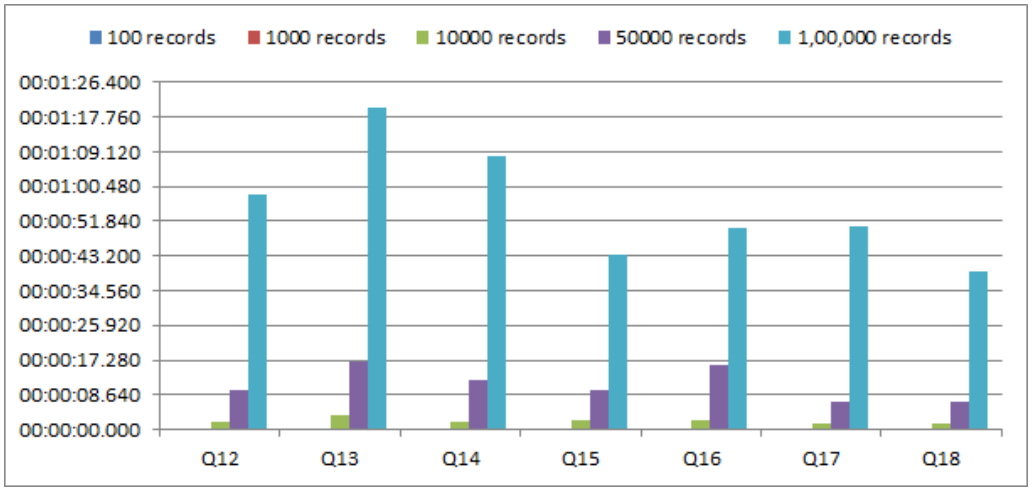
IBMDB2.9 XML-RDBMs Storage: Query Execution For Data Set 1.2

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 XML-RDBMs: Query Execution For Data Set 1.2

(Query 12 - Query 18-FLOWER Query)



Experiment Result for Data Set2: Set 2.1

Input File:- the experiment input file for dataset 2.1

XML file Structure for Data Set 2.1

Dataset	Number of node at 1st Level (Assume root level no 0)	Max fan-out (maximum node at a particular level)	Depth of the file	Total number of nodes
SET 2.1 Mix mode File :- Chp.xml				
SET 2.1.1	10 - nodes	40	5	71
SET 2.1.2	100 - nodes	402	5	699
SET 3.1.3	1000 - nodes	3920	5	6861
SET 3.1.4	10000 - nodes	39200	5	68601
SET 3.1.5	50000 - nodes	196000	5	343001
SET 3.1.6	100000 - nodes	392000	5	686001

- SAXON: Query Execution Timing for Data Set 2.1 are listed below:-

Table 5.20 SAXON: Query Execution Timing for Data Set 2.1

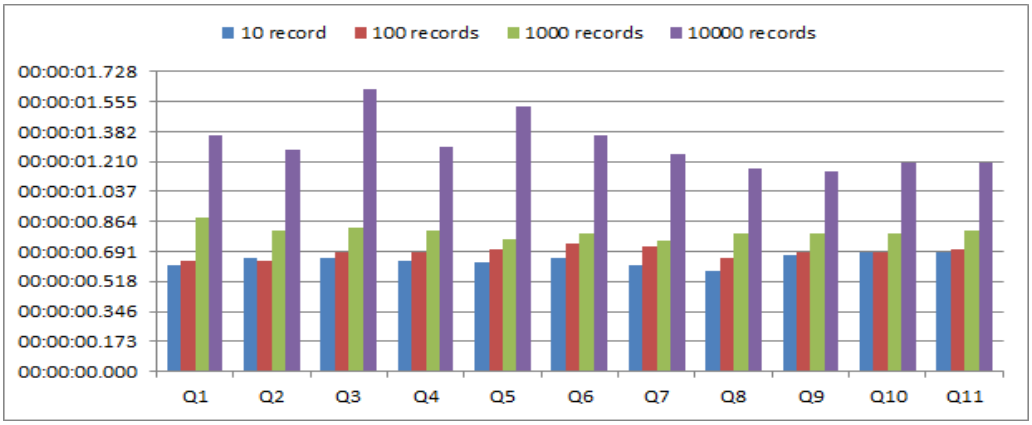
Query	10 record	100 records	1000 records	10000 records
Q1	00:00:00.610	00:00:00.641	00:00:00.890	00:00:01.360
Q2	00:00:00.656	00:00:00.641	00:00:00.813	00:00:01.281
Q3	00:00:00.656	00:00:00.688	00:00:00.828	00:00:01.625
Q4	00:00:00.641	00:00:00.687	00:00:00.813	00:00:01.297
Q5	00:00:00.625	00:00:00.703	00:00:00.766	00:00:01.531
Q6	00:00:00.656	00:00:00.734	00:00:00.796	00:00:01.360
Q7	00:00:00.610	00:00:00.719	00:00:00.750	00:00:01.250
Q8	00:00:00.578	00:00:00.656	00:00:00.797	00:00:01.172
Q9	00:00:00.672	00:00:00.687	00:00:00.797	00:00:01.156
Q10	00:00:00.687	00:00:00.688	00:00:00.797	00:00:01.204
Q11	00:00:00.687	00:00:00.703	00:00:00.812	00:00:01.203
Q12	00:00:00.688	00:00:00.735	00:00:00.860	00:00:01.422
Q13	00:00:00.688	00:00:00.734	00:00:00.828	00:00:01.422
Q14	00:00:00.688	00:00:00.734	00:00:00.859	00:00:01.203
Q15	00:00:00.703	00:00:00.719	00:00:00.875	00:00:01.312
Q16	00:00:00.703	00:00:00.703	00:00:00.812	00:00:01.281
Q17	00:00:00.703	00:00:00.719	00:00:00.844	00:00:01.312
Q18	00:00:00.704	00:00:00.704	00:00:00.844	00:00:01.203

Result SAXON

Graph5.10: SAXON Query Execution Time for Dataset 2.1

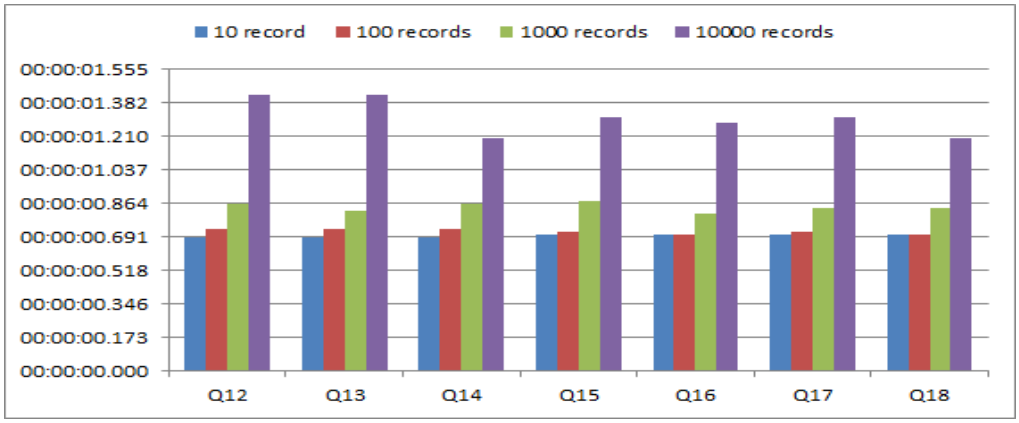
SAXON: Query Execution For Data Set 2.1

(Query 1 - Query 11- Axis Type of Query)



SAXON: Query Execution For Data Set 2.1

(Query 12 - Query 18-FLOWER Query)



- IBM DB2.9 XML-Native Storage Query Execution Timing and graph of Data Set 2.1

Table 5.21 IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 2.1

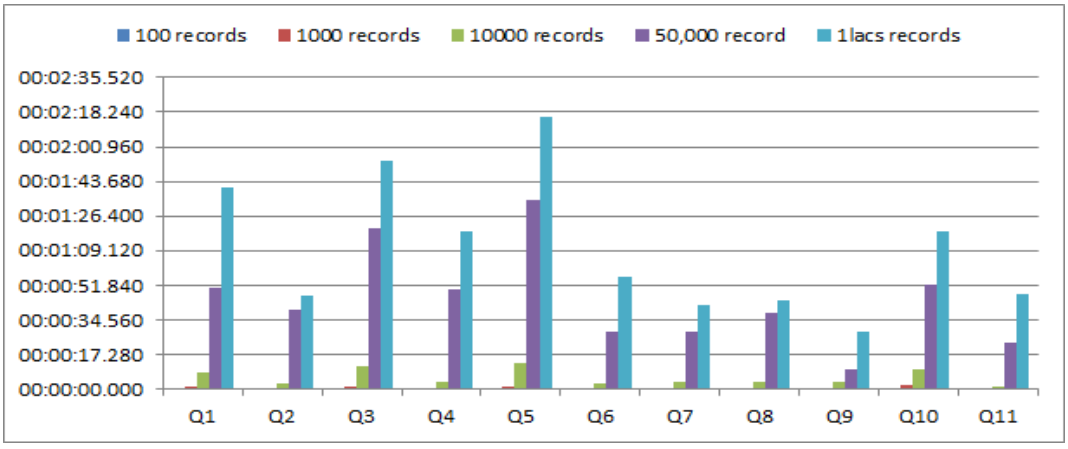
Query	10 record	100 records	1000 records	10000 records	50,000 record	1lacs records
Q1	00:00:00.062	00:00:00.109	00:00:00.875	00:00:08.078	00:00:50.500	00:01:40.766
Q2	00:00:00.062	00:00:00.094	00:00:00.218	00:00:03.125	00:00:39.656	00:00:46.469
Q3	00:00:00.422	00:00:00.141	00:00:01.141	00:00:11.156	00:01:20.172	00:01:53.812
Q4	00:00:00.063	00:00:00.078	00:00:00.672	00:00:03.938	00:00:49.656	00:01:18.656
Q5	00:00:00.157	00:00:00.391	00:00:01.140	00:00:12.968	00:01:34.234	00:02:15.547
Q6	00:00:00.234	00:00:00.125	00:00:00.578	00:00:03.000	00:00:28.531	00:00:55.750
Q7	00:00:00.062	00:00:00.188	00:00:00.203	00:00:03.312	00:00:28.641	00:00:41.953
Q8	00:00:00.125	00:00:00.172	00:00:00.390	00:00:03.422	00:00:38.000	00:00:44.016
Q9	00:00:00.203	00:00:00.297	00:00:00.406	00:00:03.359	00:00:09.860	00:00:28.375
Q10	00:00:00.125	00:00:00.282	00:00:01.703	00:00:09.860	00:00:51.765	00:01:18.641
Q11	00:00:00.062	00:00:00.078	00:00:00.188	00:00:01.219	00:00:23.469	00:00:47.219
Q12	00:00:00.063	00:00:00.094	00:00:00.781	00:00:05.609	00:00:31.890	00:00:43.703
Q13	00:00:00.063	00:00:00.407	00:00:00.422	00:00:08.250	00:00:40.859	00:01:12.250
Q14	00:00:00.063	00:00:00.110	00:00:00.578	00:00:04.828	00:00:47.094	
Q15	00:00:00.078	00:00:00.125	00:00:00.578	00:00:05.609	00:00:19.234	00:00:46.094
Q16	00:00:00.062	00:00:00.125	00:00:00.625	00:00:04.657	00:00:14.875	00:00:50.890
Q17	00:00:00.141	00:00:00.125	00:00:00.562	00:00:04.734	00:00:31.704	00:00:40.313
Q18	00:00:00.062	00:00:00.266	00:00:00.937	00:00:04.625	00:00:13.593	00:00:13.672

Result IBM DB2.9:-

Graph5.11: IBM 2.9 XML-Native Storage e for Dataset 2.1

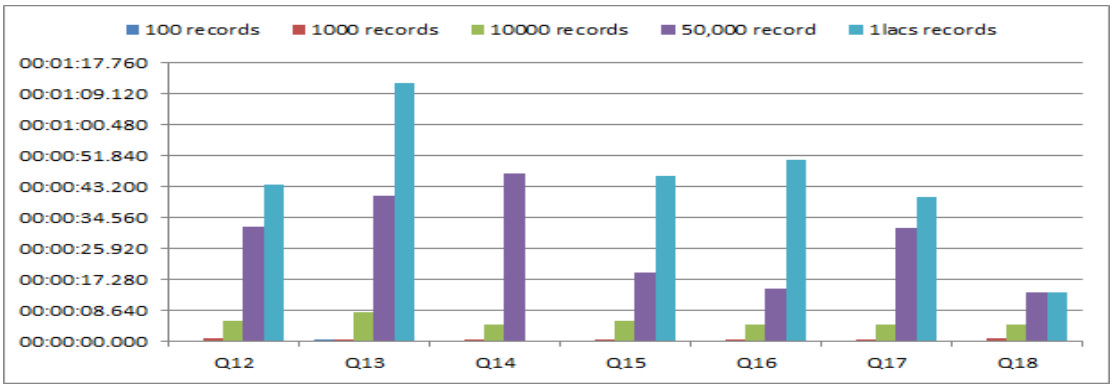
IBMDB2.9 Native Storage: Query Execution For Data Set 2.1

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 Native Storage: Query Execution For Data Set 2.1

(Query 12 - Query 18-FLOWER Query)



- IBM DB2.9 XML-RDBMs Storage :- Query Execution Timing and graph for Data Set 2.1 are listed below:-

Table 5.21 IBM DB2.9 XML-RDBMs Storage Query Execution Timing for Data Set 2.1

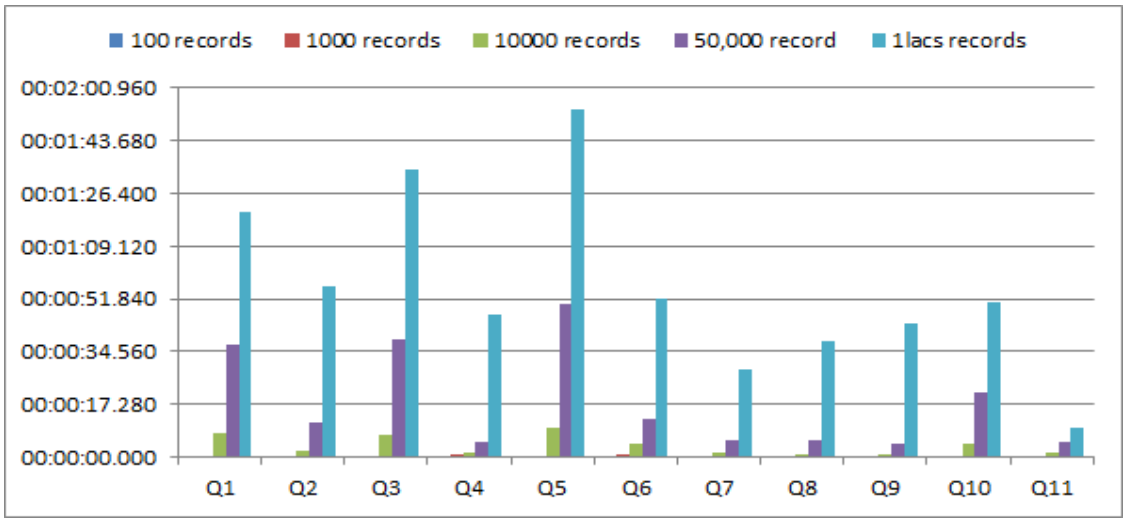
Query	10 record	100 records	1000 records	10000 records	50,000 record	1lacs records
Q1	00:00:00.047	00:00:00.219	00:00:00.250	00:00:07.953	00:00:37.062	00:01:20.172
Q2	00:00:00.250	00:00:00.125	00:00:00.328	00:00:02.375	00:00:11.547	00:00:55.750
Q3	00:00:00.140	00:00:00.313	00:00:00.359	00:00:07.468	00:00:38.875	00:01:34.234
Q4	00:00:00.078	00:00:00.204	00:00:00.781	00:00:01.313	00:00:04.921	00:00:46.469
Q5	00:00:00.062	00:00:00.094	00:00:00.484	00:00:09.375	00:00:50.297	00:01:53.812
Q6	00:00:00.063	00:00:00.188	00:00:00.797	00:00:04.515	00:00:12.719	00:00:51.765
Q7	00:00:00.078	00:00:00.156	00:00:00.156	00:00:01.250	00:00:05.359	00:00:28.641
Q8	00:00:00.047	00:00:00.063	00:00:00.156	00:00:01.203	00:00:05.671	00:00:38.000
Q9	00:00:00.063	00:00:00.063	00:00:00.125	00:00:00.891	00:00:04.656	00:00:44.016
Q10	00:00:00.063	00:00:00.141	00:00:00.547	00:00:04.125	00:00:21.016	00:00:50.500
Q11	00:00:00.062	00:00:00.078	00:00:00.313	00:00:01.328	00:00:05.016	00:00:09.860
Q12	00:00:00.062	00:00:00.078	00:00:00.203	00:00:01.562	00:00:07.047	00:00:31.704
Q13	00:00:00.047	00:00:00.094	00:00:00.453	00:00:03.454	00:00:19.234	00:00:47.094
Q14	00:00:00.063	00:00:00.078	00:00:00.313	00:00:01.922	00:00:09.000	00:00:15.875
Q15	00:00:00.063	00:00:00.079	00:00:00.204	00:00:03.312	00:00:07.531	00:00:31.890
Q16	00:00:00.063	00:00:00.078	00:00:00.250	00:00:01.640	00:00:14.875	00:00:40.859
Q17	00:00:00.063	00:00:00.078	00:00:00.172	00:00:01.172	00:00:05.016	00:00:19.234
Q18	00:00:00.031	00:00:00.062	00:00:00.156	00:00:01.156	00:00:05.000	00:00:13.593

Result IBM DB2.9:-

Graph5.12: IBM 2.9 XML-RDBMs Storage e for Dataset 2.1

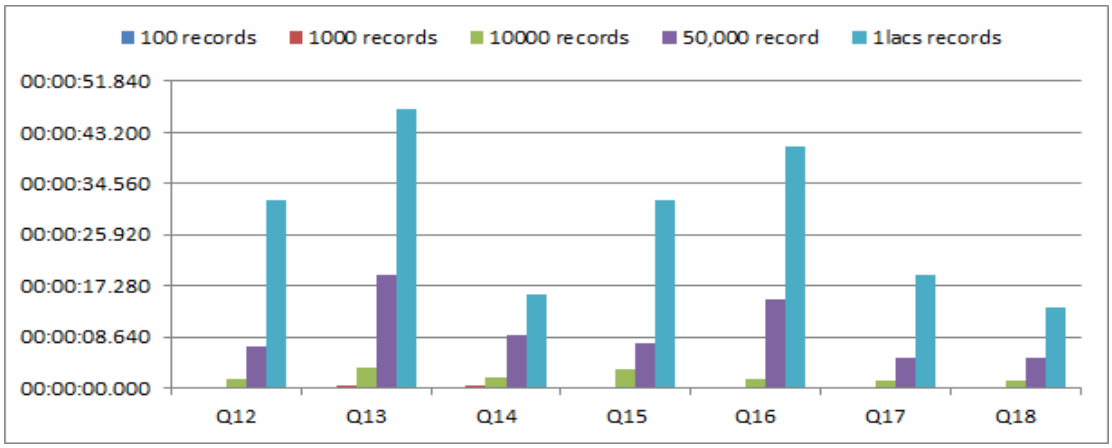
IBMDB2.9 XML-RDBMs Storage: Query Execution For Data Set 2.1

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 XML-RDBMs: Query Execution For Data Set 2.1

(Query 12 - Query 18-FLOWER Query)



Experiment Result for Data Set2: Set 2.2

Input File:- The experiment input file for dataset 2.2

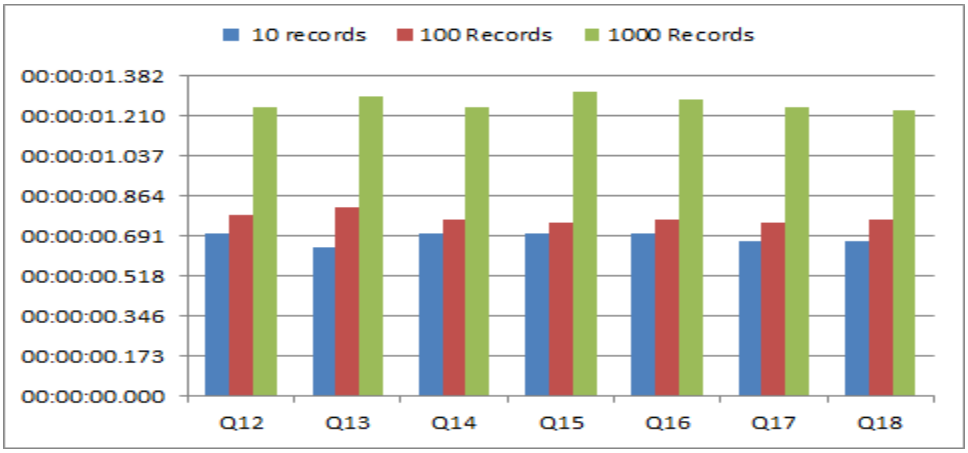
Dataset	Number of node at 1st Level (Assume root level no 0)	Max fan-out (maximum node at a particular level)	Depth of the file	Total number of nodes
SET 2.1 Mix mode File :- MXDB.xml				
SET 2.2.1	10 - nodes	40	10	181
SET 2.2.2	100 - nodes	380	10	1711
SET 2.2.3	1000 - nodes	3800	10	17101
SET 2.2.4	10,000 - nodes	38000	10	171001
SET 2.2.5	50000 - nodes	190000	10	855001
SET 2.2.6	100000 – nodes	380000	10	1710001

- SAXON: Query Execution Timing and graphs are for Data Set 2.2 are listed below:-

Table 5.23SAXON: Query Execution Timing for Data Set 2.2

Query	10 records	100 Records	1000 Records
Q1	00:00:00.671	00:00:00.843	00:00:02.297
Q2	00:00:00.672	00:00:00.750	00:00:01.219
Q3	00:00:00.703	00:00:00.828	00:00:04.984
Q4	00:00:00.703	00:00:00.781	00:00:02.125
Q5	00:00:00.672	00:00:00.750	00:00:04.969
Q6	00:00:00.734	00:00:00.782	00:00:02.516
Q7	00:00:00.688	00:00:00.719	00:00:01.625
Q8	00:00:00.734	00:00:00.750	00:00:01.250
Q9	00:00:00.687	00:00:00.766	00:00:01.234
Q10	00:00:00.672	00:00:00.750	00:00:01.281
Q11	00:00:00.672	00:00:00.750	00:00:01.234
Q12	00:00:00.703	00:00:00.781	00:00:01.250
Q13	00:00:00.641	00:00:00.813	00:00:01.297
Q14	00:00:00.703	00:00:00.766	00:00:01.250
Q15	00:00:00.703	00:00:00.750	00:00:01.313
Q16	00:00:00.672	00:00:00.765	00:00:01.235
Q17	00:00:00.672	00:00:00.750	00:00:01.250
Q18	00:00:00.703	00:00:00.765	00:00:01.282

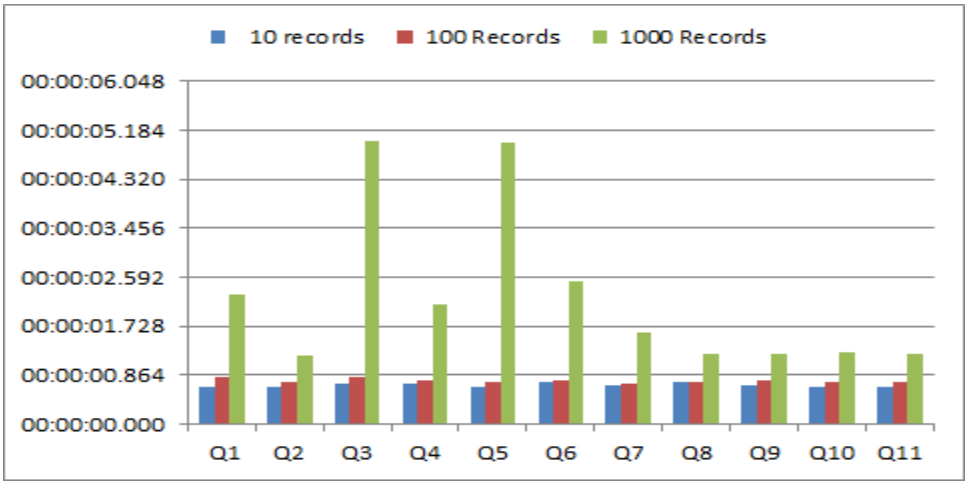
SAXON: Query Execution For Data Set 2.2(Query 12 - Query 18-FLOWER Query)



Result SAXON

Graph5.13: SAXON Query Execution Time for Dataset 2.2

SAXON: Query Execution For Data Set 2.2 (Query 1 - Query 11- Axis Type of Query)



- IBM DB2.9 XML-Native Storage the Graph and query execution time are noted below:-

Table 5.24 IBM DB2.9 XML-Native Storage Query Execution Timing for Data Set 2.2

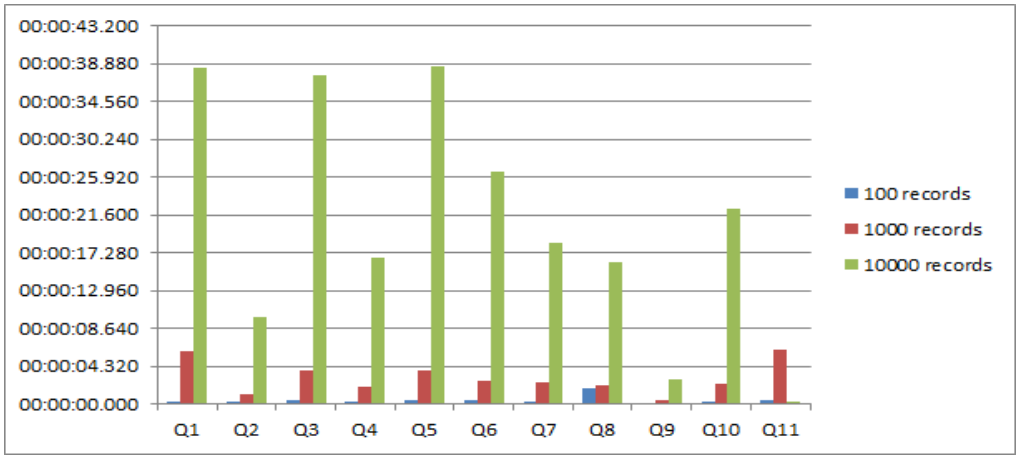
Query	10records	100 records	1000 records	10000 records	50000 records	000000 record
Q1	00:00:00.093	00:00:00.265	00:00:06.062	00:00:38.407	00:02:38.000	00:05:24.297
Q2	00:00:00.109	00:00:00.218	00:00:01.125	00:00:09.875	00:04:17.187	00:03:04.875
Q3	00:00:00.719	00:00:00.469	00:00:03.766	00:00:37.609	00:02:34.360	00:08:28.500
Q4	00:00:00.125	00:00:00.328	00:00:02.000	00:00:16.687	00:02:03.922	00:04:42.282
Q5	00:00:00.109	00:00:00.500	00:00:03.828	00:00:38.593	00:02:26.000	00:06:40.141
Q6	00:00:00.250	00:00:00.375	00:00:02.688	00:00:26.594	00:00:02.547	00:04:37.844
Q7	00:00:00.063	00:00:00.297	00:00:02.391	00:00:18.328	00:01:13.141	00:00:57.953
Q8	00:00:00.453	00:00:01.718	00:00:02.172	00:00:16.266	00:00:01.453	00:04:42.282
Q9	00:00:00.063	00:00:00.109	00:00:00.469	00:00:02.797	00:00:46.172	00:01:41.610
Q10	00:00:00.125	00:00:00.344	00:00:02.359	00:00:22.359	00:04:02.203	00:11:47.438
Q11	00:00:00.344	00:00:00.516	00:00:06.188	00:00:00.344	00:02:17.922	00:04:44.000
Q12	00:00:00.062	00:00:00.125	00:00:00.891	00:00:08.234	00:02:10.062	00:06:54.515
Q13	00:00:00.062	00:00:00.235	00:00:01.453	00:00:14.453	00:02:53.641	00:06:53.312
Q14	00:00:00.063	00:00:00.219	00:00:00.891	00:00:11.063	00:00:42.718	00:03:02.750
Q15	00:00:00.078	00:00:00.140	00:00:00.547	00:00:04.610	00:00:41.203	00:01:44.094
Q16	00:00:00.062	00:00:00.219	00:00:00.547	00:00:12.813	00:00:46.735	00:01:39.610
Q17	00:00:00.047	00:00:00.094	00:00:00.906	00:00:09.891	00:00:40.016	00:00:05.500
Q18	00:00:00.047	00:00:00.093	00:00:00.328	00:00:02.141	00:00:43.796	00:00:38.297

Result IBM DB2.9:-

Graph5.15: IBM 2.9 XML-Native Storage e for Dataset 2.2

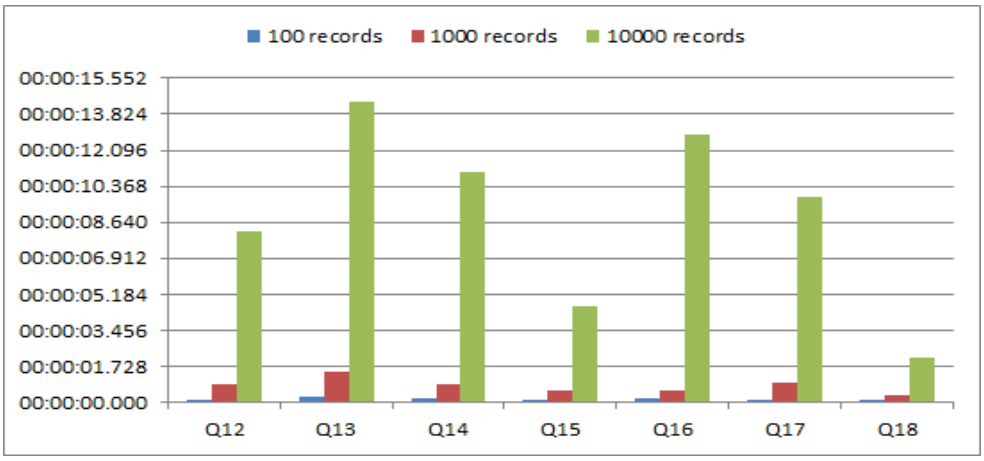
IBMDB2.9 Native Storage: Query Execution For Data Set 2.2

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 Native Storage: Query Execution For Data Set 2.2

(Query 1 - Query 11- Axis Type of Query)



- IBM 2.9 XML-RDBMs Storage Query Execution Timing and graph for Data Set 2.2

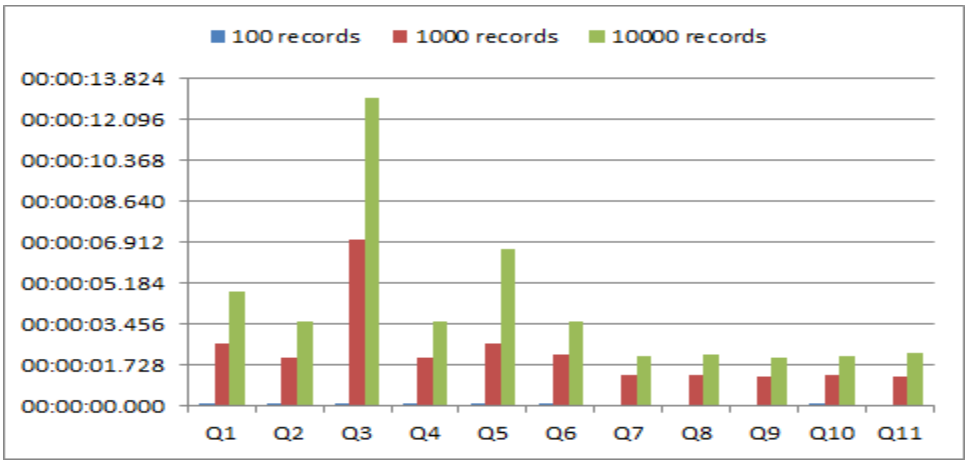
Table 5.25 IBM 2.9 XML-RDBMs Query Execution Timing for Data Set 2.2

Query	100 records	1000 records	10000 records	50000 records	1000000 records
Q1	00:00:00.109	00:00:02.625	00:00:04.813	00:02:45.563	00:08:14.359
Q2	00:00:00.078	00:00:02.062	00:00:03.578	00:01:13.141	00:02:38.813
Q3	00:00:00.078	00:00:07.000	00:00:13.032	00:01:56.937	00:05:12.531
Q4	00:00:00.078	00:00:02.062	00:00:03.578	00:04:15.391	00:22:42.500
Q5	00:00:00.109	00:00:02.625	00:00:06.609	00:02:34.360	00:06:21.125
Q6	00:00:00.078	00:00:02.171	00:00:03.578	00:00:03.938	00:07:53.594
Q7	00:00:00.063	00:00:01.296	00:00:02.078	00:00:38.984	00:00:41.844
Q8	00:00:00.062	00:00:01.312	00:00:02.156	00:00:01.250	00:05:12.531
Q9	00:00:00.047	00:00:01.250	00:00:02.062	00:00:39.297	00:01:57.938
Q10	00:00:00.093	00:00:01.297	00:00:02.078	00:01:51.781	00:05:28.891
Q11	00:00:00.062	00:00:01.250	00:00:02.219	00:01:55.797	00:04:46.578
Q12	00:00:00.063	00:00:01.437	00:00:01.969	00:00:04.219	00:01:03.687
Q13	00:00:00.078	00:00:01.985	00:00:02.328	00:01:32.578	00:03:17.703
Q14	00:00:00.063	00:00:02.000	00:00:02.125	00:00:39.594	00:01:26.297
Q15	00:00:00.047	00:00:01.031	00:00:01.984	00:00:29.813	00:01:34.437
Q16	00:00:00.047	00:00:01.047	00:00:02.062	00:00:31.531	00:01:27.875
Q17	00:00:00.047	00:00:01.032	00:00:01.844	00:00:03.984	00:00:23.938
Q18	00:00:00.047	00:00:00.922	00:00:01.828	00:00:32.672	00:00:32.156

Graph5.15: IBM 2.9 XML-RDBMs Storage e for Dataset 2.2

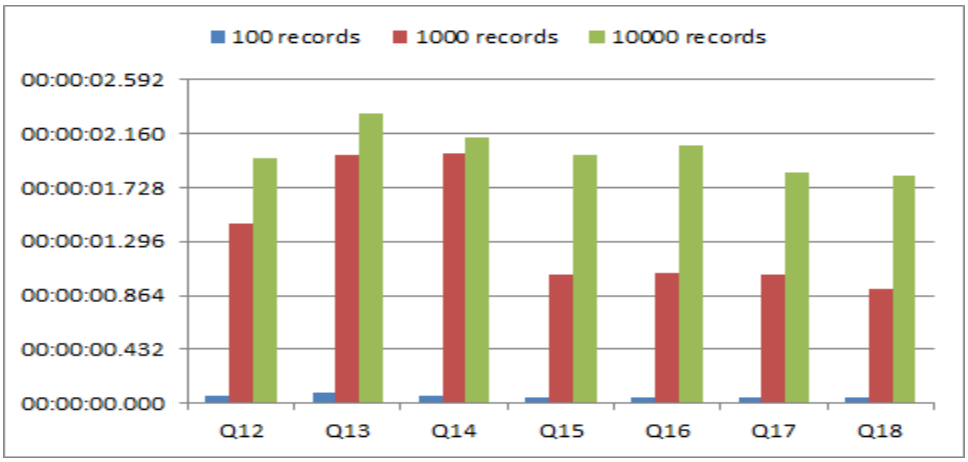
IBMDB2.9 XML-RDBMs: Query Execution For Data Set 2.2

(Query 1 - Query 11- Axis Type of Query)



IBMDB2.9 XML-RDBMs: Query Execution For Data Set 2.2

(Query 1 - Query 11- Axis Type of Query)



5.5.1.2 Analysis of Query execution time of SAXON and IBM DB2.9:-

The query execution time was depicting in millisecond. . The four set of XML file with different size were used for experiment. The data file size is varying from (10 records, 100 records, 1000 records, 10000 records, 500000 records, 100000 records). The observation and finding as follow:-

From all above graph , for ‘Axis’ type of query Q9 required minimum time for execution and Q3 required maximum time for query execution. From the query execution plan (which is depicted into Annexure -II) the following points are noted :-

- Query 3 and Query 1 used table-scan operation
- Query 2, Query 4 till Query 9 used index operator. i.e. ‘XISCAN’ operator.
- the index is used by and query1,

In ‘FLOWR’ type of queries Q18 required minimum time and Q13 required maximum time. Query measurement plan suggest in “FLOWER” type of query, retuning node not only depends upon index operation but also on selectivity value also. The selectivity value depends upon predicate value and XML structure. Query 18 was a simple chain path query; it returns nodes on simple XML path. Query 13 was a branch-path query. It returns a branch of XML tree. In query execution time Query18 required minimum time for query execution while Query13 required maximum time. The query execution timing summaries in following table 5.26 , are listed below:-

Table5.26: Query Execution Time Analysis

Experiments Set	Database File	Type Of Query	Minimum Query Timing (Query)	Maximum Query Timing (Query)
CD.XML				
SET 1.1.1	SAXON	Axis	Q 9	Q 5, Q3,Q1
SET 1.1.2	DB2		Q 9	Q 5, Q3,Q1
SET 1.1.1	SAXON	FLOWR	Q 18	Q 13
SET 1.1.2	DB2		Q 18	Q 13
WS.XML				
SET 1.2.1	SAXON	Axis	Q 9	Q 3,Q5
SET 1.2.2	DB2		Q 9	Q3,Q 5,Q1
SET 1.2.1	SAXON	FLOWR	Q 18	Q 13
SET 1.2.2	DB2		Q 18	Q 13
CHP.XML				
SET 1.1.1	SAXON	Axis	Q 9	Q 3
SET 1.1.2	DB2		Q 9	Q 4
SET 1.1.1	SAXON	FLOWR	Almost same time for every query execution	
SET 1.1.2	DB2		Q 18	Q 13
MXDB.XML				
SET 1.2.1	SAXON	Axis	Q 9	Q 3
SET 1.2.2	DB2		Q 9	Q 4
SET 1.2.1	SAXON	FLOWR	Almost same time for every query execution	
SET 1.2.2	DB2		Q 18	Q 13

Comparative analysis of IBM DB2.9 and SAXON :-

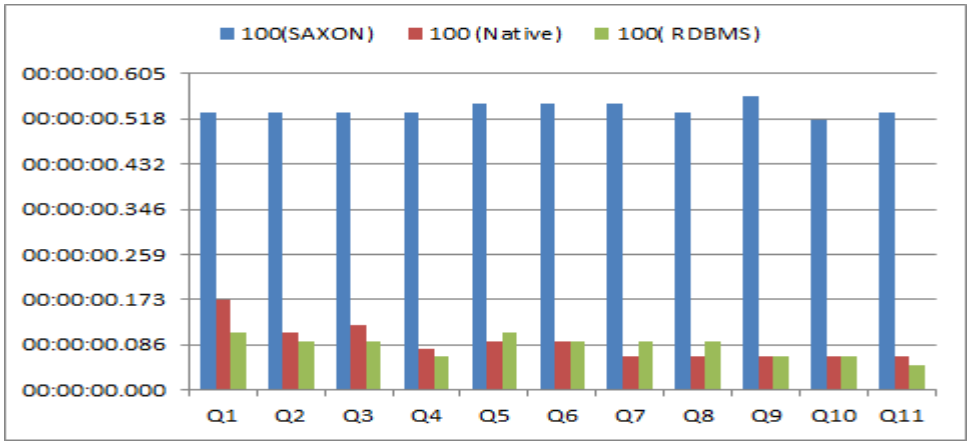
5.5.2 Comparative Study of Query Execution time SAXON and IBM DB2.9 :-

Table 5.27 : Comparative Query Execution Time for Data Se 1.1

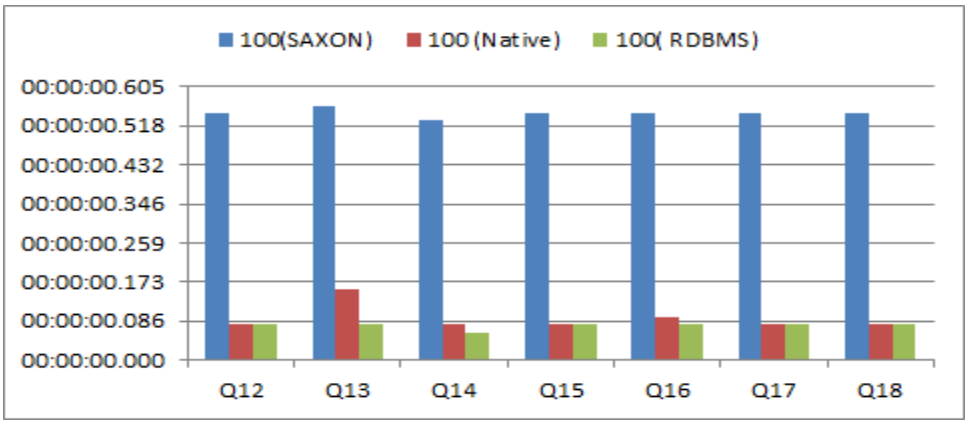
Query	100(SAXON)	100 (Native)	100(RDBMS)
Q1	00:00:00.531	00:00:00.172	00:00:00.109
Q2	00:00:00.531	00:00:00.109	00:00:00.094
Q3	00:00:00.531	00:00:00.125	00:00:00.094
Q4	00:00:00.532	00:00:00.078	00:00:00.063
Q5	00:00:00.547	00:00:00.094	00:00:00.109
Q6	00:00:00.547	00:00:00.094	00:00:00.094
Q7	00:00:00.547	00:00:00.063	00:00:00.094
Q8	00:00:00.531	00:00:00.063	00:00:00.094
Q9	00:00:00.563	00:00:00.063	00:00:00.063
Q10	00:00:00.516	00:00:00.063	00:00:00.063
Q11	00:00:00.531	00:00:00.063	00:00:00.047
Q12	00:00:00.547	00:00:00.078	00:00:00.079
Q13	00:00:00.563	00:00:00.157	00:00:00.078
Q14	00:00:00.531	00:00:00.078	00:00:00.062
Q15	00:00:00.547	00:00:00.079	00:00:00.079
Q16	00:00:00.547	00:00:00.094	00:00:00.078
Q17	00:00:00.547	00:00:00.079	00:00:00.078
Q18	00:00:00.547	00:00:00.078	00:00:00.078

- Graph5.16 SAXON: Comparative Query Execution Graph for Data Set 1.1

Comparative Study of - Axis Type of Query



Comparative Study of - FLOER Type of Query



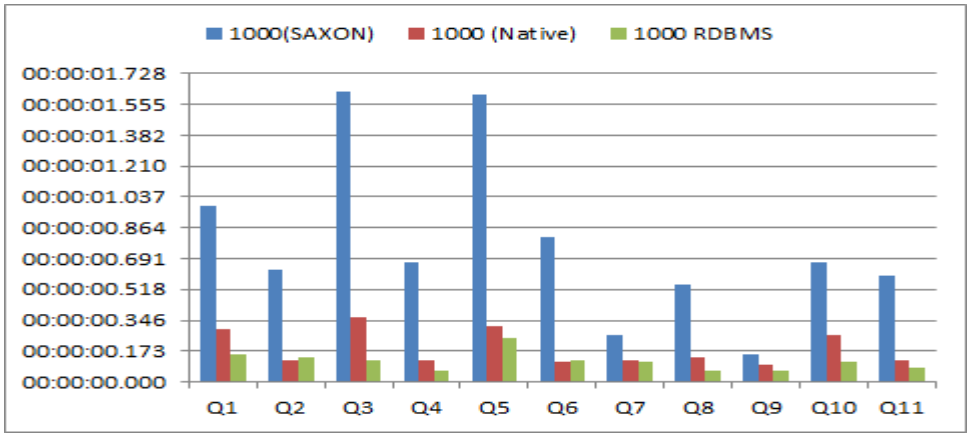
- Comparative Query Execution Graph for Experiment Data Set 1.2:-

Table 5.28 : Comparative Study of Query Execution Time for Data Set 1.2

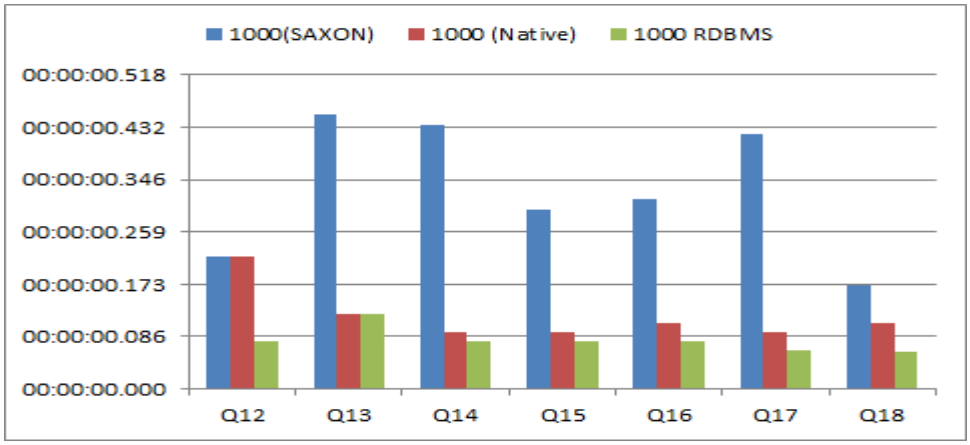
Query	1000(SAXON)	1000 (Native)	1000 RDBMS
Q1	00:00:00.985	00:00:00.297	00:00:00.156
Q2	00:00:00.625	00:00:00.125	00:00:00.141
Q3	00:00:01.625	00:00:00.360	00:00:00.125
Q4	00:00:00.672	00:00:00.125	00:00:00.063
Q5	00:00:01.609	00:00:00.313	00:00:00.250
Q6	00:00:00.812	00:00:00.110	00:00:00.125
Q7	00:00:00.265	00:00:00.125	00:00:00.109
Q8	00:00:00.547	00:00:00.141	00:00:00.063
Q9	00:00:00.157	00:00:00.094	00:00:00.063
Q10	00:00:00.672	00:00:00.266	00:00:00.109
Q11	00:00:00.594	00:00:00.125	00:00:00.078
Q12	00:00:00.219	00:00:00.219	00:00:00.078
Q13	00:00:00.453	00:00:00.125	00:00:00.125
Q14	00:00:00.437	00:00:00.094	00:00:00.078
Q15	00:00:00.297	00:00:00.093	00:00:00.078
Q16	00:00:00.313	00:00:00.110	00:00:00.079
Q17	00:00:00.422	00:00:00.094	00:00:00.063
Q18	00:00:00.171	00:00:00.110	00:00:00.062

Graph5.17: Comparative Study of Query Execution Time for Data Set 1.2

Comparative Study of - Axis Type of Query



Comparative Study of - FLOER Type of Query



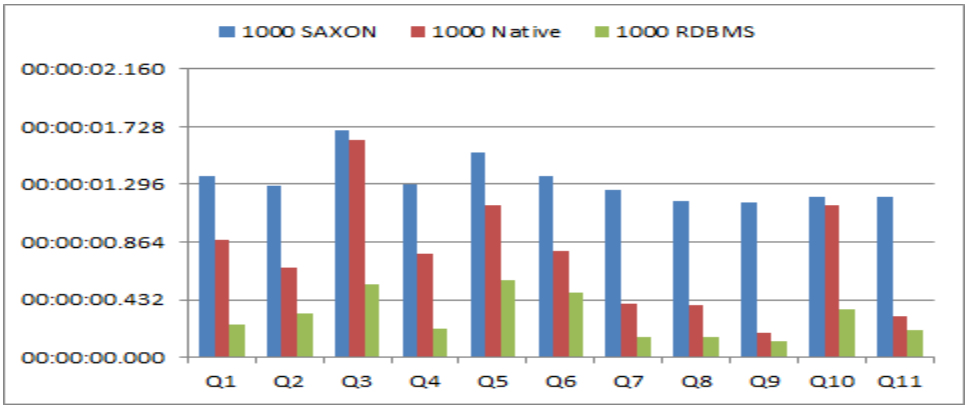
- Comparative Query Execution Graph for Experiment Data Set 2.1:-

Table 5.29: Comparative Study of Query Execution Time for Data Set 2.1

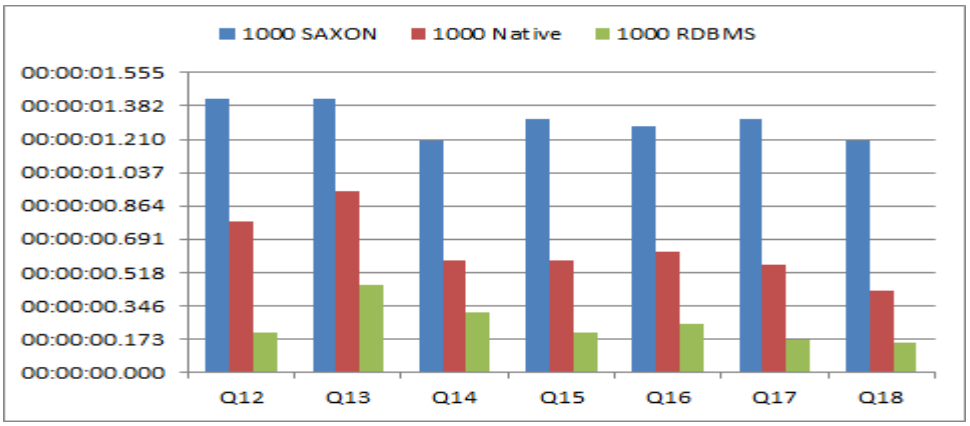
Query	1000 SAXON	1000 Native	1000 RDBMS
Q1	00:00:01.360	00:00:00.875	00:00:00.250
Q2	00:00:01.281	00:00:00.672	00:00:00.328
Q3	00:00:01.703	00:00:01.625	00:00:00.547
Q4	00:00:01.297	00:00:00.781	00:00:00.218
Q5	00:00:01.531	00:00:01.140	00:00:00.578
Q6	00:00:01.360	00:00:00.797	00:00:00.484
Q7	00:00:01.250	00:00:00.406	00:00:00.156
Q8	00:00:01.172	00:00:00.390	00:00:00.156
Q9	00:00:01.156	00:00:00.188	00:00:00.125
Q10	00:00:01.204	00:00:01.141	00:00:00.359
Q11	00:00:01.203	00:00:00.313	00:00:00.203
Q12	00:00:01.422	00:00:00.781	00:00:00.203
Q13	00:00:01.422	00:00:00.937	00:00:00.453
Q14	00:00:01.203	00:00:00.578	00:00:00.313
Q15	00:00:01.312	00:00:00.578	00:00:00.204
Q16	00:00:01.281	00:00:00.625	00:00:00.250
Q17	00:00:01.312	00:00:00.562	00:00:00.172
Q18	00:00:01.203	00:00:00.422	00:00:00.156

Graph5.18: Comparative Study of Query Execution Time for Data Set 2.1

Comparative Study of - Axis Type of Query



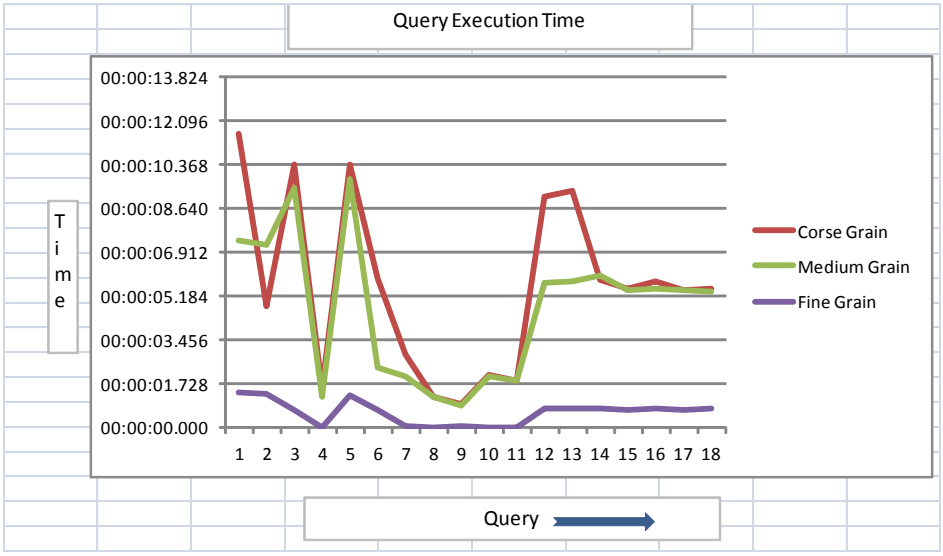
Comparative Study of - FLOER Type of Query



Comparative study of Query execution time on IBM DB2.9 of different storage:-

The XML files can be stored into three formats: - Coarse Grain, Medium Grain and Fine Grain. The "CD. XML" was stored into three formats and the query execution time is recorded into graph 5.19 the graph suggests the best storage time is for fine grain storage.

Graph 5.19: Comparative Query Exaction Graph of IBM DB2 for XML storage

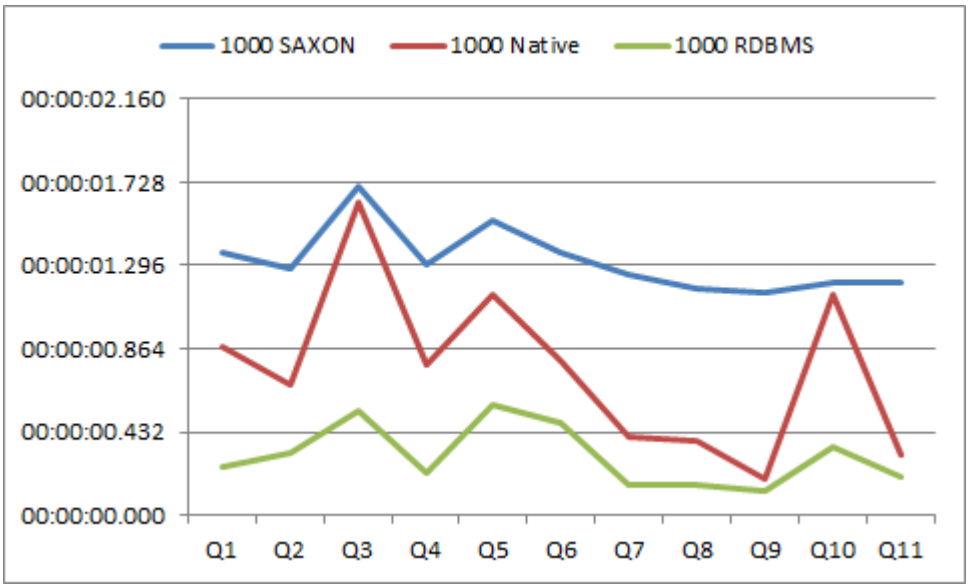


The above graph suggests minimum query execution time is required for fine grain storage and maximum time is required for coarse grain storage.

Comparative study of IBM DB2.9 and SAXON Query Execution time :-

The comparative study of XML SAXON and IBM DB2.9 storage, XML-RDBMS depicted below:

Graph5.20: Comparative study of SAXON and IBM Query Execution Time



-

From the above graph noted that following points:-

- Fine grain (XML-RDBMS) storage required the minimum time for the query execution.
- The Saxon required maximum time for storage. It follows navigation approach for XML storage.
- The moderate query execution time is required for coarse grain storage. In coarse grain the query execution time is directly proportional to number of nodes retrieved from the list.

5.5.3 Comparative Study of Disk Space SAXON and IBM DB2.9 :-

Table 5.30 : Disk Space Analysis for - IBM DB2 and SAXON

Disk Space		
	IBM	**SAXON**
SET 1.1.1	4KB	4 KB
SET 1.1.2	12KB	20 KB
SET 1.1.3	84 KB	160 KB
SET 1.1.4	820 KB	1.68 MB
SET 1.1.5	4088 KB	8.41 MB
SET 1.1.6	8244 KB	15.1 MB

5.5.4 Comparative Study of Disk traffic SAXON and IBM DB2.9

Table 5.31: Disk Traffic - IBM DB2

Disk Traffic	
	IBM
SET 1.1.1	1
SET 1.1.2	3
SET 1.1.3	21
SET 1.1.4	205
SET 1.1.5	1022
SET 1.1.6	2061

.

The observations are listed below:-

1. IBM DB2 is the database product hence all database functionality can perform easily.
2. SAXON store the information into text file so new storage structure, query optimization has to defined from the scratch.
3. The graph 5.16 till 5,20 suggests that query execution time for IBM DB2 is less then SAXON.
4. From table 5.30, IBM DB2 disk space is less than SAXON
5. The disk traffic parameter cannot be comparable because Saxon used heap size for query execution while in IBM DB2 the disk traffic depends upon the buffer paging, cardinality and selectivity factors.

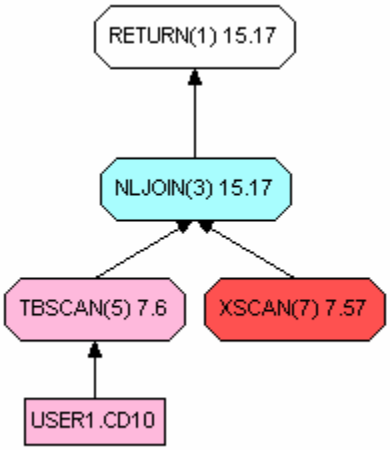
5.5.5 Query execution Plan for XML-RDBMS and XML-Native Storage:-

The query execution plans and Statistical Analysis of Query Execution Plan are listed below:-

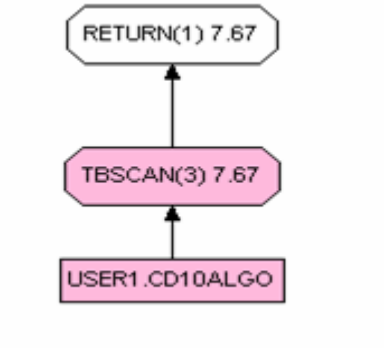
Table 5.20 Query Execution Plan and Statistical Analysis of Query Execution Plan

Query 1 :- for $c in doc("CD.XML")/CATALOG/CD return $c

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



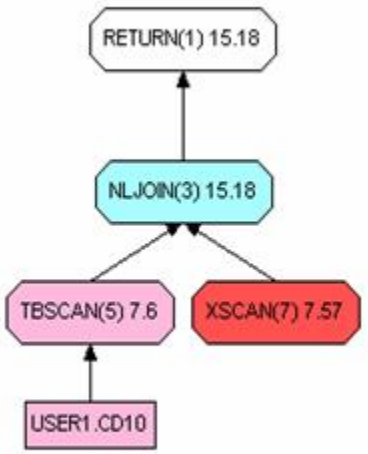
Statistical Analysis of Query Execution Plan:-

Total Cost	15.17
CPU Cost	71,957 Instruction
I/O Cost	2 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	14.33
Total buffer pool pages used	2
Count of buffer pools	0

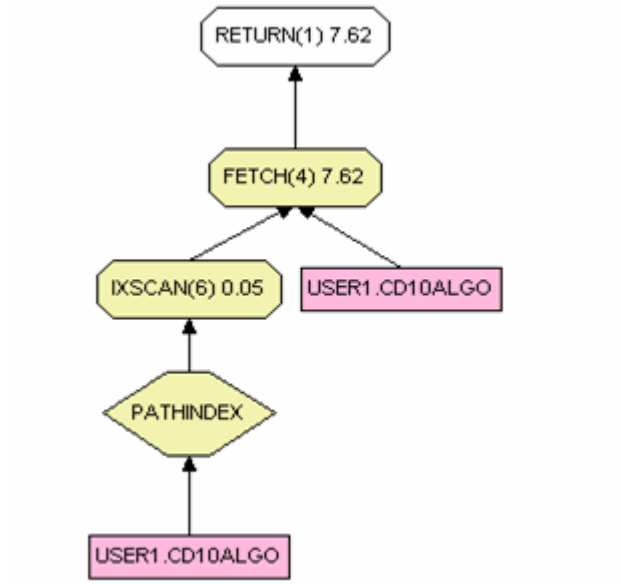
Total Cost	7.67
CPU Cost	147025 Instruction
I/O Cost	1 I/Os
Cardinality	55
Total buffer pool pages used	1
Count of columns	5

Query 2 :- for $c1 in doc("CD.XML")/CATALOG/CD return <item> {$c1/TITLE}

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



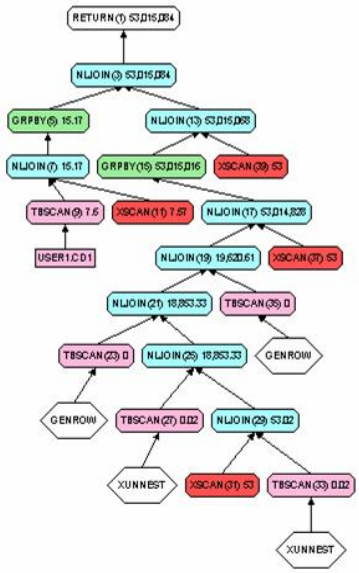
Statistical Analysis of Query Execution Plan:-

Total Cost	15.18 timerons
CPU Cost	73,187 Instruction
I/O Cost	2 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	14.33
Total buffer pool pages used	2
Count of buffer pools	0

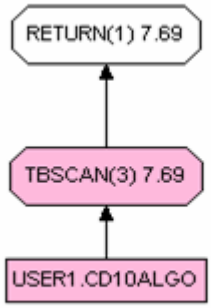
Total Cost	7.62
CPU Cost	85430.14 Instruction
I/O Cost	1 I/Os
Cardinality	7.86
Total buffer pool pages used	2
Count of columns	1
Predicates	2
Selectivity (Q1.PATHID=12)	0.14

Query 3 :- for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/following::*}

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



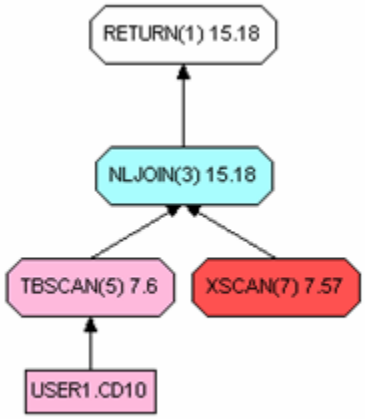
Statistical Analysis of Query Execution Plan:-

Total Cost	53015084 timerons
CPU Cost	100365959168 Instruction
I/O Cost	7002592 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	14.33
Total buffer pool pages used	7,002,492
Count of buffer pools	0

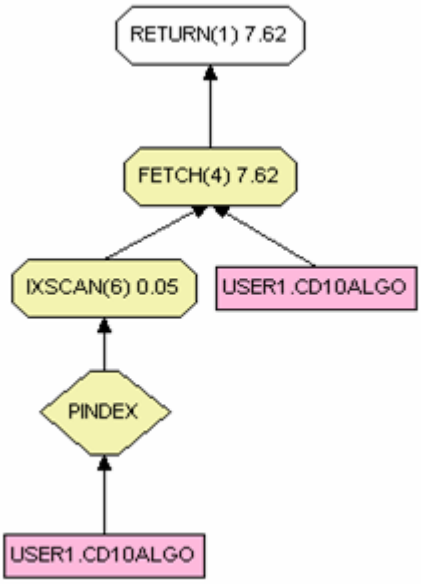
Total Cost	7.69
CPU Cost	171500 Instruction
I/O Cost	1 I/Os
Cardinality	55
Total buffer pool pages used	1
Count of columns	1
Predicates	2
Selectivity	1 (2<Q1.NODEID)

Query 4 :- for $c1 in doc("CD.XML")/CATALOG/CD[5] return {$c1/self::node()}

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



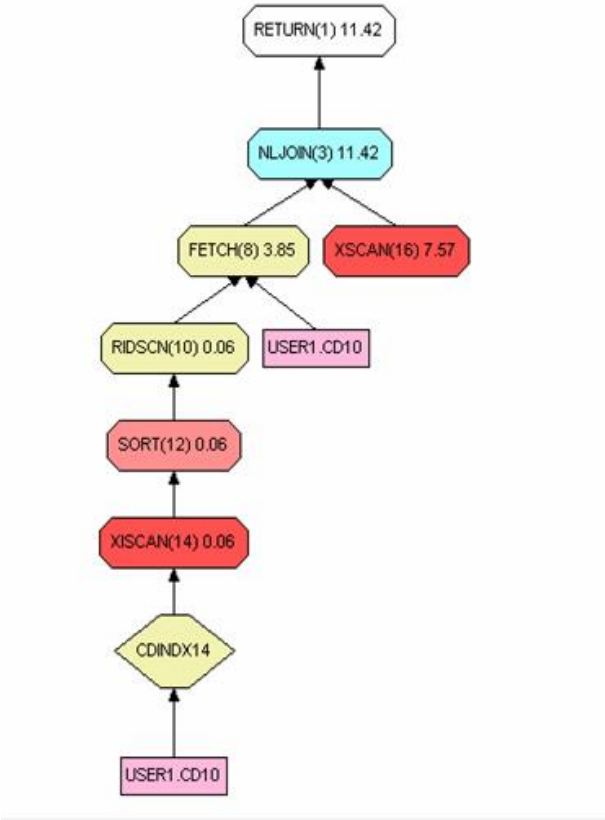
Statistical Analysis of Query Execution Plan:-

Total Cost	15.18 timerons
CPU Cost	73187 Instruction
I/O Cost	2 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	14.33
Total buffer pool pages used	2
Count of buffer pools	0

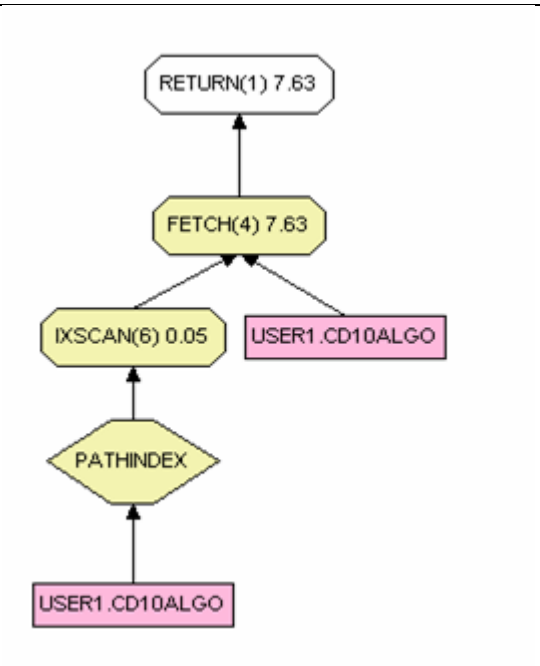
Total Cost	7.62
CPU Cost	77913 Instructions
I/O Cost	1 I/Os
Cardinality	5
Total buffer pool pages used	2
Count of columns	5
Selectivity on PID (Q1.PID=3)	0.09

Query 4:- for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR=1985 return $c1

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



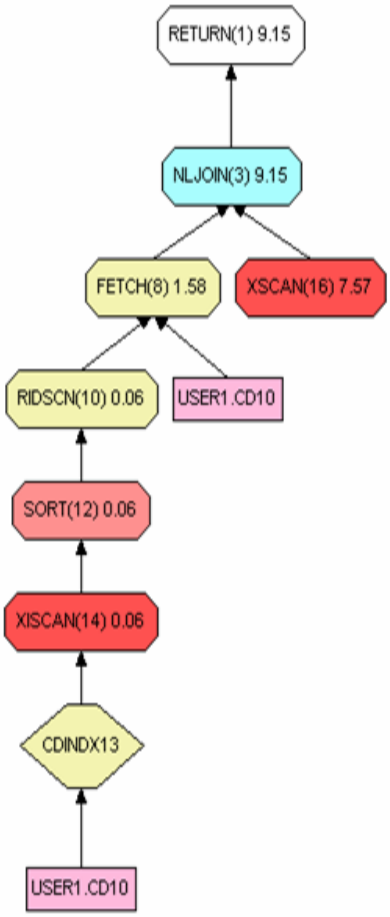
Statistical Analysis of Query Execution Plan:-

Total Cost	11.42 timerons
CPU Cost	107469 Instruction
I/O Cost	1.5 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	1
Total buffer pool pages used	2
Count of buffer pools	0

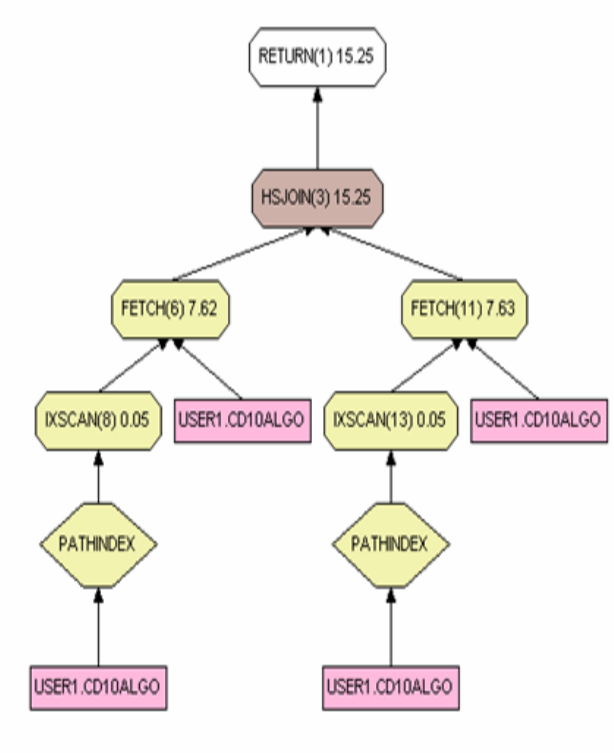
Total Cost	7.63
CPU Cost	88926.57 Instructions
I/O Cost	1 I/Os
Cardinality	0.25
Total buffer pool pages used	2
Count of columns	5
Selectivity on PID (Q1.PathID=17)	0.14

for $c1 in doc("CD.XML")/CATALOG/CD where $c1/YEAR<1985 return $c1

Query Execution plan : XML-RDBMS



Query Execution plan : XML Native Storage



Statistical Analysis of Query Execution Plan:-

Total Cost	9.15 timerons
CPU Cost	104428.2 Instruction
I/O Cost	1.2 I/Os
Tables	SYSIBM.dt
Count of Columns	1
Order of Columns	None
Count of Predicates	0
Cardinality	0.05
Total buffer pool pages used	2
Count of buffer pools	0

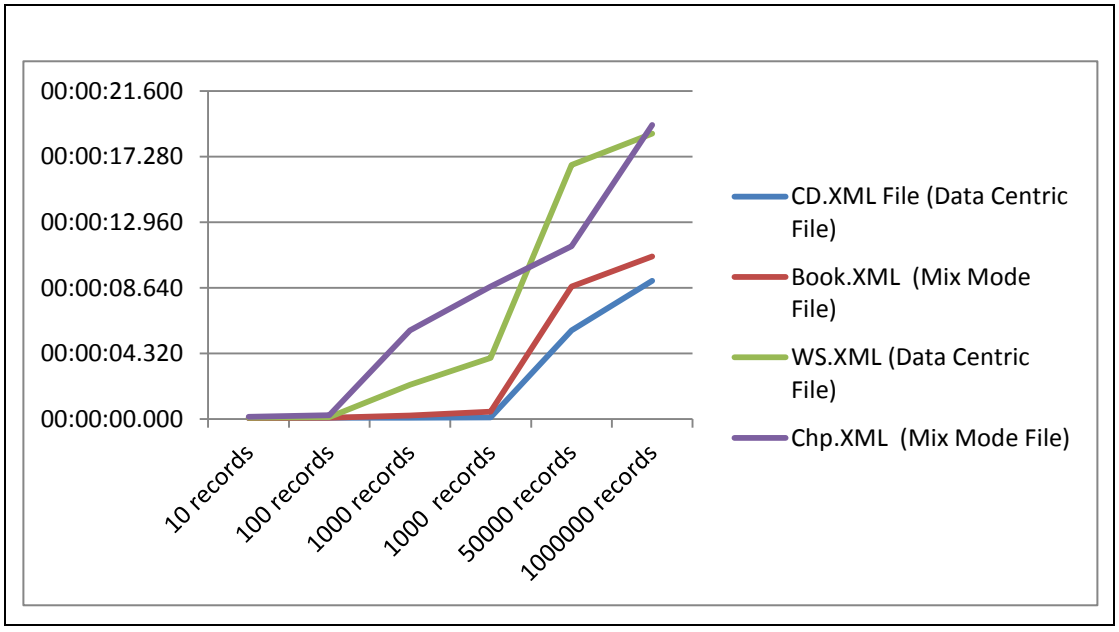
Total Cost	15.25
CPU Cost	176264 Instructions
I/O Cost	2 I/Os
Cardinality	0.17
Total buffer pool pages used	4
Count of columns	1
Selectivity on Pathid (Q1.PathID=17)	0.14
Selectivity on value <'1985 (Q1.EVALUE<'1985')	0.03
Selectivity (Q2.Pathid = 12)	0.14
Selectivity (Q2.PID=Q1.PID)	0.09

5.5.6 Index Creation Time measurement:-

The effective index can execute the query faster. In IBMDB2.9 XML-native storage path index are created and in XML-RDBMs storage unique index is created on a unique identified field.

The java program is used for index creation time measurement. Java program connects to the database and read the time for index creation. The programme is executed ten times and different time measurement was recorded and average index time of construction was calculated. The following graph 5.21 plots the index creation time.

Graph5.21 Index Creation Time



The index creation time for CD.XML and for "Book.XML" is almost the same and WS.XML and Chp.XML is the same. The file structure of CD.XML and "Book.XML" were similar and file structure of "WS.XML" and "Chp.XML" was similar. The graph 5.22 suggests the index creation time depends upon the file structure not on the content of the file.

5.6 General Observation and finding :-

After performing different experiment the XML, following points are observed:-

- XML is used for data management, content management, and used for transformation of data from one machine to another machine. It eliminates third party involvement in e-business transaction because it is inexpensive, extensible. XML is also used as database management. But sill RDBMS systems are needed in business. Because they are matured and all data management functions like transaction management, security etc are easily performed by these system still XML unable to manage these features.
- In a business, for performing a single business transaction, it needs to communicate different systems, and they need to work together for completion of a transaction. These transactions involved thousands of computers, therefore for successful transaction, the platform independency is needed, and this independency is achieved by XML. XML is capable for communicating to different database, different resources etc.
- XML is used for representation of large and complex data. It is also used for metadata representation. e.g. system configuration data, the workflow procedure in the company like production cycle of the product is explained in XML. Business complex data is represented by using XML file e.g. in financial business, company financial product is represented by it, and security note is also designated by it. It is also used for representation of lab rotary research article etc. Whether forecasting data, Olympic game data like timetable of events. The event history records, event scoring information are maintained by XML. XML is used not only for data representation but also used for document representation like product manuals are represented using XML.
- The businesses are global; hence one of the important characteristics is multilingualism. i.e. the software system will translate information from one language into another language. XML supports this property. The different language systems can communicate to each other .e.g. 'Chinese' language website can communicate to any 'Marathi language 'web site. The translation between these two sites is performed

by XML. E.g. in Drupal (a content management System) supports theses functionality, in Drupal different API used XML database for transferring theses information.

- Thus conclusion of study is XML is used for data representation of Meta-data, due to inter-operaratibility property, it is easy to transfer data from one web site to another web site. But still it is used with a combination of Relational database and HTML. These three components have their own different functionality/responsibility. HTML describes display of web pages on screen, XML is a data descriptive markup which describes data records so system can convert and interpret it into its own data format. Relational Database is better to handle large volumes of data within system. It is mature management systems that can efficiently and reliably maintain large quantity of structure data. This data can updated via transactions that ensure the integrity of the database and content can be extracted very quickly when properly configure. There is no equivalent XML management system available. XML documents for storing large volume of data prove insufficient and unreliable.

5.7 Testing of Hypothesis

For the analysis purpose, researcher followed laboratory research and some earlier study documents, a small questionnaire to the different software experts.

If the replies of a majority of the respondents support a hypothesis that hypothesis will be considered as confirmed. Otherwise, it will be considered as rejected. The data connected with the hypothesis and obtained from respondents has been used for this purpose. Conclusions of earlier studies made elsewhere may also be used to supplement the hypotheses of the study.

H_1 - Instead of using XML in isolation better results can be obtained by using combo model which includes RDBMS and HTML.

H_2 - Existing model of XML does have some weak points

H_3- XML-RDBMS storage is always better than XML-Native storage and XML-Navigational storage

H_4- There is no difference between query processing cost of XML databases and RDBMS database

H_5 - XML index size is directly proportional to query processing cost and effective organization and data retrieval of XML data file.

H1: Instead of using XML in isolation better results can be obtained by using combo model which includes RDBMS and HTML.

H0- Null Hypothesis: XML always provide better result with RDBMS and HTML
H1- Alternative Hypothesis: XML utilization is always good without including RDBMS and HTML.
This hypothesis has been tested by using IT expert's opinion and another data collection "case study "analysis method. The inference from IT experts is listed into

Respondent	**Sample Size**	**Proportion Who said Yes**	**Standard error**	**z - statistic**
Software developers+ Software Tester	1021	0.8374	1.1755758	0.6294788

As the sample size is >= 30 therefore normal approximations is satisfied. The z-test is performed on it. The Z- statistic of "XML cannot replace RDBMS and HTML" is 0.6294788 which is < 1.64, hence accept Null hypothesis at 5 % level of significance. Thus it is seen that 83 percent respondents have a positive attitude towards XML utilization along with RDBMS and HTML. It means that "XML always provide better result with RDBMS and HTML" and hence **the hypothesis of the study is accepted.**
The hypothesis is also tested from case study method. The case study data present the role of HTML, RDBMs and XML in web development system. The table 5.3 suggests that the XML is a tool for representation the business document structure. It is not only represents the document structure but also represent the data on it. The representation of document structure is nothing but metadata representation. Table 5.2 presents that XML data is always used with the combination of RDBMS and HTML. From the study, the following points have been observed for utilization of XML

- XML represents data and file structure independently. In other words, if software upgrades and even it moves to different software products, XML data can be moved without the fear that it will become incompatible with its new environment.
- XML is used as data transformer from one application to another.-XML data is capable for interaction with suppliers' and partners' data and it makes applications much more easier using industry standards.
- XML is used for translating Data:- By using XML, business will have the freedom to repurpose and re-use data in multiple situations, rather than repeatedly recreating it.
- IT operation needs to be aligned not only with tactical business needs today, but also with the long-term strategy and direction for the business. Over time, your electronic information could become your most valuable asset; XML is one big step towards it.
- XML supports Unicode and Multilanguage support for data exchange over net.
- XML is used as a database for the small application.
- XML is used for metadata information representation.
- XML is used for data transfer on the web. But in back end still the RDBMS products are used.
- Data is still stored into RDBMS format but to transfer the complex information, the XML is used instead of transferring information into table / record format.
- XML is a very future-proof method for ensuring long-term protection of content. It is the format chosen by many digital archives and national libraries. It is mainly used today for API development, financial transactions, and messaging — and by web developers.
- The database features like security, concurrency and d ACID property cannot be replace by XML. Hence it never replaces the RDBMS data.

Thus hypothesis is accepted: XML always provide better result with RDBMS and HTML

H2: Existing model of XML does have some weak points

H_0- Null Hypothesis > 85% more business represent their data into XML as per their requirement. Additional support is not required

H_1- Alternative Hypothesis < 15% Company takes helps of other technology to represent the data

XML is developed from SGML. SGML is complex and difficult to understand, hence HTML was introduced. But HTML is user friendly. HTML is a text descriptive markup which tells a web browser how to display a page on the screen or through a printer. XML is a data descriptive markup which describes data records so a system dissimilar from the one that created the data will be able to convert it into its own data format. This hypothesis checks whether XML is self sufficient to represent the data.

This hypothesis has been tested by using IT expert's opinion and content analysis method. The data was collected from IT experts on following points: (1) XML represents data as database. (2) XML represent data as documents (3) XML represents data as contentment management and (4) XML is used to represent complete business data (i.e all document type , data type and content type) ,. It has been seen that majority of respondent (85.74%) were agreed that XML can represents the business data.

Respondent	**Sample Size**	**Proportion Who said Yes**	**Standard error**	**z - statistic**
Software developers	470	0.85	1.647047	0.451219

As the sample size is >= 30 therefore normal approximations is satisfied. The z-test is performed on it. Calculated value $Z(0.45) < 1.64$ hence accept the H_0 hypothesis means XML is sufficient to represents the business data . The weighted average method is also tested on table. The 97% software developers agreed the XML is sufficient to represent the business data.

The hypothesis also tested on the business data represented by different company framework (listed into table 5.3). This standardized business XML-framework has been design by standard organization. This standardization is required because business needs to communicate to each other on line i.e. e-business. Today's business in not limited to email or FAX transaction. From the table, the following observations were noted.

- Business represents their framework. This framework is represented by XML.
- XML is used for representation of the business record /object.
- All the business operations are represented by ISO. This information is mapped by using XML.

- The XML not only represents the product information but also the business workflow.

The table5.6 represents the XML framework for the business data representation. It suggests the business data can be built-in into XML since it is extensible. Table5.7 suggests the different web services which uses the XML representation on for the data transformation from one end to the other end of the system. These tables suggests XML is sufficient to represent the business record and to perform end to end transaction, it does not need to add additional technology support for representation this **proof the hypothesis to null.** XML is sufficient for representation of the Data.

H3 XML-RDBMS storage is always better than XML-Native storage and XML-Navigational Storage.

H0 - Null Hypothesis - Data can retrieve effectively and easily in XML-RDBMS storage than XML-Native and XML navigational storage.

H1- Alternative Hypothesis - Data can retrieve effectively and easily in XML-RDBMS storage than XML-Native and XML navigational storage.

To test this hypothesis, researchers followed literature review and experimental method. The data storage is concerned with effective data management and effective data retrieval. In XML the data storage is nothing but maintaining P-C, A-D and sibling relationship as well as follows easily all database characteristics like transaction management, query optimization , security etc. The data retrieval is related to all types of query execution.

XML-Native storage all database characteristics need to develop from the scratch while XML-RDBMs storage, once the data is stored into relational table than the existing database characteristic can apply as it is. In navigational storage XML data is stored as text file. The detail study refer chapter 3 page 48

Effective data retrieval depends upon minimum query execution time, minimum disk space and disk traffic. These parameters were measure through experiment. The experimental results are noted into graph. The graph study suggests the minimum query execution time is required to XML-RDBMS than XML-native storage. Maximum Query execution time is required for XML-native storage. The experiment result observed the following result:-

1) IBM DB2.9 stores XML file in native format and relational database format. The XML-native storage used ".XML" extension. This storage is called as coarse grain storage. If a complete XML file is unable to store into a single row then it is split and stored into different rows with ".XML" extension. This storage is called as medium grain storage. An XML-RDBMS storage XML file is split into different tables. The
2) The graph 5.19 suggest out of these three storage "fine grain (XML-RDBMS) storage" required less time in query execution.
3) The graph 5.16 till 5.20 suggests that query execution time for IBM DB2 is less then SAXON.
4) From table 5.30. Listed the disk space, IBM DB2 required less disk space than SAXON
5) The disk traffic parameter cannot comparable because Saxon used heap size for query execution while IBM DB2 the disk traffic depends upon the buffer paging, cardinality and selectivity factors.

Hence the hypothesis is tested for positive. The XML-RDBMS storage is best solution.

H_4: There is no difference between query processing cost of XML databases and RDBMS database

H0- Null Hypothesis = the query execution plan is similar for XML-RDBMS storage and XML-native storage.

H1- Alternative Hypothesis = the query execution plan is different for XML-RDBMS storage and XML-native storage.

This hypothesis is tested through the experiment method. The query execution plans are plotted into table 5.20. In XML-RDBMS storage index in on "element value" and in XML-native storage index is on "Path Value "attribute. The both plans measure the query execution cost on parameters of "CPU cost, I/O cost, cardinality and buffer pool pages. But the query execution plans are different for both types of storage. In XML-RDBMS storage "join" operations were used frequently while XML-native storage, the query execution plan utilized 'path index' with XISCAN operation (IBMDB2.9 index operation). These plans prove alternative hypothesis, both types of the storage used similar parameters for query processing but their query executions plans are different. So query processing cost is different.

H_5XML index size is directly proportional to query processing cost and effective organization and data retrieval of XML data file.

H0- Null Hypothesis = the query execution time is less for effective organization of data and proper index structure.

H1- Alternative Hypothesis = the query execution time is not depend upon organization of data as well as proper index structure.

The experiment performs, on three different type of XML storage: SAXON – XML navigation storage, IBM DB2 – Native storage, IBM DB2 –XML RDBMs storage. The index strategy is design for IBM DB2 – Native storage, IBM –XML RDBMs in chapter 4 pageno:89 table 4.2. In SAXON it is native storage, for query execution the query file and XML file is passed as stream of string and these strings are matched for desired results. Hence, in SAXON the index definition is not possible. The graph suggest the Maximum time for SAXON Product. Indexes are defined for IBMDB2 native storage and XML-RDBMS storage. The "CD.XML" is data centric file and "book.xml" is mix-mode file. The structure of these files is same. Similarly "WS.XML" and "Chp.XML" file structure is same. The index creation time required for creation of "CD.XML" and "Book.XML" is almost same. Index creation time for "WS.XML" and "Chp.XML" is same. Index creation time for CD.XML is less than "WS.XML". This graph suggest index creation time is also depends upon the path of XML file. As the path of XML file is increased the index creation time is also increased. Thus index creation time is also depends upon the depth of XML file .Another important parameter of query execution depends upon 'XML data Organization'. XML is hierarchical data; it is represented in three structure format. From graph 5.20 suggest the query execution time is higher for coarse grain storage and it is less for fine grain storage. These observation is tested the positive hypothesis.

5.8 Conclusion and Summary:-

Above study suggests the importance of XML in web application development. It is adopted by all three communities: database, document and functional community. It is sufficient to represent the complete business framework. This framework is nothing but the application of data centric XML format.

XML file is stored into XML navigational, XML-Native and XML –RDBMS format. The 'XML-RDBMS' storage is always best out of these three. Once the data is converted into RDBMS format then it can take the advantages of the existing RDBMS features like indexing, query optimization etc.

But when the data is stored into RDBMS format, the data must be stored such that it must identify P-C, A-D and sibling relationship.

With reference to this analysis, researcher proposed new model which is based upon XML-RDBMs storage. The new model, its architecture, index strategy is discussed into next chapter.

References:-

1 A. Schmidt, M.L. Kersten, M. Windhouwer, and F. Waas, "Efficient Relational Storage and Retrieval of XML Documents," Proc. Third Int'l Workshop Web and Databases (WebDB '00), 2000.

2 B.M. Monjurul Alom, Frans Henskens, Michael Hannaford DRXQP: A Dynamic Relational XML Query Processor ; 2010 Seventh International Conference on Information Technology 978-0-7695-3984-3/10 $26.00 © 2010 IEEE

3 Chun Zhang, Je_rey Naughton, David DeWitt, Qiong Luo , Guy Lohman On Supporting Containment Queries in Relational Database Management Systems , SIGMOD '01 Santa Barbara, California USA Copyright 2001 ACM 089791886/ 97/05 pp 425-36

4 Chun Zhang, Je_rey Naughton, David DeWitt, Qiong Luo , Guy Lohman On Supporting Containment Queries in Relational Database Management Systems , SIGMOD '01 Santa Barbara, California USA Copyright 2001 ACM 089791886/ 97/05 pp 425-36

5 D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDMBS," IEEE Data Eng. Bull., vol. 22, pp. 27-34, 1999.

6 H. Jiang, H. Lu, W. Wang, and J.X. Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data," Proc. 13th Australasian Database Conf. (ADC '02) , 2002.

7 Khan L., Chen Q., Rao Y.[2000] : A Comparative Study of Storing XML Data in Relational and Object-Relational Database Management Systems, paper of Dept. of CS University of Texas at Dallas, TX 75093-0688

8 M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," vol. 1, pp. 110-141, 2001. ACM Trans. Internet Technology,

9 R. Bourret, XML Database Products. Available http://www.rpbourret.com/xml/XMLDatabaseProds.htm

10 S. A1-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of the IEEE International Conference on Data Engineering, 2002.

11 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, "Indexing XML Data Stored in a Relational Database," Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04), 2004.

12 Samini Subramaniam , Su-Cheng Haw , Poo Kuan Hoong Mapping and Labeling XML Data For Dynamic Update, Second International Conference on Computer Research and Development , 978-0-7695-4043-6/10 $26.00 © 2010 IEEE DOI 10.1109/ICCRD.2010.173

13 Xiaojie Yuan , Xiangyu Hu, Dongxing Wu, Haiwei Zhang+ and Xin Lian' XML Data Storage and Query Optimization in Relational Database by XPath Processing Model' , © 2013 ACADEMY PUBLISHER

14 Y. Chen, S.B. Davidson, and Y. Zheng "A Bi-Labeling Based XPath Processing System Information Systems 35, 2010, doi: 10.1016/j.is.2009.05.005

15 Y. Chen, S.B. Davidson, and Y. Zheng, "BLAS: An Efficient Xpath Processing System," Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), 2004.

16 J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. F. Naughton, and I. Tatarinov. A General Techniques for Querying XML Documents using a Relational Database System. SIGMOD Record, 30(3):20{26,2001.

17 Mustafa Atay , Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi Efficient schema-based XML-to-Relational data mapping www.elsevier.com/locate/infosys

18 F. Tian, D.J. DeWitt, J. Chen, and C. Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies," SIGMOD Record, vol. 31, no. 1, pp. 5-10, 2002.

19 Mohammad, S., and Martin, P., 2009. Index Structures for XML Databases. In Li, C., and Ling, T. W. (Eds.). Advanced Applications and Structures in XML processing: Label Streams, Semantics Utilization and Data Query Technologies. IGI Global. pp. 98-124

20 M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi. Efficient schema-based XML-to-Relational data mapping. Avaliable online at www.sciencedirect.com since 15/12/2006]20/1/2006

21 W3C Website. XQuery 1.0: An XML Query Language. Online: http://www.w3.org/TR/xquery/,Access on: 30/01/2007

22 Microsoft's BizTalk : www.microsoft.com/biztalk/default.asp

23 ebXML web site: www.ebxml.org

24 the eCo framework : www.commerce.net

25 Open Financial Exchange : www.ofx.net

Chemical XML web site www.cxml.org
26
Special issue of Communications of the ACM on e-services, June 2003
27
http://www.usfca.edu/fac-staff/morriss/651/tech_projects/XML/how_useful_is_xml_in_e-commerce.htm
28
http://www.xml.com/pub/a/2001/08/15/xml-ecom.html
29
http://about.reuters.com/researchandstandards/events/2001/05/xml-europe/how-and-where-xml-is-changing-the-markets-paper.htm
30
http://www.service-architecture.com/articles/xml/finance_xml.html
31
http://webdesign.about.com/od/xml/a/aa060401a.htm
32
33 http://www.webxsystems.com/Pharma.htm

Chapter 6
Proposed Model: Path Encoding Structural Summary (PESS) MODEL

6.1 Introduction

Two approaches are extensively used for XML storage and querying database: XML-RDBM approach and other XML-Native storage. In XML-RDBMS storage, XML data is stored into RDBMS format and acquires the advantage of mature RDBMS technology. The query processing time is high due to multiple join operation on it. Equijoin operation on query processing is very difficult. The containment type of queries can execute on it easily but still 'twig' query evaluation is difficult.

Other option of managing this data is to build a specialized data manager that contains a XML data repository at its core [23]. It is difficult to achieve high query performance using XML data repositories, since queries are answered by traversing many individual element to element paths requiring multiple index lookups [24]. Additional flexibility is needed in order to traverse data whose structure is irregular or particularly unknown to the user. It produced high intermediate result in query processing operation.

Hence a model is required which can take the advantage of existing RDBMS system and for query execution be coupled with specialized data manager. Thus, design the model from the same concept. The preliminary step is storing data into RDBM format and design query processing algorithm which is capable to perform query join algorithm.

However extensive study is performed in XML-RDBM storage. The storage is classified into schema centric project and schema less storage. In schema centric approach, XML data is validated as per DTD or XML Schema representation. It generates different relational schema for each DTD element. For the query processing, the cost of join operation is higher than node and path materialization approach. This drawback is over come [15]. [9] [1] suggest the adequate work is done on schema centric approach. But in schema less approach structural-join operation is still open problem for query evaluation.

In Schema less approach different proposals [13] have been studied, important milestones are : (1) Edge Approach [6] (2) Node Approach [5] (3) Path Materialization [11] [17]. The study listed following limitations from these proposals:-

1. The number of join operation is high for evaluation of [6] "twig" and "equi join "operation.
2. It is very hard to determine the path between node 'A' and node 'B'. Thus difficulties in processing partial matching queries.
3. The "Node Approach" [5] reduced join operation but it lead to the problem of dynamic updating of the node.
4. The Path Materialization [11] approach reduced these drawback but the paths are stored into the string hence cannot support efficient evaluation of multiple "//" axes queries. It cannot support "Like" operation efficiently.
5. For answering the query in XML, it needs to travel many paths. All travel may not lead to final answer for each query. Thus query execution time is increased by such traversal.
6. Multiple queries are executed on the same XML database. To execute these queries, they need to traverse some common path. This common traversal increases query execution time. So there is need to design a mechanism which will reduce it.

These limitations were overcome by PESS model by:-

1. The new unique number is assigning to each XML node such that the node_id identifies the P-C and A-D relationship.
2. These Node_id are capable to perform dynamic updates.
3. XML file is converted into Structural summary and path_id is created for each path.
4. The structural summary is created for the XML file and listed all node_ids on this path.
5. The different files are created and stored into relational tables. Once the data is stored into table, than it can use the existing RDBMS facilities.

6.2 Model Overview: Path Encoding and Structural Summary (PESS)

The model PESS model i.e. Path Encoding Structural Summary summarizes XML Path information and effectively maintains the structural relationship. From the XML file, equivalent label paths are identified and they are encoded by predefined number. The node indexing

technique is used for maintaining structural relationship. The overall structure is presented below in fig 6.1:-

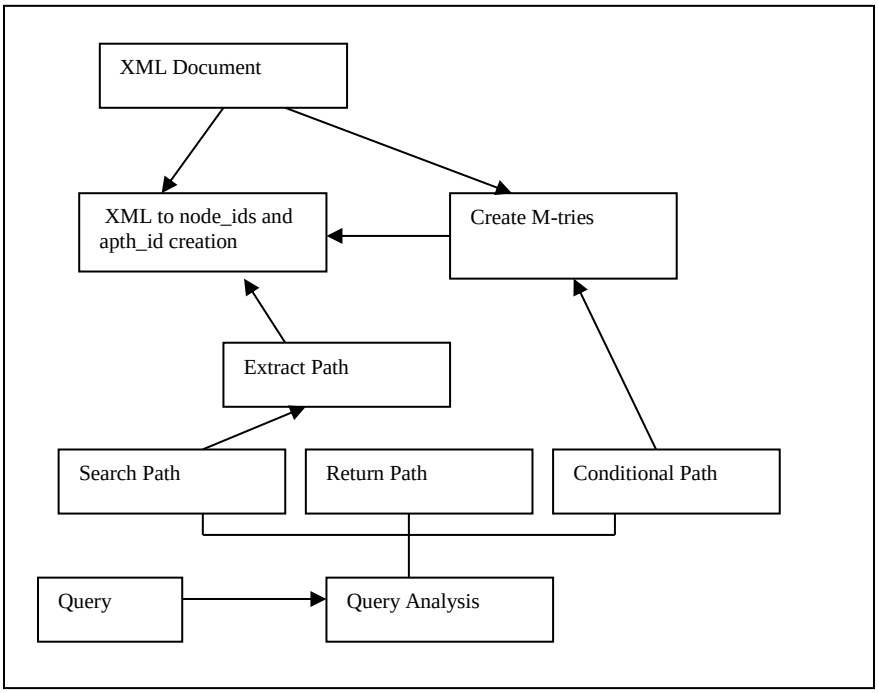


Fig 6.1 PESS Mode Abstraction

The model is divided into two ports: Manage XML file and Query execution. The XML data is organized as:-

- XML Document: - The XML document is read.
- XML Node_id and Path_id creation :- For each XML node a unique identifier is assigned, creation algorithm is explained on page no:178 . Once node_ids are created, than structural summary information of XML file is created and the path_ids are assigned to each path. Algorithm is on page no:185.
- M-tries creation: - It organized XML node information.
- Query organization:-
    - The XML query is read and split into three parts: search path, return path and condition path. Algorithm page no :
    - As per query executions all these paths are searched on the M-tries for final answer. Algorithm is on Page no
- The model will accomplish the: -

- Reduced I/O cost (Query Processing time):- PESS model maintains a data node table. In this table, all data values are stored. This table maintains the information about a node of a particular path. The content based queries are directly evaluated from this table; hence the I/O cost is significantly reduced. Redundant traversal for the query execution is also avoided.
- Reduced Join operation: - The node_ids represent the structural relationships between the XML nodes. These node_ids reduce the join operation for answering the queries.
- Reduced Processing Cost: - Node number defines P-C and sibling relationship, no additional processing is required for fetching records from database file.
- Dynamic Update: - The PESS model avoids the node re-labelling problem. Dynamic updates are possible by using this schema.
- Our index structure avoids multiple lookups in XML file

6.3 Architecture of Path Encoding and Structural Summary (PESS) Model:-

The overall structure of the model is explained in the following fig 6.2. The model is broadly classified into two parts: (a) XML Data Organization and (b) Query Organization.

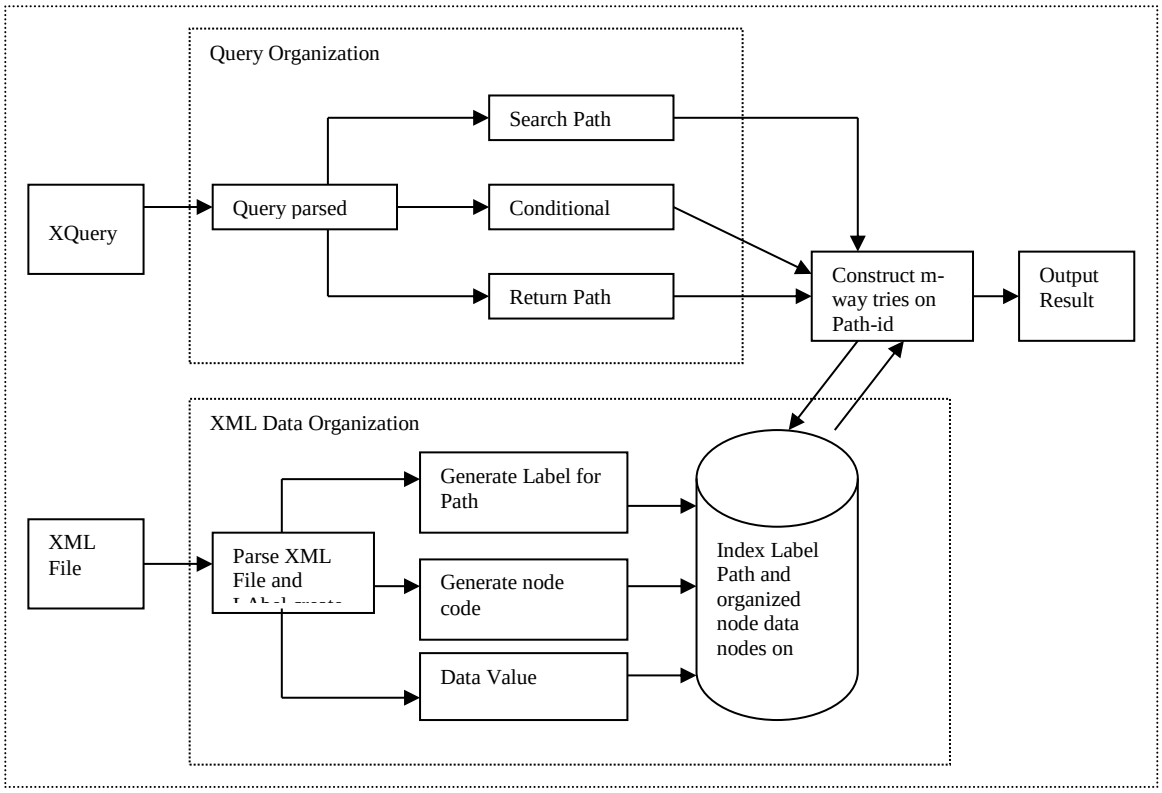


Fig 6.2 : PESS Model

(a)**XML Data Organization** :-

- **XML file is parsed and encoded with different parameters**:-
  - Each data node of XML tree is assigned a unique number. This unique number is a combination of parent code, level number and self code. This node number defines P-C relationship and sibling relationship. By using simple calculation, researchers can also determine P-C and A-D relationship.
  - For the path summary, all equivalent paths are identified .They are assigned a unique number. E.g. an XML tree has three nodes with tag values: 'book', 'title' and 'bookstore'. Each node tags of the XML file are assigned a number - 1, 2, and 3 respectively. A path is a collection of element node tags. Each path in the XML file is assigned a path_id. This path_id is nothing but a collection of these path numbers. For example, XML file has three node labels 'bookstore' with number 1, 'book' with number 2, 'title' with number '3', the label path "bookstore/book/title" is encoded as 123.
- **Reconstruction of XML file**: - Add this parameter as an attribute to XML file and store this information into XML file. Reconstruct the XML file.
- **Storage of XML Data :-**

i. **XML to RDBMS storage** :- For this storage, researchers select the two techniques:

- XML file is stored with new data type ".xml" and b+ index is created on path_id value and secondary index is created on value.
- XML data is shredded into it is stored into three tables:-
- P-table (Path_id, Node_id, type of node)
- D-table (Node_id, value);
- N-table (Node_id, c_id)
- Index is created on Path_id on P-table and D-table index is created on value column and for N-table index is created on Node_id column. Existing query optimization technique is used for query evaluations.
- Third option, XML data is shredded into RDBMs tables and a specialized query processing algorithm is written for it. For this storage the XML data is stored into following RDBMS tables
- P_table (path_id, node_id, type of node, list-of node-attach, value of node)

ii. XML to Native Storage :-

XML data is stored by using multiple link list structure. For effective searching XML node, 'm-way tries' structure is built on path_id.

(b) Query Organizer :-

- **XML to RDBMS storage: -** It supports exiting query optimization algorithm for query evaluation.
- **XML to Native Storage: -** A specialized query processing technique is used. In the above diagram query optimization algorithm is written for it. The detail discussion is in subsequent chapter.

The detail study of the model is discussed below:-

6.4 PESS Model : XML Data Organization Approach:-

Each XML nodes are assigned a unique value. This value is assignment by defining Node Labeling Schema. Similarly all probable XML paths are assigned a number. The PESS node labeling schema and Path Labeling Schema is explained below:-

6.4.1 Node Labeling Schema for XML Database:-

Importance and usage of labeling Schema

XML is hierarchical data. Therefore, in query processes recgnition of structural relationship between the nodes are important. This structural relationship is maintained by assigning a unique number to each XML node. From the literature review two major problems are noted in query processing. The first problem is redundant traversal of XML tree while processing query. For answering XML query, tree navigation operation is needed. Each tree traversal may not lead to the final result. Thus unnecessary disk I/O cost is increased which increased query processing cost. Another problem is to determine reachability between tree nodes. This is calculated by pre-compute transitive closure of data tree. However the sizes of the transitive closures are too large to be used in practice. Thus it would desire to have a method to compactly represent reach-

ability between pair of nodes in a data tree. These two problems are effectively solved by number schema.

From the literature review, researchers conclude that XML node indexing is classify into two types: interval labeling schema (range bases schema) and prefix labeling schema. In general 'interval labeling' scheme, XML nodes are represented by 3-tuple (docNo, LP: RP, level). 'DocNo' is the identifier of document. The pair of 'LP' and 'RP' can be generated by doing a depth-first traversal of the tree and sequentially assigning a number at each visit. The 'level' is the nesting depth of nodes in the tree. This encoding easily determines the containment structural relationship between tree nodes. But the order relationship between tree nodes preceding-sibling/following-sibling does not capture. In the prefix labeling schema, each node is encoded with a unique string S such that (1) S(v) is before S(u) in lexicographical order if and only if node v is before node u in document order (2) S(v) is a prefix of S(u) if and only if node v is the ancestor of node u. The comparative study of these node labeling schema is listed into table 6.1 [23] [13] [6] [3][18][19][20]

Table 6.1 Node Label schema Comparison

	Criteria for comparison	Interval Labelling Schema	Prefix Labelling Schema
1	Label Size	Fixed it is depending upon the depth of the tree	Label size increases as the depth increase
2	Data type used for representation	Numerical	String
3	Time required for computation of relationship between node	Fixed time	it is directly proportional to the depth of the tree
4	Query Execution time	Fast	Slow as compared to Interval labelling
		Index size is small so memory base index can be created from it.	Index size is changeable or length is extensive so this index cannot fix into memory. Creation of disk based indexes is more convenient.
5	Index Updating cost	Complex	Easy
		When New node is inserted into data tree, than all the nodes in the tree, except the left sibling sub tree(s) of the inserted node, have to be updated	When a new node is inserted , only the nodes in the sub tree(s) rooted at the following sibling(s) need to be updated
6	Tree traversal from any position of XML node	Cannot traverse data tree from any node	Data tree easily traverse from any node of the tree
		These node_id represents the range of the node thus difficult to determine parent node-id	Node-id represents parent node this can easily traverse the XML tree
7	Orthogonal (i.e. Orthogonal means adopting another labeling scheme.)	No	Yes

- **Limitation of Existing Labeling Schema :-**

From comparative study, advantages and disadvantage of both numbering schema are listed. (1) The query execution time for interval labelling schema is always good then prefix labeling schema. Because numbering schema represents the nodeids in number format while prefix labeling schema represents node-ids in string format. (2) The label size is fixed in interval labelling schema while it is increasing as per the depth of XML tree in prefix labelling schema. (3)Storage space of prefix labelling schema is always less then numbering schema.(4) Prefix numbering schema infer node's P-C and A-D relationships from the label of the node and easier to update. Hence researcher inspire to designing the prefix labeling schema with objectives:-

- Node_id can easily determine containment and order relationships.
- Dynamic Indexing is possible.
- Node_ids can satisfy the orthogonal property.
- Design of Node_id is such that in RDBMS storage, it reduced structural join operation and in Native storage, it reduced intermediate result calculation.
- Query performance time should increase.

6.4.2 PESS Model Approach for Designing Node_id Indexing:-

6.4.2.1 Introduction :-

All above mentioned drawbacks are overcome by the PESS model. The model has Prefix based labeling schema with following characteristics:-

- Easily calculate Parent- Child, Ancestor – Descendent relationship.
- Sibling relationship can be calculated.
- Unique identification of node is maintained.
- The ordering information is maintained. (e.g. the 'book' tag appear in the tree 4 time, the node is must represent it is first occurrence or second occurrence of the tag)
- Dynamic Indexing is possible.
- Reduction in Index size.
- Node_id helps in reconstruction of XML file. (i.e. converting from RDBMS to XML)
- Nod_id is maintaining document order relationship.

6.4.2.1.1 Generating the Unique_id for the XML file:-

The XML file is read and for each node, unique_id is assigned. This node_id is composed of three components -'Parent identification number + self-identification number id+ level number'. Hence, it easily recognizes P-C and A-D relationships. Detail algorithm is discussed in the subsequent chapter. For example in the following XML file 6.3, the root node 'bookstore' is assigned the number '0'. Its child node 'book' has a number '011' where '0' represents parent code, next '1' represents the self-identification number of the code and last number'1' represents the level number of the tree. The node 'book' has a child 'title', which is represented by '0112'. The last digit represents level number. It is '2', so first two digits represent the parent code, and the digit in between the level number and parent code represents self identification number. So this code is a combination of the relationship between nodes. P-C and A-D relationship can be evaluated by simple node processing. The code is also utilized for dynamic updates. When a new node is added, then unutilized number is assigned to it. The re-labeling is not required. The example is explain in fig6.3 and the sample nodes are presented in fig 6.4

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday  </title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
 </book>
 <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
 </book>
</bookstore>
```

Fig 6.3 XML File

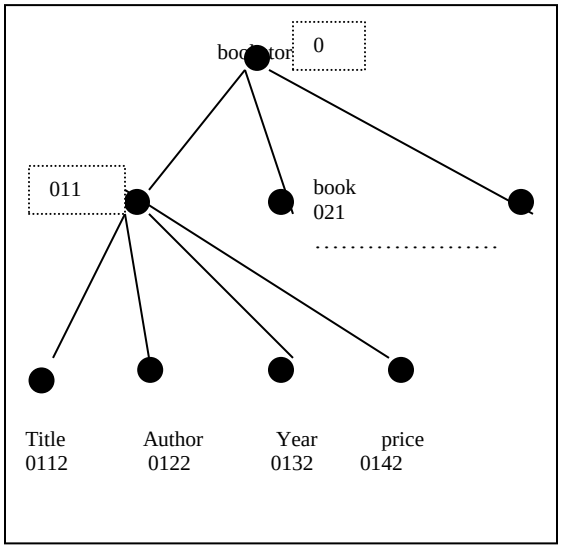


Fig 6.4 XML Node_ids

6.4.2.1.2 Node_id Structure:-

Every node in the XML file is assigned a unique number, which maintains P-C and hierarchical relationships. There are two steps for creation of node_id. The steps are:

1. The code '0' is assigned to the root node.
2. Each non root node is assigned a code as per the following instruction :

a. The node_id of ith child element of parent N is 'Node_id of (N)+ unique number of node + level number. For example:-

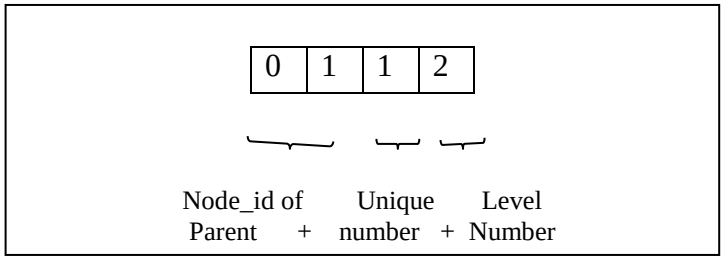


Fig 6.4 Node+id Structure

Algorithm6.1. Assign label to XML file

1. Read XML File
2. If the node is root nodes than assign '0' to root node
   Else
3. Assign node label with previous label plus the next new number and level number
4. Perform the step 3 repeatedly until not eof() of XML file.

Psudocode 6.1 Assign XML node label to XML file

```
Output a computed label for each visited node
Begin
01: while (! n.isEmpty ()) // n is the currently visited node
02: if (n is the root node) then
03:    label (n) =null;
04: else
05:       n.prefix label = label (n.parent);
06:             self-label [0] = 0;
07 :            for( i = 1;i <= n.parent.childnum; i++) do
          // n.parent.childnum denotes the number of children
          // of n.parent node
08 :     self-label[i]= 1 + self-label[i-1];
          // self-label[i] denotes the self-label of i^th child node
          // + denotes the concatenation operation
09 :        n.self-label = self-label[i];
10:        label (n) = n.prefix label+ n.self label + level;
11:     end for
12:   end if
13: end while
```

6.4.3 Label Insertion Algorithm for PESS Model :-

- **Labeling Scheme for Update:**

All these issues are addressed by our numbering schema:-

- The new XML node is added at three position of the XML file: (1) the start of the tree, (2) in between, and (3) at the last position of the XML file.
- From the literature review; the node_id of the XML node is changed as per the order of insertion, because in some research work the node_ids are assigned on mathematical calculation (for example, node_id is a product of two consecutive prime numbers). In some

research work, it is a character string; the labels are assigned as per lexicographical order or alphabetical sequence.

But our labelling schema is neither dependent on lexicographical order, nor on mathematical calculation. Thus, the new node_id is assigned without re-labelling existing number schema. The following steps are used for assigning the new label to the code:-

- The algorithm reads the parent code of the node being inserted newly.
- It reads the children of the parent node; and split each child code into 'self-code', 'parent code' and 'level code'.
- The algorithm organizes all the self-codes of the child nodes in ascending order. Let us say the last self-code of the children node is 'n'.
- If all self-nodes are in sequence :-
    - Newnode self-label = n+1;
- Else
    - New Node self-label = unused number between two consecutive self-numbers.
    - Node_ids = 'parent_code' + Newnode self lebel + level code.

This algorithm assigns the new node_id, without relabelling the existing node_ids. The literature, have noted another problem of numbering schema. There is no re-utilization of deleted node_ids from the XML tree. This issue is also addressed by our algorithm by checking two consecutive numbers of the self node_ids list. The sample node_id are explain in fig 6.6,fig 6.7 and fig 6.8

Algorithm6.2:- New node insertion into XML file :-

1. Read the node_id
2. Read the level no for insertion of new node
3. Read the code on that level
4. Separate the level number,self node_id and pre parent no.
5. Read the last self _id number and add the next successive number .
6. Add the combination of parent code , new self-node and level node.

Psudocode 6.2 New node Insert into XML File

Input left self label N_{left} and right self label N_{right}

Output new inserted self label N_{new}

Begin

01: if (N_{left} is empty but N_{right} is not empty) then // Case 1

02: $N_{new} = N_{right} + n$;

03: else if (N_{left} and N_{right} are not empty) then // Case 2

04 : if (len (N_{left}) <= len (N_{right}) then

05: $N_{new} = N_{right} + 0$;

06: else if (len (N_{left}) >= len (N_{right}) then

07: $N_{new} = N_{left} + 1$;

08 : else (N_{left} is not empty but N_{right} is empty) // Case 3

09 : $N_{new} = N_{left} + 1$;

10 : end if

11: return N_{new}

End

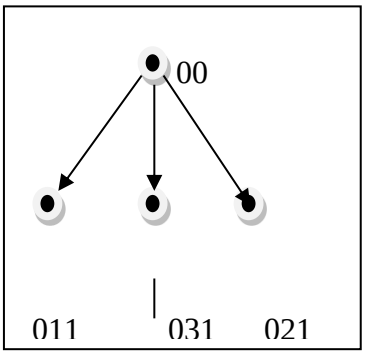


Fig 6.6 Node insertion in between

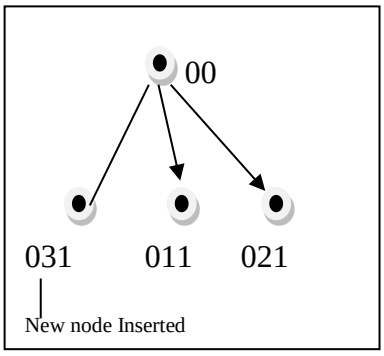


Fig 6.7 Node insertion before

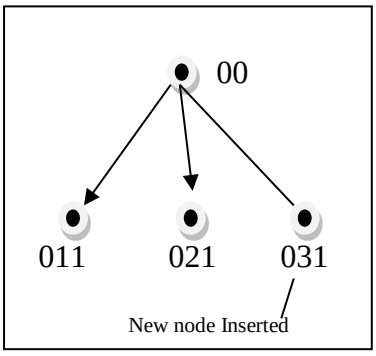


Fig 6.8 Node insertion at end

6.4.3.1 Node Labeling Relationship :-

Algorithm 6.3 Node - Labeling Schema for XML Node relationships

The node_id determines the P-C, A-D and sibling relationship but some additional processing is required for identification of this relationship.

Algorithm 6.3.1 Parent- Child relationship: -

- **Parent Relationship: - If label of node x is equal to the prefix_label of node(y) + (level number(y) – 1), then x is parent of y.**

  Let's consider the example, in the node_id '01113', the last digit represents the level number.

  1. The level number is '3'. Therefore, the first three digits represent the *parent code and it is '011'.*
  2. For identification of the parent code, the algorithm subtracts 1 from level number, i.e. (3-1 =2). Therefore, the *level number of the parent code is '2'.*
  3. For the parent node_id, the algorithm adds the code (from step1) and the level number (step2); it is 0112. So the parent of node_id '01113 'is '0112'. '

- **Child Relationship: - If label of node y is equal to the prefix label of node(x) + (level number(x) + 1), then y is child of x.**

  1. Let's consider the node_id '0112'. The children are
  2. The last digit i.e. level number of the children is '3' i.e. 2+1
  3. The parent code is '011'.
  4. The search () function is executed on 'node_ids' list. This function finds out all the child whose level code is '3', and first three digit is '011'.

Algorithm 6.3.2 Sibling relationship

- **Sibling: - if node x is a sibling of node y iff level of x == level of y and parent code of x = = parent code of y.**

  Let us take the example of consider node_ids '011','021','031'.

  - The last digit represents '1'.i.e. all nodes are on same level.
  - The level number is '1'. Therefore the first digit represents the parent code and it is '0'. All node_ids have the same parent code. i.e. '0'.
  - The parent code and level code are the same for all node_ids, hence they are siblings.

Algorithm 6.3.3 Ancestor-descendent relationship: -

- **Ancestor relationship: - If the label of node x is ancestor of y iff level of x < level of y. label x equal to strncpy(label(y) , level number)**

  Let us consider the node _id is '01113'

  1. The last digit '3' is level code, so the first three digits constitute the parent code of the node. It is '011'. On subtracting '1' from level code '3' i.e. gets '2'. Thus the parent code of parent node is '0112'.
  2. Similarly the node_id of parent code '0112' is '011'.
  3. Similarly the node_id of parent code of '011'. It is '00'.
  4. Thus, the ancestor of node_id of '01113' is: '0112', '011', '00'.

Let us consider another example of the node_id '021114'. After performing all above steps one by one , researchers find the ancestor list : '02113', '0212','021', '00',

- **Descendent relationship**: - **:- If the label of node x is descendent of y iff the first digits of label x contain the label y.**

  Let us consider the node_id '021'-

  - The last digit is the level number. i.e. '1'.
  - The first two digits represent the node_id. - it is '02'.
  - Then the algorithm perform the search () function on node_id list, and extracts only those node_ids whose starting two digits represent the number '02', and the level number is greater than '1'. All these nodes are descendents of the node '021'

6.4.4 Generating XML Path-Id index:-

The path summary index is summaries, all the paths from root node till any arbitrary node of the document [19]. An XML queries are evaluated by simply accessing path summary information instead of on XML document. These type of indexes are good for answering simple path query but it involves multiple join for answering branch path query (twig query). From the literature, these proposals classify into two types: XML-RDMS path encoding approach and XML-to-Native path encoding approach. The first proposal reduces storage cost and preserves the XML structure. Hence excellent performance is achieved in answering simple path query. These

proposals lead to the problems of high 'join' operation for answering query and high updating cost. They are failing to process the equijoin operation.

The XML-Native approach bi-similarity technique is applied for XML data storage. These proposals are providing best execution time as compared with the previous proposals [12]. However, there is a trade-off between the size of a path summary index and its performance. This problem can be solved by designing a flexible path summary index that can adjust its size according to the query workload and database size [5]. 'Feature-based indexes' are designed from it. The detail discussion of the framework discussed below

Definition 1:- A *label path* for a node *v* in an XML data model *D*, denoted by *path* (*v*), is a sequence of dot-separated labels of the nodes on the path from the root node to *v*.

For example, node 0112 in Fig. 1 can be reached from the root node 0 through the path 00.011.0112. Therefore, the label path for node 0112 is *bookstore.book.title*.

Definition 2:- Nodes in an XML data model D are equivalent if they have the same label path. For example, node 0112 and 0122 are equivalent since their label paths are the same. The parents of equivalent nodes share the same label path and thus are equivalent. For example, the parent nodes of 0112 and 0122 are nodes 011 and 012 respectively, which are equivalent.

Path id generation:-

1. Create tag directory: - A table is created for each XML tag and a number is assigned to it.
2. Path directory creation: - XML path is considered here read from root node till leaf node. Thus each XML tag is read and equivalent number of tag (which is created in previous step) is assigned. The combination of these tag values determines a complete path. The partial path of tree can be identified by substring searching in the path_id column.

e.g. Each XML node is either an element node, or attribute node. Each node is assigned a tag. For each tag an integer number is assigned. For example in fig(1) '*bookstore*' is assigned number 1, '*book*' is assigned number 2, '*title*' is '3' . Thus label path '*bookstore.book.title*' is coded by the number 123. In fig (6.9) all label paths are identified and coded by the following numbers.

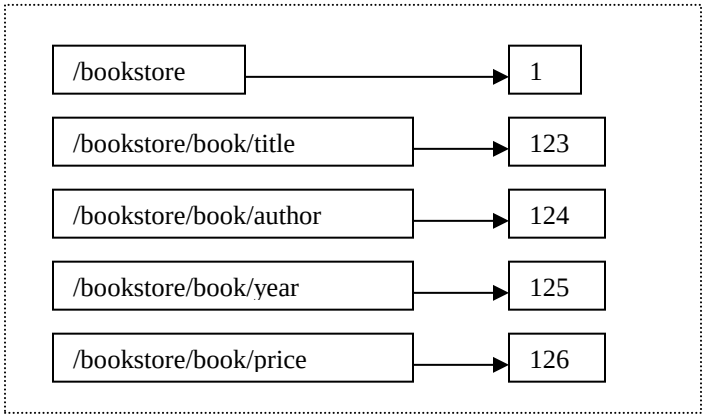


Fig 6.9: Node Label

Above XML File is encoded as:-

```
<nodeid= 00 , parentid=00 ,  pathid = 1>
   < nodeid= 011 , parentid= 00 ,  pathid = 12>
      < nodeid= 0112  , parentid=011 ,  pathid = 123>
         Everyday
       <nodeid= 0121 , parentid=011  ,  pathid = 124>
         Giada De Laurentiis
      < nodeid= 0131 , parentid=011 ,  pathid = 125>
         2005
      <nodeid= 0141 , parentid=011 ,  pathid = 126>
        30.00
-------------------------------------------------------------------------
     < nodeid= 021 , parentid= 00 ,  pathid = 12>
      < nodeid= 0212  , parentid=012 ,  pathid = 123>
        Harry Potter
       <nodeid= 0222 ,  parentid=012 ,  pathid = 124>
         J K. Rowling
      < nodeid= 0232  , parentid=012 ,  pathid = 125>
         2005
      <nodeid= 0242 , parentid=012 ,  pathid = 126>
         29.99
```

All equivalent node_ids and path_ids are identified and the XML file is shredded into relational table with attribute node-id, parent-id.

Algorithm 6.4 : Creating Path_id for XML File

1. Read the root node and store into path dictionary and assign path number as one.
2. Read the next tag name.
3. If tag name is not in path dictionary then store it into path dictionary and assign the next number
4. Read all tag names from the XML file
5. Read the XML file and assign the number with respective tag number.
6. For each respective path add the respective node_id number.

Psudocode6.3 : Generating Path identifier for XML file

Input: - Input XML file.

Stack {name of tag: string; No: integer}

Output: the path_id

1. While not end of file
2. Read the first tag of the file
3. If(tag == stack [tag])
   i. Stack[top++];
   ii. Stack[tag-name]= tag;
   iii. Stack[No] = no++;
4. End of file
5. //create the path-id
6. While not end of file
7. Read tag
8. Search tag into stack
9. Do
10. { If (found)
    i. No = stack[no]
    ii. Path_id= Path_id + No;
    } while(not (end of path)
11. End of file.
12. End

6.4.5 PESS Model data storage schema:-

The XML-RDBMS storage is widespread [9]. Consequently, a large quantity of non-XML data has already been stored in them. In order to work in closer cooperation with such traditional data, it is useful to store XML data in the same type of databases. [13]. Query optimization techniques and the processing mechanism in relational databases have been studied for a quarter of a century, and have reached full growth. Thus, it is pragmatic to cope with them.

Different approaches are used for storing XML database. The data is stored as string format (CLOB/BLOB) or used user defined data type for XML file storage 'XMLVARCHAR', 'XMLCLOB','XMLFILE' [11] The XML queries cannot be executed on it. A Specialized data type ".xml" is used for storing a complete XML file. Each document is stored as a value of type "xml" in a separate tuple. Special purpose indices are created to index XML data. The other approach of storage is tree representation. XML data is modeled as a tree and stored using relational i.e. tree like structure. The storage easily maintains P-C and A-D relationships. But the element gets broken into many pieces, and a large number of joins are required to reasonable sub elements into an element. For our experiment, select ".XML" data type storage and tree structure storage for storing XML file.

a) **Tree structure storage** :- XML data is shredded into three data files-

- P-table(Pah_id, Node_id, type of node)
- D-table(Node_id, value);
- N-table(Node_id, c_id)

- P-table (Pah_id, Node_id, type of node):- This table stores path information. For each path, all equivalent nodes are stored. The third column takes the value either 'E'- for element data information or 'A' – attribute node. Index is on Path_id value.
- D-table(Node_id, value):- This table stores the information of the value nodes. The table is composed of two columns 'Node_id' which stores unique node_ids of the XML file node. The 'value' column stores the node value information. Index is on Node_id value.
- N-table (Node_id, c-id) If the nodes are element nodes, than their information is stored into this table. The column 'node_id" store the node identification and 'c-id' stores its children node_id. Index on Node_id.

Index on XML document:-

Two types of indexes are created on XML document. The B+index is crested on 'Path_id' and secondary index is created on node_id.

6.5 PESS Model :Query Execution Approach:-

The model consists of two types of algorithm for query execution, they are

A. Execution of query on predefined RDBMS algorithm:-

Progression of RDBMS technology is from 1970; hence, today it is steady and stable technology. It has been studied from all aspect of the database feature and effective algorithms are already available. Thus researchers execute the existing query optimization algorithm.

B. **Native algorithm for query execution:-**

From [7], suggested a framework for XML query execution. As per the proposal , it is always good if the data is stored into RDBS format and native query execution algorithms are designing for it. On the same concept "PESS" model is designed. The data is stored into relational tables and native query execution algorithm is designing for it. The algorithm is dived into three steps:- Query Parsing, Design three paths for query execution and Query evaluation. The algorithm Steps are:-

Algorithm: 6.5 XML Query Execution (For Native Storage)

1. Query Parsing
2. Separate the Query Execution Path
3. List all S-Path , R-Path and C-Path
4. Search Path into Path-dictionary
5. Execute the query Path.

The detailed discussions of these algorithms are as follow:-

- **Step1- Query Parsing:-**

1 **Static Analysis:-** The query is checked for grammatical error i.e. syntax error. If the [expression/query] is not well-typed, static type of error occurs. For example, the comparison between an integer value and a string value might be detected as static type error during the static type analysis. The next phase is query normalization; i.e. complex query is rewritten into simple query expressions. The complex 'for-let-where-return' expression is split into individual expression for, let, where and return clause. The query tokens are listed into table 6.2.

Table 6.2 Query Token

Token Class Name	**Pattern**
ForToken	FOR or for
InToken	IN or in
VariableToken	$(a-z)(a-z)*/ [1]
DocToken	Document("*path*") or Doc("*path*") [2]
PathToken	((//|/)(a-z)*)((/)(a-z)*)*
TagToken	<(a-z)(a-z)*>	</(a-z)(a-z)*>
RelationToken	<	>	<=	>=	=
LabelToken	"(a-z,0-9)(a-z,0-9)*"
WhereToken	where	WHERE
ReturnToken	return	RETURN
UnknownToken	Any unidentified string
EndToken	N/A

- **Step 2:- Design three paths for query execution [2]** The query path is split into three types:- (1) 'Search Path', 'Condition Path' and 'Return Path'. The search path is associated with 'for' clause. It returns the path where the query nodes are supposed to search. The conditional clause specifies the specific condition on the return node. The return path specifies the return node of the query. Three different paths are identified from the query expression - C-path (Conditional path), R-path (Return path), and S-Path (Search Path). C-path is used for conditional searching, R-path identifies the return expression, and S-path is used for searching the query nodes. For example the following figure6.5 shows these different paths:

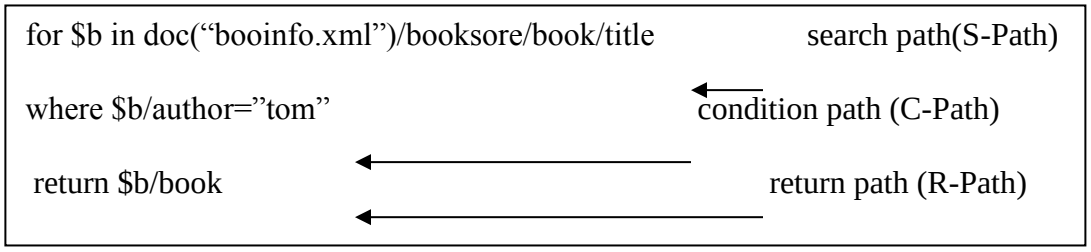


Fig 6.5 XML Path Segregation

The respective path_ids are identified from this path information.

Step 3:- Query Evaluation:-

1. The search path and return path are validated. Both paths are valid if they have same root element.
2. Desire node_ids are retrieved from database.
3. If node_id is a 'value node' then the respective value is displayed.
4. If node_id is 'element node', then the respective child node_ids are retrieved by performing some processing computation on it desired result is displayed.

Algorithm 6.5 :- Query Execution for XML-RDBMS Storage :-

1. "Query Tokenizer" class will separate the query into different token.
2. "Query Path Separater "class will separate the different path SPAHT, CPATH, RPATH.
3. First search path in look into Path Dictionary.
4. If (path exist) store flag value into variable "temp".
5. **if Simple Query :-**

5.1. **Query Contain C-Path and R-Path**

5.1.1. Read C-Path and R-Path code

5.1.2. Open Path index file

5.1.3. Read label code of respective node address

5.1.4. Compare 'value' of respective node

5.1.5. If (found)

5.1.5.1. Compare Label code of R-path node.

5.1.6. If (match found)

5.1.6.1. Display content of matching node

5.1.7. Else

5.1.8. Display error message

1.1.1. Else Display "Error Message"

5.2. **Query Contain only R-Path**:-

1.1.2. Read R-Path

1.1.3. Read Node address

1.1.4. Return value of node address

2. Twig Query Execution

2.1. Query Contain Conditional path

2.1.1.Read R-Path and C-Path

2.1.2.Open Path index file.

2.1.3.Read C_Path node value

2.1.4.If (Valid result)

2.1.4.1. Read node label of respective path

2.1.4.2. From label code read:-

2.1.4.2.1. Read parent code

2.1.4.2.2. Compare parent code of (R- path and C-Path)

2.1.4.2.3. If (valid(parent code + level code))

2.1.4.2.4. Read level code of match R-Path node

2.1.4.2.5. Open node index file

2.1.4.2.5.1. Read node from parent code+ level code

2.1.4.2.5.2. Open XML DataModel File

2.1.4.2.5.2.1. Read connecting node address

2.1.4.2.5.2.2. Read it till value node not found

2.1.4.2.5.2.3. Display content of value node

2.1.4.3. Else Display “ERROR!!!!1”

2.1.5.Else Display “No record Found”

2.2. Query without Condition Path

2.2.1.Read R-Path

2.2.2.Open Path-index file

2.2.3.Read node address from Path-index file

2.2.4.Open XML Data Model file.

2.2.4.1. Read address of connecting node

2.2.5.If node is element-value node

2.2.5.1. Display element value

2.2.6.Else

2.2.6.1. Read next connecting node address

2.2.6.2. Continue process

3. Display Query Result

6.6 Results and Analysis of PESS Model:-

The experiment is performed on PESS model. The data is stored into relational schema (Schema is mention on page No 178) and existing RDBMS query optimization algorithm is used for query execution. The PESS model is evolved from XML-RDBMs storage approach. The different XML-RDBMs approaches were listed into chapter 3 at page no: 44. PESS model structure is similar with existing "Reverse Path Model". Hence the results of PESS model is compared with existing model –i.e. with "Reverse Path Approach".

6.6.1 Measurement Parameter for experiment :-

The PESS model is a "Path Encoding Structural Summary" in which the structural summary information is stored and each node is assigned a unique identifier. The combination of Path summary information and unique identification of node will increase the query performance. Therefore to measure the effectiveness of the model the following parameters are selected: (1) Query Performance Measure (2) Efficiency of node identifier.

6.6.2 Criteria 1 - Query Performance Measurement :-

The first set is used for studying XML data storage and indexing mechanism. It has impact on the query performs. [11] Proposed guidelines for the presentation and comparison of index techniques. Based on these guidelines, researcher selects the following set of comparison criteria. The following parameters are measured during the experiment:-

a. Query evaluation speed: - The main purpose of indexes is to accelerate the query execution. This parameter measures the query execution time that index required to evaluate a query in comparison to other index
b. Disk space: - The amount of disk space is nothing but the total number of pages required for storing the index file, or data file.
c. Disk traffic: - The disk traffic is nothing but the total number of pages fetched from the disk for query execution. It is depend on index.

d. Scalability: - The size of documents stored in database constantly grows and cost for storage space increases. Index structures therefore need not only be able to handle small amount of data, but need to scale up to larger document.

6.6.3 Criteria-II Measurement Parameter for Node identifier:-

In last decade, literature is rich with different proposal of node_id called as node labeling. These node_id worked as optimizer for query [2]. The node_id is designed such that it solved the problem of reconstruction and decision [10]. i.e.

1. *Reconstruction problem*:- With a given label of a node how parts of the tree structure of the database can be reconstructed.
2. *Decision problem*: By reading node label , how to determine relations between two nodes without accessing the database.

The first point suggests the XML tree construction process while the second point suggests the structural relationship between the XML nodes. This structural relationship solved the query execution effectively. All types of queries required the identification of the structural relationship. Generally researchers categorized queries into four types: (1) Tree Structure queries: - which are further classified into two types : Simple Queries and Twig queries which determine/based on structural relationship. (2) Start node queries which further classify total matching (complete path matching) and partial matching (3) Containment relationship (4) Ordering queries: - the queries can traverse the XML tree in forward or reverse direction. All these queries required structural relationships between nodes i.e. parent-child relationships, ancestor decedent relationship and sibling relationship. User queries always fall into one of these categories. Therefore, the main goal of the existing labeling schema is: - a) provide unique identity to each XML node b) preserve the nested hierarchical relationships so the query can answer with minimum processing time and minimum I/O cost.

The following parameters are considered while measuring numbering schema:-

1. Deterministic: The relationships between two nodes can be uniquely and quickly determined simply by examining their labels.
2. Dynamic: Updating XML files will not require the relabeling of nodes in the XML trees. In updating cost measurement, it is classified into: (1) updating cost of node (2) updating cost of index.

- Updating cost of Node: - The new node_id is assigned to each node of XML file. By using extensibility feature, XML is user can insert or delete node as per his/her requirement. This updating must be reflected into XML file without relabeling of the XML file. This parameter measure the cost of relabeling of the node.
- Index update cost : - This measure updates the time which it takes to update an index when updating a document. This includes insertion, deletion and modification of node in the XML file

3. Compact: The size of the labels should be minimal in order to fit in the main memory.
4. Flexible: The scheme can be used to support all kinds of XQuery/XPath functions.

6.6.4 Result and Analysis of Criteria I: Query Performance Measure

The experiment was performed for query evaluation speed, disk space measurement, disk traffic and scalability. The experiment was performed on new model "PESS" as well as on exiting model "Revere Path Approach". The results are displayed below:-

Study of Existing Method – Reverse Path Approach [1]

Introduction of Reverse Path Approach:-

Reverse Path Model is RDBMS storage technique. The study is as follows:-

This model has three important components: Data Loader, Query Interface and Query Translator. The Data Loader reads the input from the XML file. The XML document is then parsed and stored into relational tables. The Query Interface provides visual and command line query facilities. Query Translator translates XML query into RDBS format. This schema maps the XML files into five relational tables as follows:

- Label Path (ID, Len, and Path):-
  - This table stores the information of each label of the XML file. The label path and each label path is identified by using a unique ID. The 'Len' column stores the total number of edges of the same label. For example, if the label '*book*' occurs ten times, the term '*boo*k' is stored under '**ID'**; while number 'ten' would appear under '**Len**'. The column 'Path' records the path value.

- DataPath (PID, CID) :-
  - It maintains parent-child relationship. The ‘PID’ stores parent identifier and ‘CID’ stores child identifier.
- Ancestor table ((Did, Ancestor, Level):-
  - This table maintains ancestor relationship between two nodes. The ‘Did’ is a data path identifier which identifies a data path. The ‘Ancestor’ and ‘Level’ maintains ancestor descendent relationships. For example, a tuple (&9, &1, &2) indicates that the grand parent of node &9 is the node &1.
- Element (Pathid, Did, Ordinal):-
  - This table stores element information.
- Data (PathID, Did, Ordinal, Value):-
  - This table stores value node information.
- Each XML element is stored separately which reduces database size and avoids redundancy of information. This style of storage answers all types of queries. However the disadvantages of this approach are:
- It maintains a separate ancestor table for storing ancestor-descendent relationship which leads to two problems:-
  - Extra disk space:-The parent table and ancestor table are two alternatives to store the data-paths of XML data. Although ancestor table speeds up the query processing time, it takes too much disk space for storing the data.
  - Overhead for update operation: - The parent and ancestor table maintain redundant information which is overhead for the updating operation.
- Too many tables require excessive joins for complex query, and query with various combinations.
- The tables which store parent-child and ancestor-descendant relationships may grow. The table growth depends upon the complexity of the XML document. If table grows, eventually the database size also grows.

The experimental setup and query set , input and output criteria is explain below:-

Experiment Setup (input and output parameter) :-

XML is working as a defector standard for data exchange on web application. Hence it is an essential language for web application. This thesis proposes an effective indexing method for XML data storage and proposed new framework for indexing. To exhibit the advantage of the proposed technique, this chapter evaluates the performance of proposed indexing technique. The result is compared with the existing approach.

From [3] suggest the unique identification of node enhanced query execution cost. The PESS framework consists of labelling schema for each XML nodes. XML labelling schema is an alternative for query optimization [11] , different optimization algorithm is not required. Thus researcher wants to test new node label schema for it as well as for dynamic updates.

As per our study, three parameters are important for studying XML data for effective data management into tables. These parameters are height of the tree, width of the tree and type of storage. Therefore for the experiment the input is selected by considering:-

1. Selection of file: - Thesis is more focused on data centric application, hence the input files are broadly classified into two types: 'data centric' and 'mix mode' file.
2. Height of the tree:- To study the effectiveness of the algorithm, the XML file different tree levels are selected. So further two files are selected from each type (data centric' and 'mix mode'). The first file has fifth level and second file has level ten.
3. Width of the files:-Another important point of the study total number of records stored into the file. Therefore each file is further classified into six files, starting from 10 records till 100000 records in it. So total six files are selected.

1) File Structure :-

The input data set and query set is described in chapter 4, page no:91 and 93. Total sixteen XML files and eighteen queries were considered for the experiment.

Index Structure:- The following table 6.3 display index strategy:-

Table 6.3: Index Strategy of RPM and PESS

Existing Method	Table Structure	Index On Field
Reverse Path Approach	(ID primary key, XDOC XML)	Primary Index on ORDNODE attribute
	The table is shredded and the new table is created as	Secondary index on value ,Path and property
	(ORDPATH, TAG, NODE_TYPE VALUE, PATH)	
PESS Model Approach	P-table(Pah_id, Node_id, type of node)	B+ tree index on 'Path_id'
	D-table(Node_id, value);	Node_id Index
	N-table(Node_id, c_id)	Node_id Index
	1) Coarse- Grain type	Path id attribute
	2) Medium-Grain Type	Path _id attribute + Node_id attribute

Experiment Output:-

This section studies the impact of index structure on query evaluation. Query execution time, disk traffic, disk space and scalabilities are comparative criteria of query evaluation. The following output are generated :-

- Result and Analysis of Existing method.- Reverse Path Approach
- Result and Analysis of PESS Model
- PESS Model - Comparative study of Order queries
    - PESS Model - Comparative study of FLOWR queries
    - Comparative study of index structure with 'PESS' model and 'Reverse Path Approach'
- Comparative Study of same index structure (PESS Model) with the varying data set
- Comparative Study of index structure (PESS Model) on different type of document.

6.6.4.1 Result and Analysis of Existing method.- Reverse Path Approach

One of the popular examples of storing XML file is on RDBMS storage. The XML data is stored into the different tables which increased number of node join operation for query evaluation. To reduce number of node join [6] proposed the PM approach. Basic idea for the approach is that the XML data is stored on path value instead of storing it on node-id values which will reduce number of join operation. The B+tree index is created on path value. The query engine searches the query string on the storage. This matching and searching string increased query processing cost and cannot process "%like%" type of string operation on it. The new proposal [7] proposed new idea of storing path value. It stores path information in reverse order so the first tag differentiates the label. It creates a table with an integer column and an XML column as follows: **(ID int primary key , XDOC XML)**

The XML values saved in the XDOC column can be trees ("XML document") or fragments ("XML contents"). They are stored in an internal, binary representation that is stream able and optimize for query processing. XML file is shredded and B+ tree index is created. The table structure is **(ORDPATH, TAG, NODE_TYPE VALUE, PATH)**

Where the node labeling schema called ORDPATH [88] is used which captures document order and document hierarchy, the 'TAG' is captures the XML tag name, Node_type defines whether it is element node, text node (value node) or attribute node. The Path column stores the XML Path information. Three secondary indexes are created – PATH, PROPERTY and VALUE to optimize path based queries, value index and order queries. The data analysis presented below:-

6.6.4.1.1 Experiment output of Data Set 1- Set-1 :-

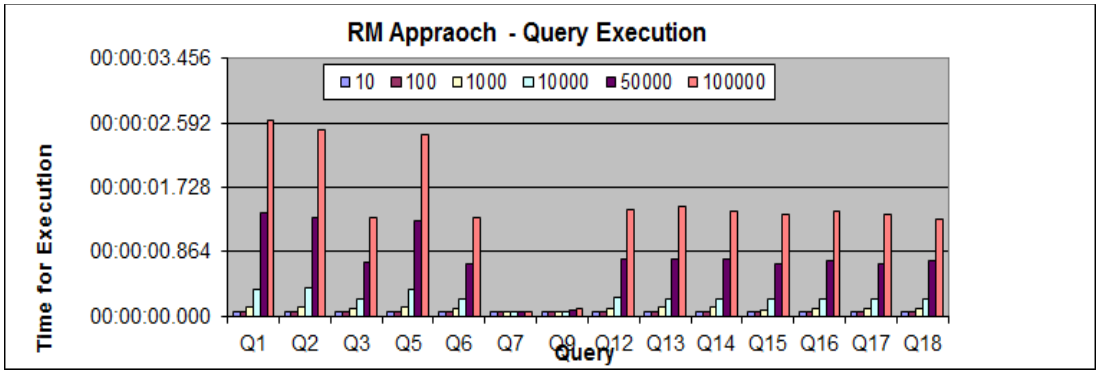
1. **Query Performance Result** :-

Set 1 :- 'Cd.XML' File :- This is a simple file with depth four . The P-C and A-D relationship is simple. The different files are considered with different records set: 10 records, 100 records, 1000 records, 10,000 records, 50000 records and 100000. And these files are having records from the different queries that are executed on it. The query Q1 and Q2 are simple queries Q3 till Q12 are order queries and Q13 till Q18 are twig pattern query. The result is mapped into Graph 6.1

Table 6.4 : Reverse Path Approach -Query execution Time for CD.XML

Query	10	100	1000	10000	50000	100000
Q1	00:00:00.063	00:00:00.063	00:00:00.110	00:00:00.359	00:00:01.375	00:00:02.625
Q2	00:00:00.047	00:00:00.062	00:00:00.125	00:00:00.375	00:00:01.328	00:00:02.500
Q3	00:00:00.063	00:00:00.062	00:00:00.094	00:00:00.219	00:00:00.719	00:00:01.313
Q5	00:00:00.062	00:00:00.062	00:00:00.109	00:00:00.360	00:00:01.281	00:00:02.438
Q6	00:00:00.062	00:00:00.063	00:00:00.094	00:00:00.219	00:00:00.703	00:00:01.313
Q7	00:00:00.063	00:00:00.063	00:00:00.062	00:00:00.062	00:00:00.063	00:00:00.063
Q9	00:00:00.047	00:00:00.047	00:00:00.062	00:00:00.063	00:00:00.078	00:00:00.094
Q12	00:00:00.063	00:00:00.062	00:00:00.094	00:00:00.250	00:00:00.766	00:00:01.422
Q13	00:00:00.062	00:00:00.062	00:00:00.125	00:00:00.219	00:00:00.766	00:00:01.469
Q14	00:00:00.063	00:00:00.063	00:00:00.109	00:00:00.234	00:00:00.766	00:00:01.407
Q15	00:00:00.047	00:00:00.047	00:00:00.078	00:00:00.235	00:00:00.688	00:00:01.360
Q16	00:00:00.063	00:00:00.062	00:00:00.093	00:00:00.219	00:00:00.734	00:00:01.407
Q17	00:00:00.062	00:00:00.062	00:00:00.094	00:00:00.219	00:00:00.688	00:00:01.359
Q18	00:00:00.062	00:00:00.062	00:00:00.093	00:00:00.234	00:00:00.734	00:00:01.297

Graph 6.1: Reverse Path Approach -Query execution Time for CD.XML



2. **Disk Space:-** Disk space is measured as per the following rules: -

Creation of database and different tables: The 'table space' information is obtained from the system. This information notifies the page size, total number of rows in the table, average row length, and total number of pages required for the database file. The information obtained through these parameters is used for calculating the disk space, using the following formula:-

Disk size of the file: = (Total number of rows * average row Length) *

(Total number of Pages * Page Size)

The storage space are listed below:-

File Name	Storage Space	Storage Space
SET 1.1.1	10 - nodes	4.00 KB
SET 1.1.2	100 - nodes	20 KB
SET 1.1.3	1000 - nodes	160 KB
SET 1.1.4	10000 - nodes	1.68 MB
SET 1.1.5	50000 - nodes	8.41 MB
SET 1.1.6	100000 – nodes	15.1 MB

3. **Disk traffic:-** The query execution plan provides the 'cardinality', 'selectivity', and 'total buffer pool pages' used in query evaluation, CPU cost, I/O instruction cost. The disk traffic is nothing but total number of buffer pages and the I/O cost required for query execution for the particular cardinality, and selectivity with the specific CPU cost. The disk I/O cost is as follows:-

File Name	Storage Space	I/O Cost
SET 1.1.1	10 - nodes	1
SET 1.1.2	100 - nodes	1
SET 1.1.3	1000 - nodes	1
SET 1.1.4	10000 - nodes	1
SET 1.1.5	50000 - nodes	3
SET 1.1.6	100000 – nodes	4

6.6.4.1.2 Experiment Output of Data Set-I –set-II

6.6.4.1.3

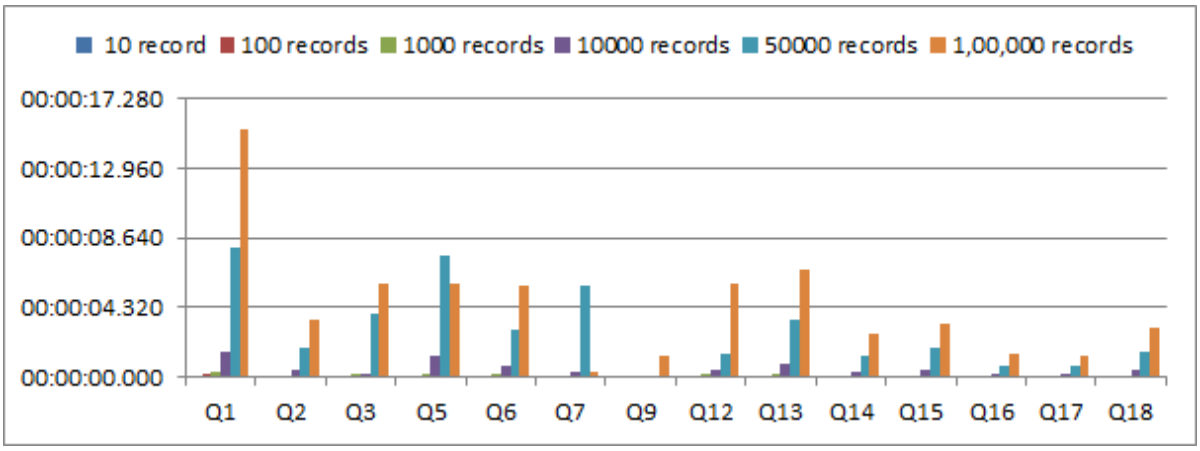
'**WS.XML' File:-**. The file has nested relationship. The A-D relationship is extended till level ten. Sibling relationship is varying from 10 records till 100000 records. Total six files are considered and observed the query execution pattern. The query Q1 and Q2 are simple queries, Q3 till Q12 are order queries and Q13 till Q18 are twig pattern query.

1 Query Execution Time: - the comparative study of query execution time is in the following Graph 6.2.

Table 6.5 Reverse Path Approach -Query execution Time for WS.XML

Query	10 record	100 records	1000 records	10000 records	50000 records	1,00,000 records
Q1	00:00:00.063	00:00:00.172	00:00:00.297	00:00:01.610	00:00:08.000	00:00:15.390
Q2	00:00:00.047	00:00:00.062	00:00:00.110	00:00:00.437	00:00:01.844	00:00:03.516
Q3	00:00:00.062	00:00:00.078	00:00:00.156	00:00:00.156	00:00:03.875	00:00:05.812
Q5	00:00:00.063	00:00:00.094	00:00:00.218	00:00:01.266	00:00:07.515	00:00:05.844
Q6	00:00:00.063	00:00:00.078	00:00:00.156	00:00:00.688	00:00:02.968	00:00:05.657
Q7	00:00:00.063	00:00:00.063	00:00:00.062	00:00:00.328	00:00:05.688	00:00:00.375
Q9	00:00:00.046	00:00:00.062	00:00:00.062	00:00:00.078	00:00:00.125	00:00:01.359
Q12	00:00:00.078	00:00:00.109	00:00:00.156	00:00:00.437	00:00:01.453	00:00:05.844
Q13	00:00:00.063	00:00:00.094	00:00:00.172	00:00:00.813	00:00:03.531	00:00:06.735
Q14	00:00:00.063	00:00:00.078	00:00:00.109	00:00:00.360	00:00:01.375	00:00:02.719
Q15	00:00:00.047	00:00:00.063	00:00:00.125	00:00:00.453	00:00:01.765	00:00:03.328
Q16	00:00:00.047	00:00:00.062	00:00:00.078	00:00:00.250	00:00:00.750	00:00:01.407
Q17	00:00:00.047	00:00:00.063	00:00:00.078	00:00:00.218	00:00:00.719	00:00:01.329
Q18	00:00:00.047	00:00:00.063	00:00:00.109	00:00:00.406	00:00:01.609	00:00:03.109

Graph 6.2: Reverse Path Approach -Query execution Time for WS.XML



2 Disk Space:- The calculation of disk space are recorded below:-

File Name	Storage Space	Storage Space
SET 2.1.1	10 - nodes	27 KB
SET 2.1.2	100 - nodes	256 KB
SET 2.1.3	1000 - nodes	3 MB
SET 2.1.4	10000 - nodes	32 MB
SET 2.1.5	50000 - nodes	148 MB
SET 2.1.6	100000 – nodes	178 MB

3 Disk traffic: - The disk traffic is calculated and display below:-

File Name	Storage Space	I/O Cost
SET 1.1.1	10 - nodes	1
SET 1.1.2	100 - nodes	1
SET 1.1.3	1000 - nodes	1
SET 1.1.4	10000 - nodes	7
SET 1.1.5	50000 - nodes	28
SET 1.1.6	100000 – nodes	38

6.6.4.1.4 Experiment Result of Data Set-II –Set 3

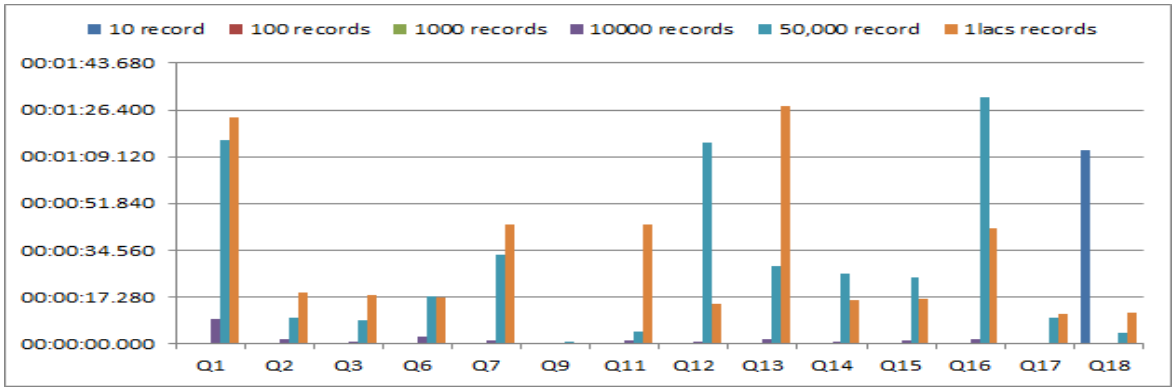
Set 3:- 'book.XML' File. The file has simple relationship. The A-D relationship is extended till level four. Sibling relationship is varying from 10 records till 100000 records. Total six files are considered and observed the query execution pattern. The query Q1 and Q2 are simple queries, Q3 till Q12 are order queries and Q13 till Q18 are twig pattern query.

1) **Query Execution Time** :- The query execution time of mix-mode approach is listed into Graph 6.3

Table 6.6: Reverse Path Approach -Query execution Time -book.XML

Query	10 record	100 records	1000 records	10000 records	50,000 record	1lacs records
Q1	00:00:00.078	00:00:00.140	00:00:00.438	00:00:08.969	00:01:15.188	00:01:23.390
Q2	00:00:00.047	00:00:00.078	00:00:00.297	00:00:01.547	00:00:09.766	00:00:19.125
Q3	00:00:00.047	00:00:00.125	00:00:00.235	00:00:00.750	00:00:08.594	00:00:17.766
Q6	00:00:00.125	00:00:00.157	00:00:00.391	00:00:02.766	00:00:17.422	00:00:17.000
Q7	00:00:00.063	00:00:00.109	00:00:00.344	00:00:01.344	00:00:32.937	00:00:43.875
Q9	00:00:00.281	00:00:00.063	00:00:00.078	00:00:00.125	00:00:00.531	00:00:00.157
Q11	00:00:00.047	00:00:00.063	00:00:00.125	00:00:01.156	00:00:04.656	00:00:44.016
Q12	00:00:00.062	00:00:00.062	00:00:00.187	00:00:00.640	00:01:14.500	00:00:14.672
Q13	00:00:00.046	00:00:00.078	00:00:00.234	00:00:01.718	00:00:28.657	00:01:27.891
Q14	00:00:00.062	00:00:00.109	00:00:00.094	00:00:00.937	00:00:26.062	00:00:15.985
Q15	00:00:00.079	00:00:00.109	00:00:00.234	00:00:01.313	00:00:24.281	00:00:16.672
Q16	00:00:00.062	00:00:00.078	00:00:00.125	00:00:01.500	00:01:30.922	00:00:42.813
Q17	00:00:00.094	00:00:00.047	00:00:00.109	00:00:00.437	00:00:09.718	00:00:10.937
Q18	00:01:11.594	00:00:00.047	00:00:00.234	00:00:00.438	00:00:03.937	00:00:11.266

Graph 6.3: Reverse Path Approach -Query execution Time -book.XML



2) Disk Space :- The disk space requirement is as follows:-

File Name	Storage Space	Storage Space
SET 2.1.1	10 - nodes	4 KB
SET 2.1.2	100 - nodes	8.6 KB
SET 2.1.3	1000 - nodes	76KB
SET 2.1.4	10000 - nodes	792 KB
SET 2.1.5	50000 - nodes	3.86 MB
SET 2.1.6	100000 – nodes	7.73 MB

3) **Disk traffic**:- The Disk traffic data are stored below:-

File Name	Storage Space	I/O Cost
SET 1.1.1	10 - nodes	1
SET 1.1.2	100 - nodes	1
SET 1.1.3	1000 - nodes	1
SET 1.1.4	10000 - nodes	1
SET 1.1.5	50000 - nodes	1
SET 1.1.6	100000 – nodes	2

6.6.4.1.5 Experiment Result of Data Set-II –Set 4

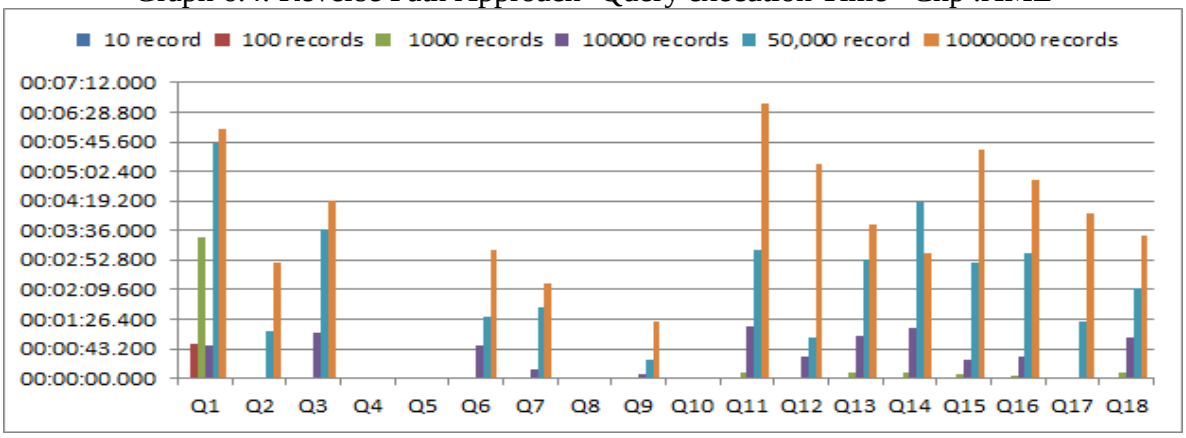
(1) 'Chp..XML' :- File stores mix mode style of data. This file is a combination of data as well as document. The file stores nested relationship between elements. The A-D relationship is extended till level ten. Sibling relationship is varying from 10 records till 100000 records. Total six files are considered and observed the query execution pattern. The query Q1 and Q2 are simple queries, Q3 till Q12 are order queries and Q13 till Q18 are twig pattern query.

1. **Query Execution Time**:- The mix mode file is recorded in graph 6.4

Table 6.7 Reverse Path Approach -Query execution Time –Chp .XML

Query	10 record	100 records	1000 recor	10000 records	50,000 record	1000000
Q1	00:00:00.234	00:00:51.312	00:03:25.062	00:00:48.094	00:05:43.188	00:06:05.250
Q2	00:00:00.047	00:00:00.297	00:00:00.422	00:00:01.531	00:01:09.672	00:02:49.063
Q3	00:00:00.078	00:00:00.203	00:00:01.453	00:01:06.594	00:03:36.641	00:04:19.281
Q4	----------			----------		
Q5				----------		
Q6	00:00:00.078	00:00:00.203	00:00:01.453	00:00:48.297	00:01:29.609	00:03:06.469
Q7	00:00:00.078	00:00:00.265	00:00:02.062	00:00:12.265	00:01:43.062	00:02:19.625
Q8						
Q9	00:00:00.062	00:00:00.110	00:00:00.859	00:00:06.031	00:00:26.406	00:01:23.562
Q10				----------		
Q11	00:00:00.250	00:00:00.469	00:00:09.250	00:01:15.797	00:03:06.360	00:06:42.094
Q12	00:00:00.093	00:00:00.297	00:00:02.156	00:00:32.203	00:00:58.562	00:05:13.157
Q13	00:00:00.203	00:00:00.297	00:00:07.922	00:01:01.031	00:02:53.562	00:03:44.781
Q14	00:00:00.078	00:00:00.172	00:00:07.438	00:01:13.547	00:04:17.187	00:03:02.000
Q15	00:00:00.266	00:00:00.360	00:00:06.297	00:00:27.093	00:02:48.515	00:05:34.297
Q16	00:00:00.094	00:00:00.297	00:00:04.813	00:00:31.234	00:03:01.828	00:04:50.469
Q17	00:00:00.141	00:00:00.078	00:00:00.390	00:00:01.422	00:01:23.390	00:04:01.703
Q18	00:00:00.078	00:00:00.250	00:00:07.453	00:01:00.625	00:02:12.422	00:03:29.391

Graph 6.4: Reverse Path Approach -Query execution Time –Chp .XML



2. Disk Space **:-** The disk space information is displayed below

File Name	Storage Space	Storage Space
SET 2.1.1	10 - nodes	1.06 MB
SET 2.1.2	100 - nodes	3.14 MB
SET 2.1.3	1000 - nodes	12.7 MB
SET 2.1.4	10000 - nodes	26.12 MB
SET 2.1.5	50000 - nodes	53.7 MB
SET 2.1.6	100000 – nodes	114 MB

3. Disk traffic**:-** Disk traffic information displayed below

File Name	Storage Space	I/O Cost
SET 1.1.1	10 - nodes	1
SET 1.1.2	100 - nodes	1
SET 1.1.3	1000 - nodes	3
SET 1.1.4	10000 - nodes	6
SET 1.1.5	50000 - nodes	11
SET 1.1.6	100000 – nodes	13

6.6.4.2 Result and Analysis of proposed model - PESS Model:-

The first step of the experiment read the XML file using java the parser. The programme read each XML node and generate node_id and path_id . These two attributes are stored into XML file back. This newly created file is shredded and stored into IBM DB2 database. DB2 provides two choices for storage: RDBMS storage and native storage. The query performance is

measured by Java Execution Time Measurement Library (JETM). In the RDBMS storage, B+tree index is created on 'Path_id' and secondary index is created on 'Node_id'.

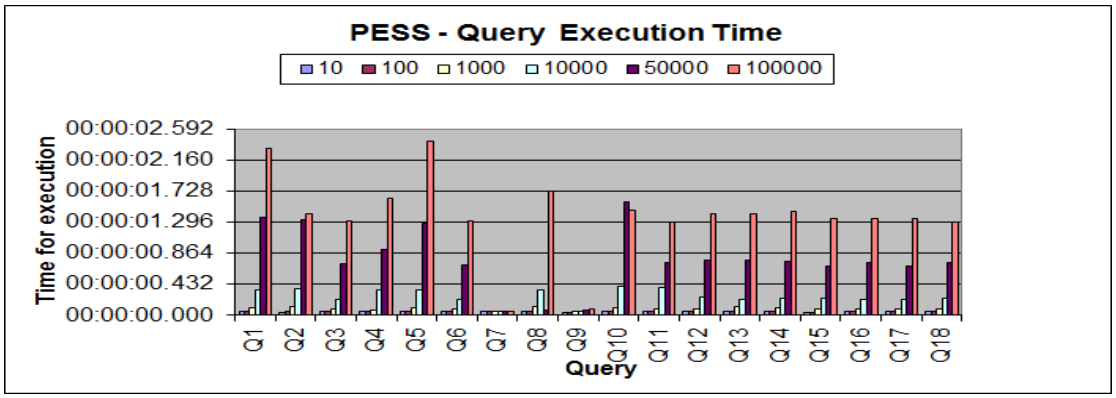
6.6.4.2.1 Experiment Result of Data Set-I:- Set 1: 'Cd.XML ' file :-

1. **Query Execution Time:-** Graph 6.5 shows the query execution time of PESS model for 'CD.XML' file.

Table 6.8 :-PESS -Query execution Time -CD.XML

Query	10	100	1000	10000	50000	100000
Q1	00:00:00.061	00:00:00.061	00:00:00.108	00:00:00.357	00:00:01.371	00:00:02.325
Q2	00:00:00.048	00:00:00.061	00:00:00.123	00:00:00.373	00:00:01.324	00:00:01.422
Q3	00:00:00.063	00:00:00.063	00:00:00.090	00:00:00.217	00:00:00.713	00:00:01.308
Q4	00:00:00.061	00:00:00.062	00:00:00.077	00:00:00.358	00:00:00.922	00:00:01.625
Q5	00:00:00.062	00:00:00.061	00:00:00.108	00:00:00.358	00:00:01.278	00:00:02.426
Q6	00:00:00.061	00:00:00.062	00:00:00.093	00:00:00.217	00:00:00.698	00:00:01.307
Q7	00:00:00.063	00:00:00.062	00:00:00.061	00:00:00.060	00:00:00.059	00:00:00.061
Q8	00:00:00.060	00:00:00.063	00:00:00.124	00:00:00.360	00:00:00.078	00:00:01.735
Q9	00:00:00.047	00:00:00.047	00:00:00.061	00:00:00.061	00:00:00.074	00:00:00.090
Q10	00:00:00.060	00:00:00.062	00:00:00.109	00:00:00.408	00:00:01.578	00:00:01.469
Q11	00:00:00.062	00:00:00.062	00:00:00.093	00:00:00.391	00:00:00.734	00:00:01.297
Q12	00:00:00.062	00:00:00.061	00:00:00.092	00:00:00.248	00:00:00.762	00:00:01.418
Q13	00:00:00.063	00:00:00.062	00:00:00.122	00:00:00.217	00:00:00.762	00:00:01.419
Q14	00:00:00.064	00:00:00.061	00:00:00.108	00:00:00.232	00:00:00.760	00:00:01.455
Q15	00:00:00.046	00:00:00.047	00:00:00.093	00:00:00.233	00:00:00.684	00:00:01.352
Q16	00:00:00.063	00:00:00.062	00:00:00.093	00:00:00.216	00:00:00.729	00:00:01.341
Q17	00:00:00.062	00:00:00.062	00:00:00.094	00:00:00.216	00:00:00.684	00:00:01.350
Q18	00:00:00.061	00:00:00.062	00:00:00.091	00:00:00.233	00:00:00.734	00:00:01.293

Graph 6.5: PESS -Query execution Time -CD.XML



2. **Disk Space** :- (PESS – Model): The required disk space for PESS model is listed below :-

	File Name	Size	Index Size
Set 1.1.1	10 Nodes	4KB	4KB
Set 1.1.2	100 Nodes	12KB	4KB
Set 1.1.3	1000 Nodes	84 KB	40 KB
Set 1.1.4	10000 Nodes	820 KB	88 KB
Set 1.1.5	50000 Nodes	4088 KB	428 KB
Set 1.1.6	100000 Nodes	8244 KB	860 KB

3. **Disk Traffic** :- The disk traffic of PESS Model are listed below:-

	File Name	Pages Fetch for Query execution in (Without indexing)	Pages Fetch for Query execution (With Indexing)
Set 1.1.1	10 Nodes	1	1
Set 1.1.2	100 Nodes	3	1
Set 1.1.3	1000 Nodes	21	10
Set 1.1.4	10000 Nodes	205	22
Set 1.1.5	50000 Nodes	1022	107
Set 1.1.6	100000 Nodes	2061	251

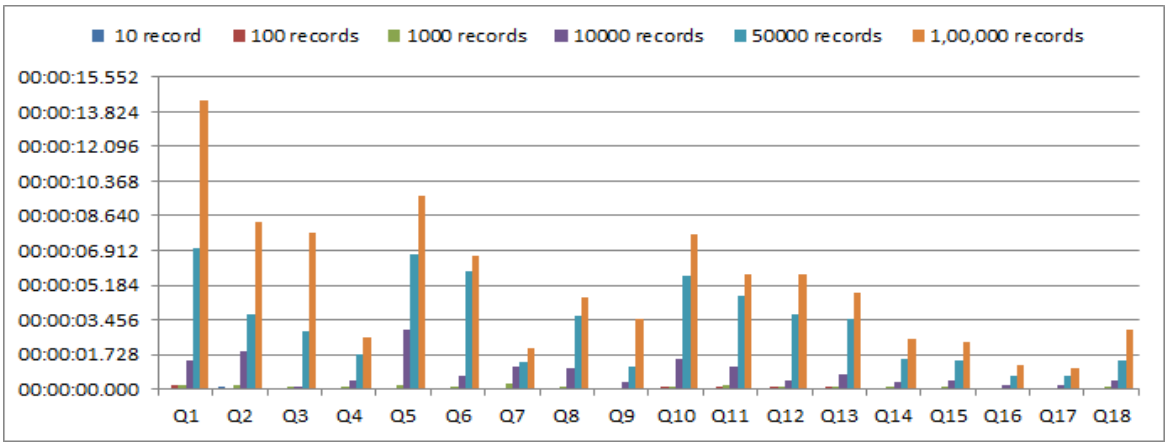
6.6.4.2.2 Experiment Result of Data Set-I:- Set 2: 'WS.XML ' file

1. **Query Execution Time:-** Graph 6.6 show the query execution time for ws.xml file

Table 6.9 PESS -Query execution Time -WS.XML

Query	10 record	100 records	1000 records	10000 records	50000 records	1,00,000 records
Q1	00:00:00.062	00:00:00.182	00:00:00.225	00:00:01.408	00:00:07.020	00:00:14.390
Q2	00:00:00.089	00:00:00.063	00:00:00.187	00:00:01.888	00:00:03.703	00:00:08.341
Q3	00:00:00.061	00:00:00.076	00:00:00.151	00:00:00.151	00:00:02.875	00:00:07.792
Q4	00:00:00.048	00:00:00.060	00:00:00.139	00:00:00.428	00:00:01.740	00:00:02.580
Q5	00:00:00.061	00:00:00.069	00:00:00.200	00:00:02.947	00:00:06.735	00:00:09.672
Q6	00:00:00.062	00:00:00.054	00:00:00.153	00:00:00.683	00:00:05.844	00:00:06.641
Q7	00:00:00.061	00:00:00.060	00:00:00.299	00:00:01.086	00:00:01.375	00:00:02.068
Q8	00:00:00.046	00:00:00.060	00:00:00.151	00:00:01.062	00:00:03.657	00:00:04.588
Q9	00:00:00.046	00:00:00.061	00:00:00.058	00:00:00.328	00:00:01.153	00:00:03.531
Q10	00:00:00.059	00:00:00.089	00:00:00.122	00:00:01.500	00:00:05.657	00:00:07.703
Q11	00:00:00.062	00:00:00.093	00:00:00.213	00:00:01.153	00:00:04.657	00:00:05.715
Q12	00:00:00.061	00:00:00.099	00:00:00.133	00:00:00.394	00:00:03.703	00:00:05.692
Q13	00:00:00.061	00:00:00.093	00:00:00.160	00:00:00.736	00:00:03.531	00:00:04.805
Q14	00:00:00.059	00:00:00.074	00:00:00.102	00:00:00.332	00:00:01.531	00:00:02.519
Q15	00:00:00.063	00:00:00.061	00:00:00.121	00:00:00.434	00:00:01.465	00:00:02.328
Q16	00:00:00.047	00:00:00.062	00:00:00.076	00:00:00.228	00:00:00.624	00:00:01.220
Q17	00:00:00.047	00:00:00.061	00:00:00.076	00:00:00.215	00:00:00.636	00:00:01.029
Q18	00:00:00.047	00:00:00.061	00:00:00.105	00:00:00.399	00:00:01.453	00:00:02.998

Graph 6.6: PESS -Query execution Time -WS.XML



2. Disk Space :- (PESS Model**)** :- Disk space is mapped as follows:-

	File Name	Table Size	Index Size
Set 1.2.1	10 Nodes	12 KB	4 KB
Set 1.2.2	100 Nodes	84 KB	4 KB
Set 1.2.3	1000 Nodes	804 KB	20 KB
Set 1.2.4	10000 Nodes	8008 KB	168 KB
Set 1.2.5	50000 Nodes	40036 KB	836 KB
Set 1.2.6	100000 Nodes	80068 KB	1668 KB

3. Disk Traffic: - (I/O Cost - PESS Model is as follows:-

	File Name	Pages Fetch for Query execution in (Without indexing)	Pages Fetch for Query execution (With Indexing)
Set 1.2.1	10 Nodes	3	1
Set 1.2.2	100 Nodes	21	1
Set 1.2.3	1000 Nodes	201	5
Set 1.2.4	10000 Nodes	2002	42
Set 1.2.5	50000 Nodes	10009	209
Set 1.2.6	100000 Nodes	20017	417

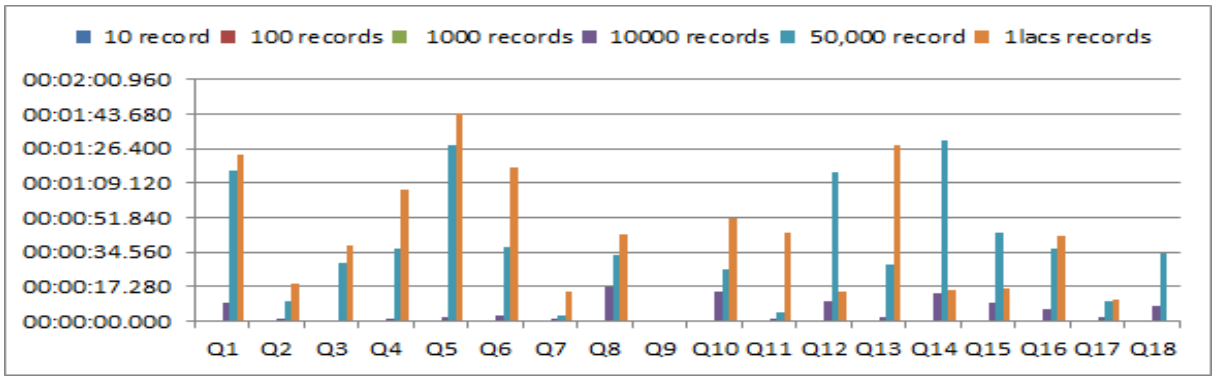
6.6.4.2.3 Experiment Result of Data Set-II- Set 3:- 'book.XML ' file :-

(1) **Query Execution Time:-** The query execution time for the book.xml file is in Graph 6.7

Table 6.10 : PESS -Query execution Time -book.XML

Query	10 record	100 records	1000 records	10000 records	50,000 record	1lacs records
Q1	00:00:00.077	00:00:00.138	00:00:00.436	00:00:08.869	00:01:15.182	00:01:23.379
Q2	00:00:00.045	00:00:00.076	00:00:00.295	00:00:01.523	00:00:09.760	00:00:19.115
Q3	00:00:00.123	00:00:00.155	00:00:00.389	00:00:00.647	00:00:29.000	00:00:38.000
Q4	00:00:00.123	00:00:00.155	00:00:00.538	00:00:01.234	00:00:36.000	00:01:06.000
Q5	00:00:00.077	00:00:00.138	00:00:00.422	00:00:02.172	00:01:28.000	00:01:44.000
Q6	00:00:00.045	00:00:00.123	00:00:00.233	00:00:02.762	00:00:37.000	00:01:17.000
Q7	00:00:00.051	00:00:00.051	00:00:00.076	00:00:01.340	00:00:02.671	00:00:14.875
Q8	00:00:00.045	00:00:00.047	00:00:00.156	00:00:17.109	00:00:32.930	00:00:43.865
Q9	00:00:00.060	00:00:00.076	00:00:00.122	00:00:00.121	00:00:00.525	00:00:00.150
Q10	00:00:00.051	00:00:00.051	00:00:00.199	00:00:14.875	00:00:26.000	00:00:51.765
Q11	00:00:00.045	00:00:00.061	00:00:00.232	00:00:01.150	00:00:04.650	00:00:44.006
Q12	00:00:00.060	00:00:00.059	00:00:00.185	00:00:10.000	00:01:14.494	00:00:14.666
Q13	00:00:00.044	00:00:00.076	00:00:00.232	00:00:01.714	00:00:28.650	00:01:27.883
Q14	00:00:00.060	00:00:00.092	00:00:00.107	00:00:14.000	00:01:30.918	00:00:15.975
Q15	00:00:00.077	00:00:00.105	00:00:00.232	00:00:09.000	00:00:44.000	00:00:16.666
Q16	00:00:00.060	00:00:00.076	00:00:00.122	00:00:06.000	00:00:36.000	00:00:42.806
Q17	00:00:00.045	00:00:00.092	00:00:00.107	00:00:02.000	00:00:09.714	00:00:10.928
Q18	00:00:00.045	00:00:00.061	00:00:00.123	00:00:08.000	00:00:34.000	

Graph 6.7: PESS -Query execution Time -book.XML



(2) **Disk Space (PESS):-** The disk space is as follow:-

	File Name	Size	Index Size
Set 2.1.1	10 Nodes	320 KB	32 KB
Set 2.1.2	100 Nodes	3200 KB	32 KB
Set 2.1.3	1000 Nodes	64000 KB	512 KB
Set 2.1.4	10000 Nodes	320000 KB	320 KB
Set 2.1.5	50000 Nodes	1664000KB	864 KB
Set 2.1.6	100000 Nodes	3552000 KB	1824 KB

Disk Traffic (PESS) : Disk traffic measurement is as follows:-

	File Name	Pages Fetch for Query execution in (Without indexing)	Pages Fetch for Query execution (With Indexing)
Set 2.1.1	10 Nodes	10	1
Set 2.1.2	100 Nodes	100	1
Set 2.1.3	1000 Nodes	2000	16
Set 2.1.4	10000 Nodes	10000	10
Set 2.1.5	50000 Nodes	52000	27
Set 2.1.6	100000 Nodes	111000	57

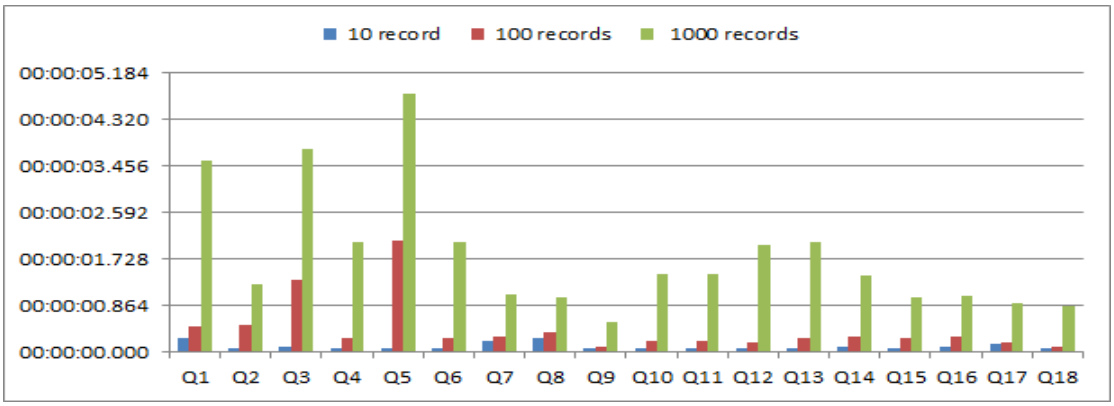
6.6.4.2.4 Experiment Result of Data Set-II- Set 4:- 'chp.XML ' file :-

1. **Query Execution Time:-**

Table 6.11: PESS -Query execution Time -chp.XML

Query	10 record	100 records	1000 records	10000 records	50,000 record
Q1	00:00:00.249	00:00:00.467	00:00:03.572	00:03:36.637	00:04:19.269
Q2	00:00:00.060	00:00:00.516	00:00:01.250	00:00:01.255	00:04:46.578
Q3	00:00:00.093	00:00:01.328	00:00:03.766	00:05:43.188	00:06:05.240
Q4	00:00:00.077	00:00:00.263	00:00:02.057	00:02:53.566	00:03:44.772
Q5	00:00:00.063	00:00:02.078	00:00:04.813	00:01:55.937	00:04:41.282
Q6	00:00:00.077	00:00:00.263	00:00:02.057	00:00:04.953	00:10:41.500
Q7	00:00:00.202	00:00:00.295	00:00:01.081	00:01:43.057	00:02:19.611
Q8	00:00:00.265	00:00:00.357	00:00:01.020	00:02:48.511	00:05:34.288
Q9	00:00:00.078	00:00:00.109	00:00:00.547	00:01:55.797	00:04:36.844
Q10	00:00:00.077	00:00:00.201	00:00:01.449	00:01:29.603	00:03:06.451
Q11	00:00:00.079	00:00:00.201	00:00:01.449	00:03:06.355	00:06:42.084
Q12	00:00:00.079	00:00:00.170	00:00:01.992	00:00:58.556	00:05:13.144
Q13	00:00:00.077	00:00:00.263	00:00:02.057	00:02:53.566	00:03:44.772
Q14	00:00:00.094	00:00:00.295	00:00:01.427	00:04:17.181	00:03:01.993
Q15	00:00:00.079	00:00:00.248	00:00:01.028	00:02:12.416	00:03:29.383
Q16	00:00:00.093	00:00:00.295	00:00:01.040	00:03:01.823	00:04:50.460
Q17	00:00:00.142	00:00:00.174	00:00:00.915	00:01:23.383	00:04:01.694
Q18	00:00:00.061	00:00:00.108	00:00:00.856	00:00:26.401	00:01:22.762

Graph 6.8: PESS -Query execution Time -chp.XML



2. **Disk Space**:-

	File Name	Size	Index Size
Set 2.2.1	10 Nodes	40	4
Set 2.2.2	100 Nodes	400	4
Set 2.2.3	1000 Nodes	4000	20
Set 2.2.4	10000 Nodes	40000	168
Set 2.2.5	50000 Nodes	200000	836
Set 2.2.6	100000 Nodes	400000 (100000)	1668

Disk Traffic:-

	File Name	Size	Index Size
Set 2.2.1	10 Nodes	1	1
Set 2.2.2	100 Nodes	3	1
Set 2.2.3	1000 Nodes	25	8
Set 2.2.4	10000 Nodes	239	80
Set 2.2.5	50000 Nodes	50000	209
Set 2.2.6	100000 Nodes	100000	417

6.6.4.3 Conclusion of Criteria 1 – Query Performance Measurement

6.6.4.3.1 Result and Analysis of Query Performance Measurement :-

From the above graph the following observation are noted:-

1) Tree Traversal Query:- These queries traverse the tree in – forward and backward direction . Total eighteen queries were used for measurement. These queries were classified into two group: A) Following/Following-sibling queries – traverse tree in forward direction B) Preceding – preceding-sibling queries.

A. Forward direction traversal (Following/Following-sibling queries[refer chapter 4 page no :93]):- The queries 'Q1', 'Q3','Q4', 'Q5' query sets are forward direction . Out of

this query set ‘Q4’ is executed in minimum time while remaining queries has maximum time for traversal. Query strategy is ‘Q4’ query starts tree traversal in between of the XML tree; the remaining queries traverse a complete tree. The sample query execution time pattern was observed for data set 1, data set2, data set3. After analysis of query execution plan, ‘Q1’, and ‘Q3’ were used table-scan operation and ‘Q4’,’Q5’ used XSCAN operation i.e. indexing is used

B. Backward Direction Queries (Preceding-Preceding–sibling queries [refer chapter 4 page no :93]): - Queries ‘Q6’,’Q7’and ‘Q8’ are in this group. These query traverse the tree in backward direction. ‘Q6’ has maximum time for query execution while ‘Q7’ and ‘Q8’ has minimum time in query execution. The query execution plan suggest ‘Q6’ used index operation with group by clause and ‘Q7’ and ‘Q8’ used only index operation for retrieving desired information

2) Containment Query (represent A-D and P-C relationship [refer chapter 4: page no: 93]) :- The Query 10 is A-D (ancestor-descendent) query and Query11 is P-C (Parent-Child) relationship. These queries are executed on the same nodes of the XML tree. These queries are using index scan operation and have the same query execution time.

3) Twig Query (return a small XML tree [refer chapter 4: page no :93]):- The query set Q2, Q9,Q12,Q13,Q14 are in this group. All these query were used index operation in query execution, Q14 and Q13 used index operation followed by sorting operation. Q2 takes maximum time for query execution; Q9 has minimum time for query execution. ‘Q12’, ‘Q13’, ‘Q14’ are almost similar; Q12 has minimum time for execution while Q14 has minimum time in query execution. This implies the query execution time depend upon the number of nodes that query is returning.

4) Content matching query (return a particular node from XML tree [refer chapter 4: page no : 93]):- :- The query set ‘Q15’,’Q16’,’Q17’,’Q18’. All this query using index operation; Q16 and Q17 required minimum time in query execution because they retuning the nodes which are matched with a particular value. And in Q15 and Q18 the predicates are ‘<’ and ‘>’ operation hence it returns all nodes which are matching with predicate value.

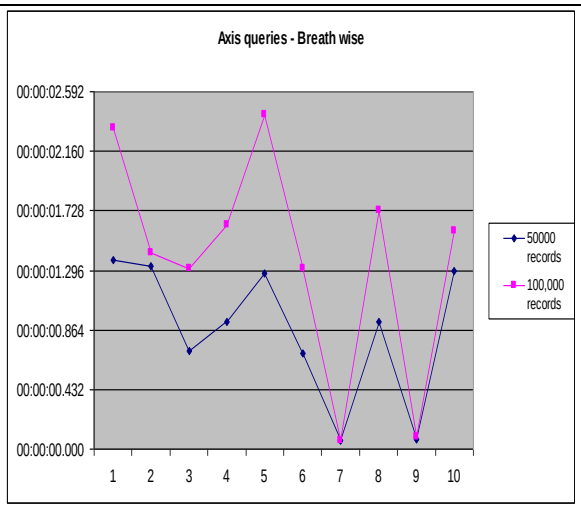
6.6.4.3.1.1 Comparative study Query Processing Time with different Data Set of PESS Model

For comparison these queries are classified into two groups: tree tarsal query and 'FLOWR' queries.

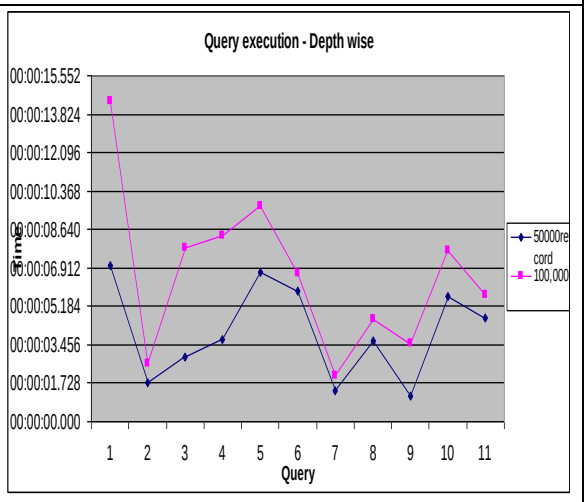
- Comparative study of Tree Traversal Query:- The following graph 6.9 mapped the query execution time

Graph 6.9: PESS Model - Comparative study of Axis Query

Data-Centric File

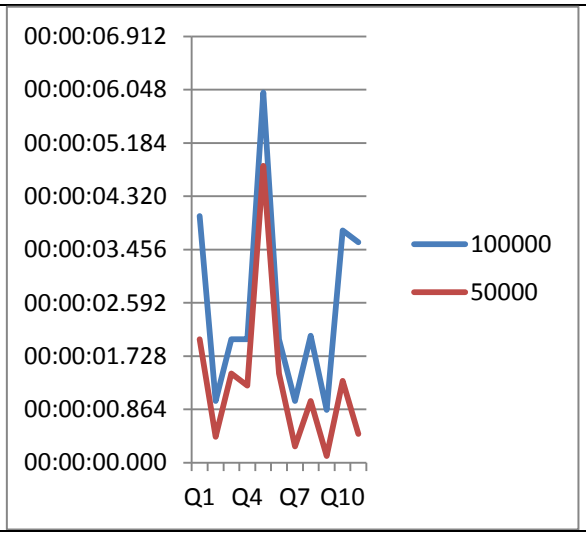


Axis Queries – Breadth Wise

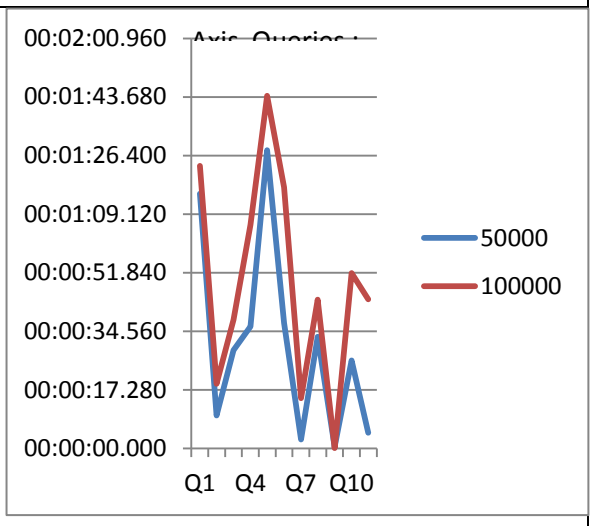


Axis Queries – Depth Wise

Mix-Mode File



Axis Queries – Breadth Wise



Axis Queries – Depth Wise

From all above graphs 6.9, the Q9 .i.e. 'self axis query' requires minimum time for query execution; Q5 and Q10 i.e. following-siblings and descendent axis query require maximum time for query evaluation. Q1 is used for displaying the complete XML tree which takes maximum time, because for evaluating this query, a complete table scans operation is performed instead of indexing operation. Q9 and Q7 are requiring almost the same time. In Q7 and Q9 retrieving only one node; I/O cost of the query is one. The I/O cost of the Q4 and Q8 is the same; hence the query execution time is also similar. This index operation is not applied on Q1 and Q3 the maximum query execution time is required for queries Q1, Q5. From the graph, the query execution is the best for those queries that are not restricted to hierarchical document structure and retuning a particular node of a tree. These queries are using index operation successfully and there is no node_id computational cost is involved hence the query execution time gets reduced.

The query execution time is high for all the queries which are returning a small tree structure. As the complexity of retrieving structure increases, the query exaction time also increases and so does the I/O cost.

Comparative study of FLOWR queries:-

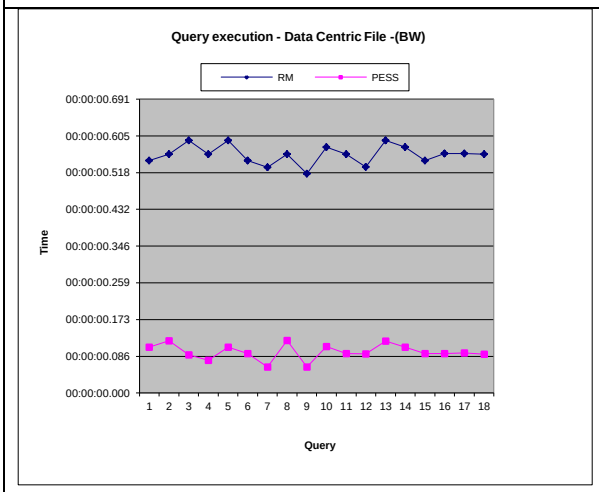
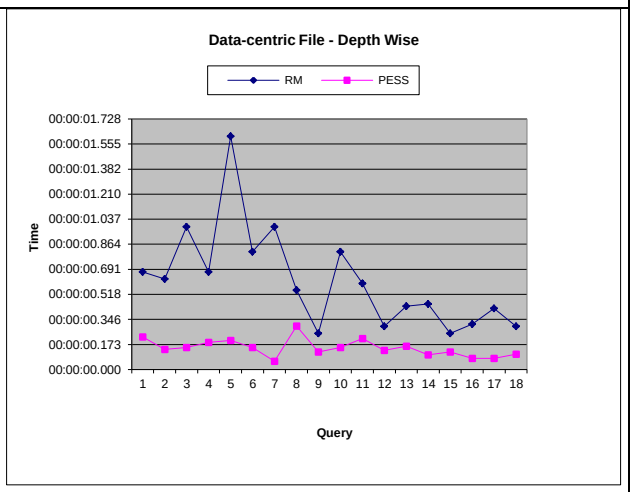
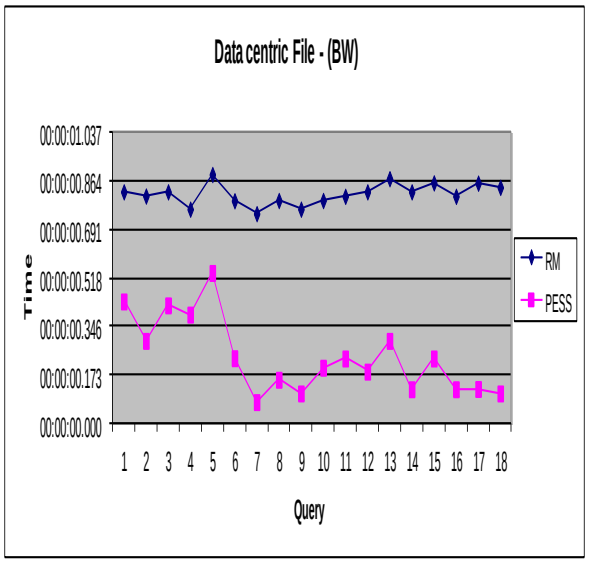
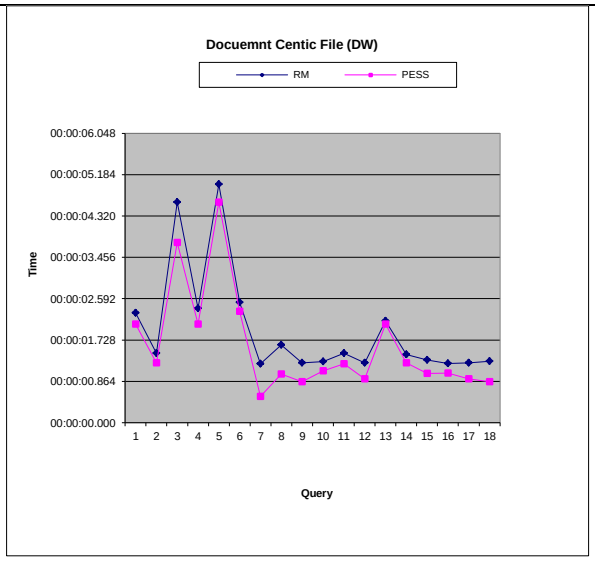
The query execution time is plotted into Graph 6.10.The FLOWR queries return two types of nodes: a simple chain path node, and tree node. These queries are classified into groups on their predicate value. The Q18, Q15 and Q14 have the same predicate value i.e. "Year < 1985"and the same selectivity and cardinality factor. The Q18 returns a simple chain path node while other two queries return a tree node, i.e. twig query. The Q15 returns only a branch of a tree while Q14 returns a complete tree structure. The query execution time for Q18 is less than Q15 and Q14.

The query execution time for Q15 is less than for Q14. It again proves query execution time depends upon the number of nodes return by the XML file. Another set of FLOWR queries are 'Q17, Q16 and Q12', the similar result is observed. These queries have the same predicate i.e. 'Year=1985', the same selectivity and cardinality factor. The Q17 returns a simple chain path node and Q16 and Q12 return twig node or tree node. The query execution time for Q17 is less than Q16 and Q12. And the query execution time is less for Q16 than the Q12.Thus, the study conclude that the query exaction time depends upon the two factors: S-path and R-path values [refer page no: 189] (S-path is search path and R-path is return path) and returning type of the node structure of the query. If the R-path is a part of S-path, the query execution time is less; if R-path is different from S-path the query execution time is higher.

6.6.4.3.1.2 Comparative study of Query Processing time of 'PESS' model and 'Reverse Path Approach **:-**

Graph 6.10: Comparative Study of Query Execution Time

(RM Approach storage Vs PESS Model)

<table>
<tr><td colspan="2">Query Execution Time – – Data Centric File</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Query Execution Time "Cd.XML"</td><td>Query Execution Time "WS.XML"</td></tr>
<tr><td colspan="2">Query Execution Time – – 'Mix-Mode' File</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Query Execution Time "Book.XML"</td><td>Query Execution Time "Chp.XML"</td></tr>
</table>

The above Graph 6.11 depicts the time in millisecond for query execution of the query. The graph represents that the query execution time for PESS model is less than RM approach The file storage schema and index structure is mention on (RM approach on page : and for PESS model it is on page) . This query execution time is reduced because 'PESS model avoid excessive join operations.

In 'PESS' model the operation is performed as follows:-

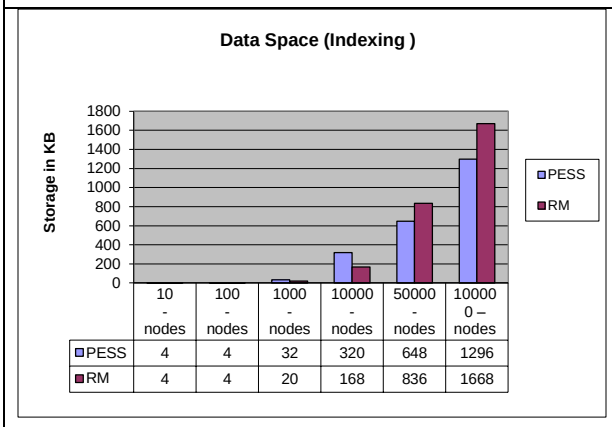
a) 'Path_id' determines all node_ids on that path.
b) Node_id determine the relationship between the nodes. Node_id computation will display desired node_id information.
c) These node_ids information are retrieved from node_table and element _table. (hash index is on node_id)

The graph shows that the query execution item pattern is similar for "CD.XML" file and "Book. XML" and "Ws.XML" and "Chp.XML". The PESS model is more effective when the file nesting is less and as the file nesting levels are increased, the I/O cost of PESS model is also increased hence index efficiency is lost when the depth of the file is high.

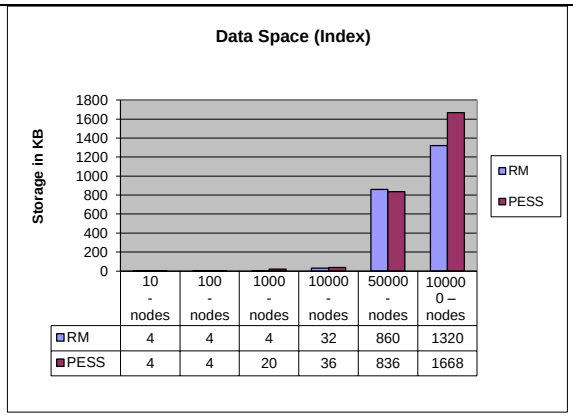
6.6.4.3.2 Results and Analysis of Disk Space of RM Approach storage and PESS Model

Graph6.11 Comparative Study of Disk Space (RM Approach storage Vs PESS Model)

Disk Space – Data Centric File

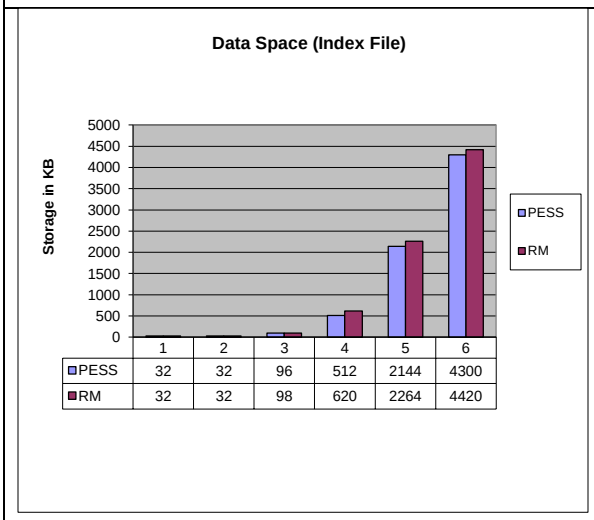


Disk Space – "cd.xml"

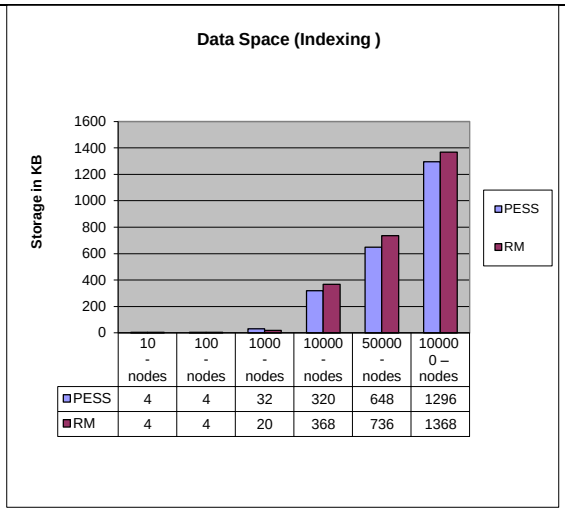


Disk Space – Ws.XML

Disk Space – Mix Mode File



Disk Space – "Book .XML"



Disk Space "Chp.XML"

For data centric file, the disk space is higher for PESS model while the disk space requirement is less for RM approach. This may be the average row length is larger than RM and PESS model created index on multiple values.

6.6.4.3.3 Result and Analysis of Index Structure

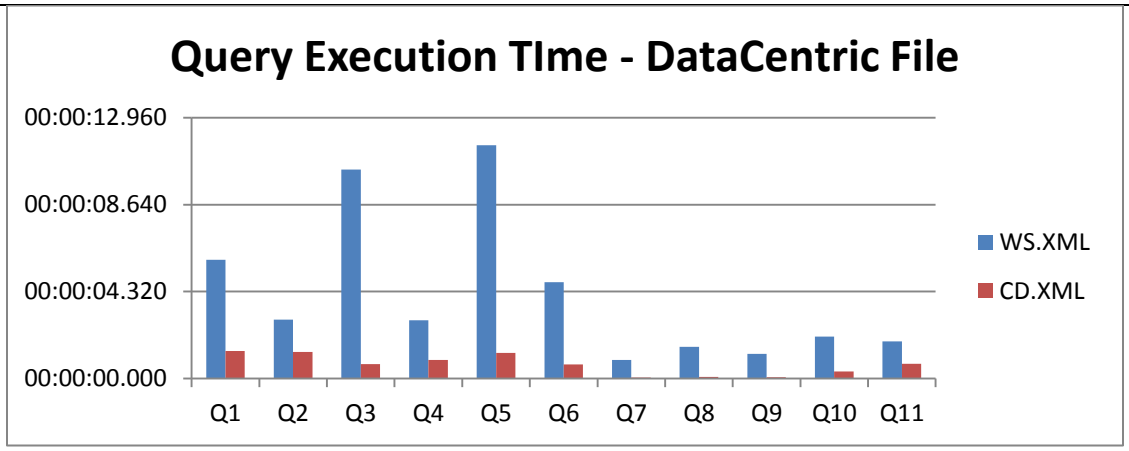
- **Study of Index Structure with the similar types of file :-**

To study this parameter the queries are classified into two types: - 'Axis' query and 'FLOWR' queries. The 'Axis' type of queries traverse a complete XML file in forward and backward direction. The containment relationship like P-C and A-D is also mapped on it. The 'FLOWR' queries are evaluated on predicate pattern on it. They are returning either a complete path of a tree or a branch of a tree i.e. 'twig' pattern query. The query execution time is mapped into following graph:-

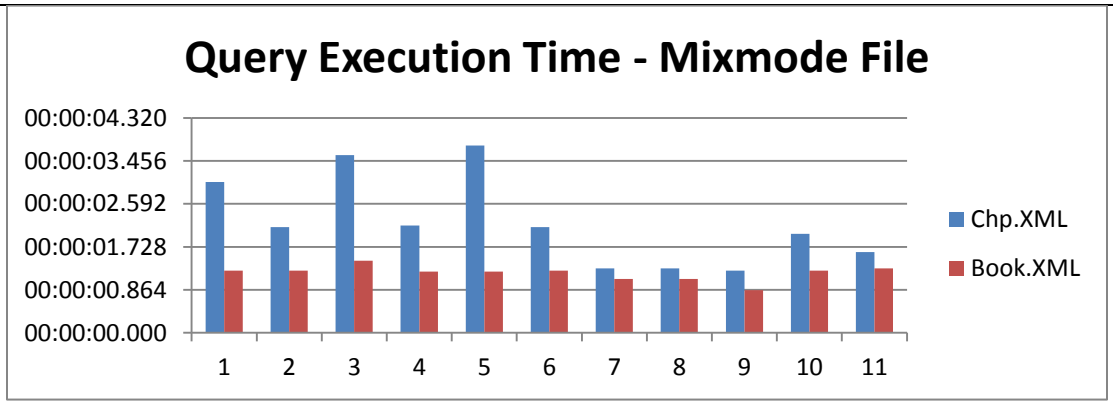
A. **Tree Traversal Query Comparison** :-

Graph 6.12: PESS- Index Structure – Axis Query

Query Execution Time – – Data Centric File



Query Execution Time "CD.XML" Vs "WS.XML"



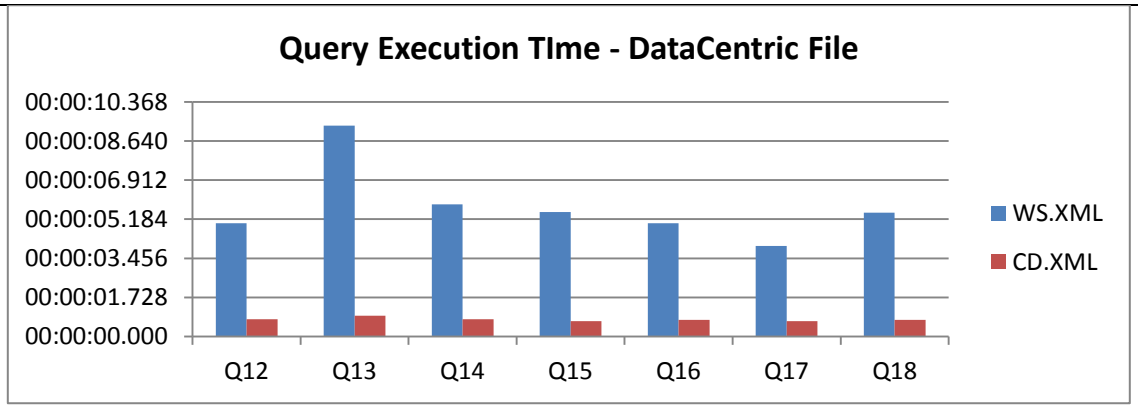
Query Execution Time "Book.XML" Vs "Chp..XML"

Forms the above graph 6.13, researchers observe that the pattern of query execution time is same. The Q5 and Q3 haves maximum time while Q9 has minimum time. The execution time is varying as the depth of the file is increased because the disk I/O cost is increased.

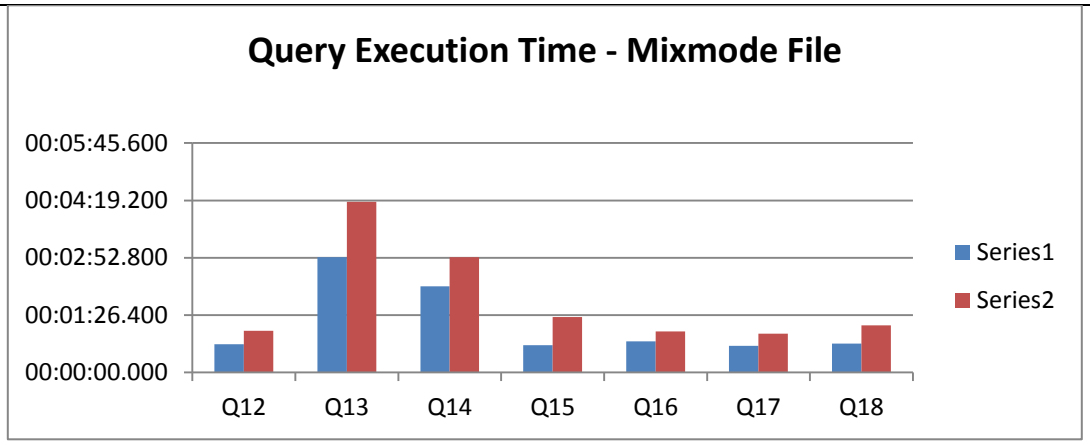
B. **FLOWR Query Comparison** :-

Graph 6.13 PESS- Index Structure - FLOWR Query

Query Execution Time – – Data Centric File



Query Execution Time "CD.XML" Vs "WS.XML"



Query Execution Time "Book.XML" Vs "Chp..XML"

Forms the above graph6.13, researchers observe that the pattern of query execution time is same. The Q13 has maximum time and Q18 has minimum time in execution. The query execution time is directly proportional to data node retrieval. Thus change in predicate value changes the number of nodes retrieved from the database. Hence the query execution time varies from query to query.

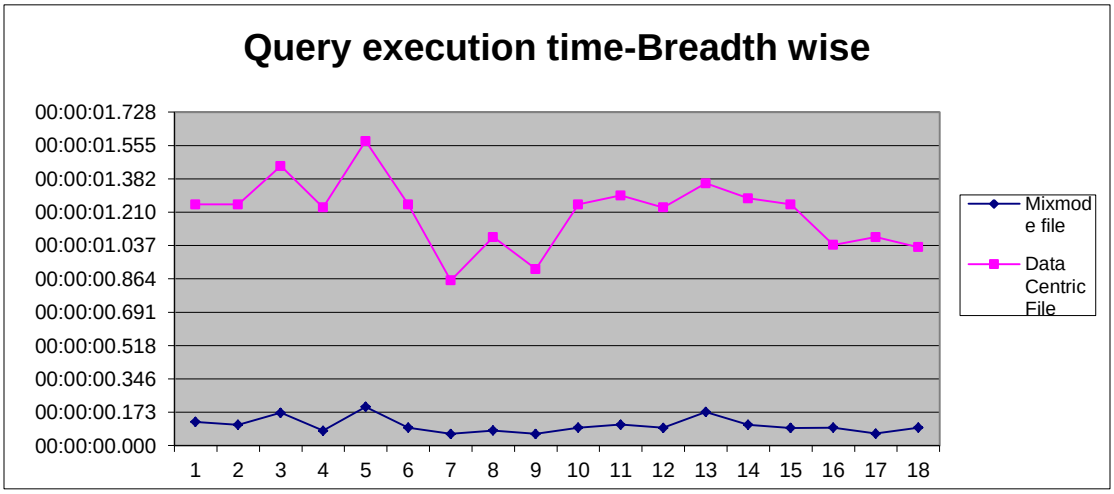
- **Study of Index Structure with the different types of file :-**

A. **Query Execution Time**:- For the study, the same index structure for the data centric and document centric file is considered . The following two graphs are plotted for the comparative study. Graph (1) is for a comparative study of the query execution time of breadth wise 'Mix-Mode' and 'Data Centric file', i.e the comparative study of (Cd.XML and book.XML file) the time are recorded into graph 6.14 while graph(2) is for a comparative study of query execution time of depth wise ' Mix_Mode' and data centric file . i.e WS.XML and Chp.xml file. The index structures of these files are same but the data size is different. The time is recorded into 6.15

Graph(1) : Breadth wise file distribution record:-

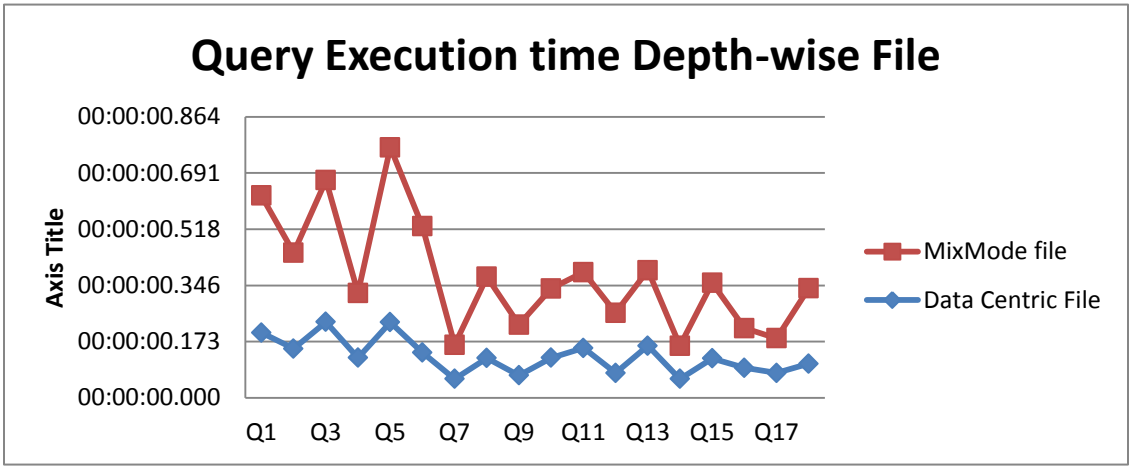
Graph 6.14: PESS- Comparative Study of Query Execution Time

(CD.XML Vs Book.XML)



Graph (2) :Depth-wise record distribution :-

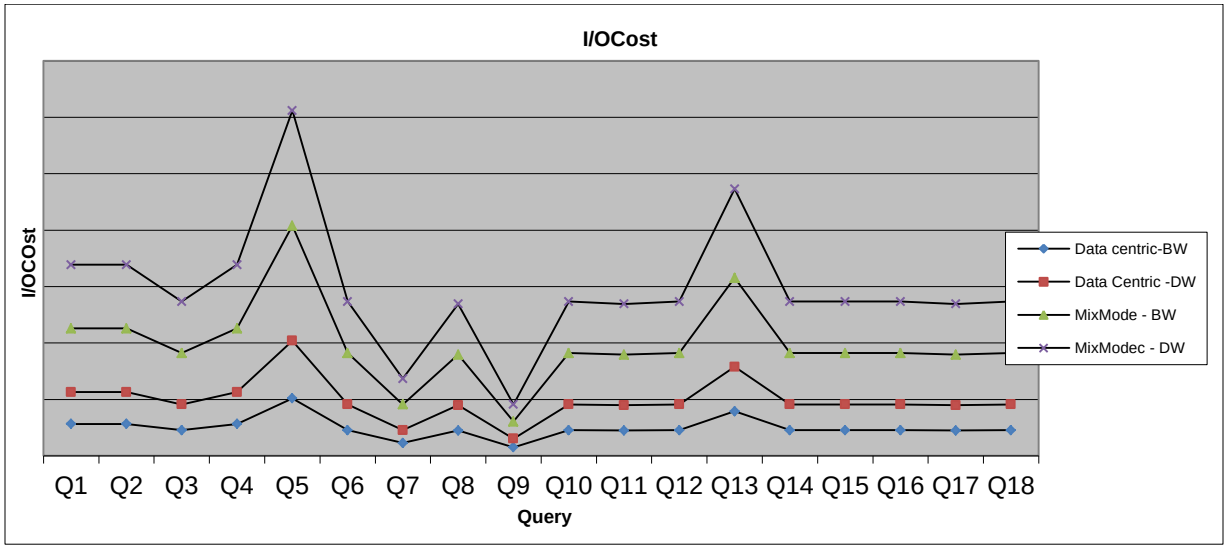
Graph 6.15: PESS- Comparative Study of Query Execution Time (WS.XML Vs Chp.XML)



From above graph 6.14 and graph 6.15, researchers observe that in axis type of query, minimum time for query execution is for query 9, and maximum time is for query 5 . Similarly for 'FLOWR' queries, minimum query execution time is for query 17, and maximum time is for query 13. Thus we conclude that the query execution pattern is similar even if the data size of the file is different but the index structure is the same. The query execution time varies as per the query work load.

6.6.4.3.4 Disk Traffic Measurement: - This is another parameter for measurement. It measures I/O cost. It is nothing but total number of pages fetched from the disk for query execution.

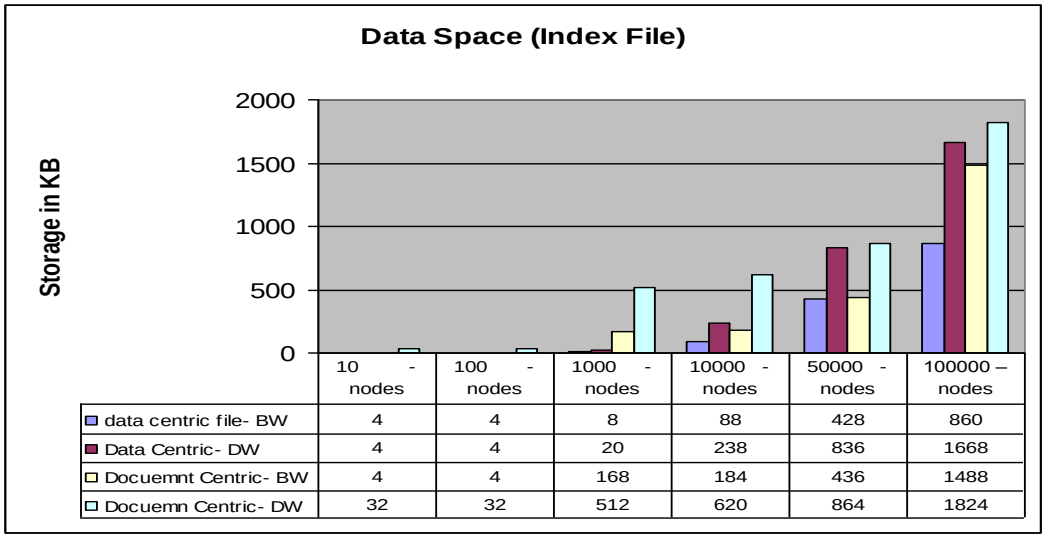
Graph 6.16: Comparative Study of Disk Traffic



From the above Graph 6.16, researcher observed that the disk traffic is proportional to the query evaluation time. A good index structure always fetches fewer pages from the disk. The graph proves that the PESS model satisfies the said properties.

Disk Space :- This parameter measures the amount of memory required for file storage. The memory size for storing a complete XML file is different. But the index file storage depends upon the number of distinct records in it.

Graph 6.17: Comparative Study of Disk Space



	10 - nodes	100 - nodes	1000 - nodes	10000 - nodes	50000 - nodes	100000 – nodes
data centric file- BW	4	4	8	88	428	860
Data Centric- DW	4	4	20	238	836	1668
Docuemnt Centric- BW	4	4	168	184	436	1488
Docuemn Centric- DW	32	32	512	620	864	1824

From the above Graph 6.17, the disk space is high for depth wise file (either data centric or document centric file). The size of index file is high for the file whose depth is high because it stores the distinct index entries. The disk space for breadth wise files is low.

6.6.5 Result and Analysis of Criteria II: Measurement of new label insertion cost and Index creation cost.

Requirement for Experiment Set II:-

In query optimization study query execution is one of the important parameter; another parameter of measurement is index cost. The second set of experiment study it. The study os classify into three aspect node/Label updating cost, index maintenance cost.

For the study the data sets selected whose depth and width increase gradually. The file structures along with no of children are explained into 6.12. The three datacenter files are selected for experiment. The firs file contains 10 records, second file contain 100 records, third file contain 1000 records. Initial the files have depth three. Initially all node_ids are added to each node. The files are increased gradually; means for every child at level 3, new four children are added. Now, the depth of the files is increased it becomes '4'. Next level i.e on 5th level additional two children are added to each node. Next level again inserts 3 children and next level i.e. at 7th level inserts another 2 children. These new node insertions affect the indexing. This experiment measure indexing updating cost. The File structure is explained below:-

Table 6.12: Input File for Node Updating

Input : File Structure					
Data File		**File Structure**		**No of records**	
		Depth of the File	Breadth of the file		
Data-Centric File -Depth wise and Breadth wise					
Data-centric file	Set 3.1.1	3	10	10	Each node of XML tree has six children and Total number of nodes are 10
		4	30	10	new 3 children are added per node
		5	60	10	new 2 children are added per node
		6	180	10	new 3 children are added per node
		7	360	10	new 2 children are added per node
Data-centric file	Set 3.1.2	3	100	100	Each node of XML tree has six children and Totla number of nodes are 100
		4	300	100	new 3 children are added per node
		5	600	100	new 2 children are added per node
		6	1800	100	new 3 children are added per node
		7	3600	100	new 2 children are added per node
Data-centric file	Set 3.1.3	3	1000	1000	Each node of XML tree has six children and Totla number of nodes are 1000
		4	3000	1000	new 3 children are added per node
		5	6000	1000	new 2 children are added per node
		6	18000	1000	new 3 children are added per node
		7	36000	1000	new 2 children are added per node

This data set is used for testing

(1) XML Node_id Label Updating Cost

(2) Index Maintenance Cost

(3) Index Updating Cost.

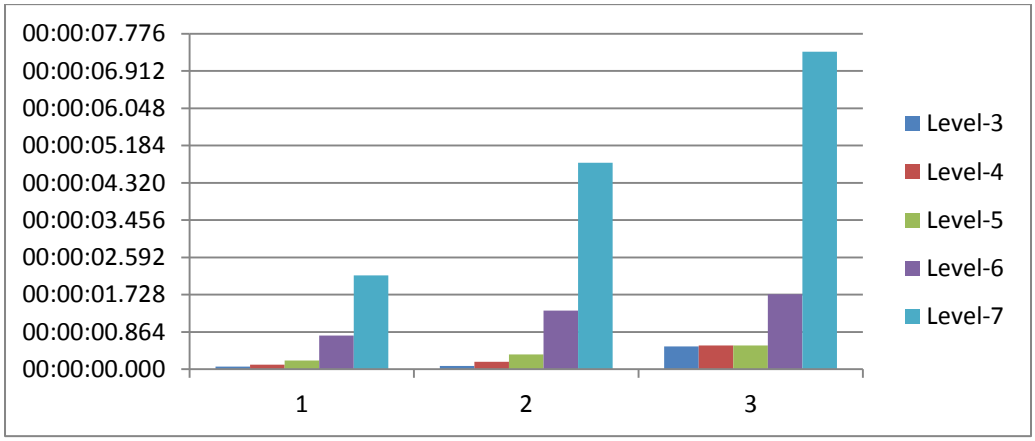
The experiment is conducted on --- Intel Pentium (R) Dual Core CPU with 2.69GHz CPU, 2.79 GB RAM and 40 GB hard disk. The operating system is Windows 7 and java (JDK1.6). The database product IBM – DB2 .9 is used for XML file storage. The node updating algorithm is implemented in Java.

Analysis of Label Creation and Updating Cost :-

This experiment is performed to check new label node insertion at leaf node, in between node, end of the tree. It also checks that reutilization of the deleted node_id when the new node is added into XML file.

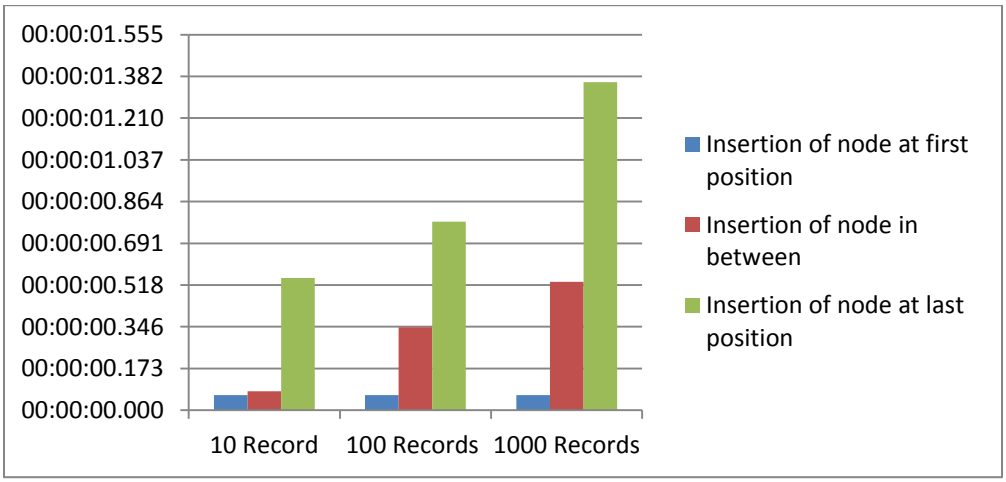
(1) **Insertion at Leaf node**: - The new node_id are inserted at different places in the XML tree. Time is measure for recaluculation of new node value. Initially the file has depth thee and number of record ten. . The process is continuing till seven levels . The insertion time is recorded in Graph 6.18.

Graph 6.18: Time Measurement for Label Insertion (leaf node)



a. **Inserting new node in between:** - The new node is inserted in between at different position of the XML file. For the experiment three cases are considered; insertion is at beginning of XML node, in between of XML node and at the end of XML node. Following graph 6.19 map the time required for inserting new node in to the tree.

Graph 6.19: Time Measurement for Label Insertion (in between)



6.7 Contribution of Researcher:-

XML data is stored into three formats. From the experiment results XML-RDBMS storage is best strategy for the storage. The researcher designed a new model "PESS" for it. This model tries to overcome the drawback of XML-RDBMS storage which was found in literature review. Major contribution of the researchers is:-

- Each XML node is uniquely identified by assigning a unique number. This number maintains structural relationship between different nodes. The node_id identify P-C, A-D as well as sibling relationship.
- The node_id structure is dynamic. It reduced the updating cost. The node_id structure is flexible and easily creates a new node_id for newly inserted node. This new record is easily updated into relational table.
- The 'Path summary' is created for all probable paths in XML file. This path summary information reduced query execution time.
- The 'Path Summary' information also useful for execution of partial matching query.
- The XML paths are always starting from root node. Hence lengths of the paths are increases as depth of the file is increased. The prefix paths are common and there is change into suffix path of path. Instead of storing the path from root till leaf node, PESS model store these path reverse way. This storage reduced traversal cost while traversing XML file.

- For storing XML file into relational schema only three tables are used for it. Therefore in query execution processes number of join operation is reduced. In query execution most probably it refers only two tables.
- The “equi-join” operations can successfully executed by PESS model with help of “Path summary” information.

References:-

1 A. A. Abd El-Aziz , A. Kannan "Survey of Mapping XML DTDs (Documents) to Relational Schemas"International Journal of Computer Science and Management Research Vol 2 Issue 1 January 2013 ISSN 2278-733X

2 A. Gabillon. An Authorization Model for XML Databases. In Proceeding of the 2004 workshop n secure web service, Fairfax, Virginia, USA, pages 16-18, 2004

3 C Zhang, J Naughton, D DeWitt, Q Luo G Lohman " On Support Containment Queries in Relational Database Management Systems" in Proc. Of ACM SIGMOD , pp 425-436, 2001, pp 425-36

4 C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. 20th Management of Data (SIGMOD '01), 2001ACM SIGMOD Int'l Conf.

5 D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDMBS," IEEE Data Eng. Bull., vol. 22, pp. 27-34, 1999.

6 F. Dietz, "Maintaining Order in a linked List" , in Proceedings of the ACM Symposium of Theory of Computing, 1982, pp 122-7 (1982)

7 Gang Gou and Rada Chirkova "Efficiently Querying Large XML Data Repositories: A Survey" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 19, NO. 10, OCTOBER 2007

8 J. Shanmugasundaram, E.J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J.F. Naughton, and I. Tatarinov, "A General Technique for Querying XML Documents System," SIGMOD Record, vol. 30, pp. 20-26, 2001.Using a Relational Database

9 John O'Donahue " Java Database Programming Bible" ISBN:0764549243 John Wiley & Sons © 2002

10 M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," vol. 1, pp. 110-141, 2001ACM Trans. Internet Technology,

11 Matthias Nicola , Bert van der Linden "Native XML Support in DB2 Univeral Database " Proceedings of the 31st VLDB Conference , Trondheim , Norway,2005

12 Mehdi Emadi, Masoud Rahgozar, Adel Ardalan, Alireza Kazerani, and Mohammad Mahdi Ariyan "Approaches and Schemes for Storing DTDIndependent XML Data in Relational Databases" World Academy of Science, Engineering and Technology 19 2006

13 Q.Li and B. Moon , " Indexing and Querying XML Data for Regular Path Expressions", in Proceedings of the VLDB, 2001 ,pp 361-70 (2001)

14 R. Krishnamurthy, R. Kaushik, and J.F. Naughton, "XML-SQL Query Translation Literature: The State of the Art and Open Problems," Proc. First Int'l XML Database Symp. (XSym '03), 2003.

15 S. Abiteboul, P. Buneman, and D. Suci. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, ISBN: 1-55860-622-X, 2000

16 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, "Indexing XML Data Stored in a Relational Database," Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04), 2004.

17 T. Yu, D. Srivastava, L. Lakshmanan, and H. Jagadish. A Compressed Accessibility Map for XML. ACM Transactions on Database Systems (TODS) Volume 29 – Issue 2, pages 363-402, 2004

18 V W.E. Kimber , "HyTime and SGML: Understanding the HyTime HYQ Query Language",http://ftp2.de.freebsd.org/pub/sgml/ifi.uio.no/HyTime/HyQ-1.1.Kimber
v Online Computer Library Center, 'Introduction to the Dewey Decimal Classification", http://www.oclc.org/dewey/

19 V E. Cohen, H Kaplan and T. Milo, "Labeling Dynamic XML Trees", in Proceedings of the ACM SIGMOD-SIGACT-SIGART, 2002, pp. 272-81

20 V P. O'Neil, E. O'Neils , S. Pal , L . Cseri, G Schaller and N. Westbury, "ORDPATHS:Insert-Friendly XML Nodes Labels", in Proceedings of the AMC SIGMOD, 2004, pp 903-8

21 W3C Website. Extensible Markup Language (XML) 1.0 (Fourth Edition). Online:http://www.w3.org/TR/REC-xml/, Accessed on: 30/10/2006

22 W3C Website. XQuery 1.0: An XML Query Language. Online: http://www.w3.org/TR/xquery/,Access on: 30/01/2007

23 Xiaodong Wu et "A Prime Number Labeling Scheme for Dynamic Order XML Trees" in 20th International Conference on Data Engineering (ICDE04) 2004 IEEE pp 66-78

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Today in the E-world era, most of the businesses like banking, trading etc required web applications to run their businesses. To run it successfully, the heterogeneous systems are used to perform a single transaction. These systems exchange the data from one system to another system, and so there is a need of a language which is portable, user friendly, and flexible. These needs are satisfied by XML. It is a textual language for representing and exchanging information over the web. It is mainly used as a standard for electronic data interchange and also as a flexible human-readable data storage format.

In 1969, IBM developed first markup language called as GML (General Markup language) which was specially used for document representation and information exchange between the document. But it had major problem of "validating parser", means once the document has been read, it automatically performed document type definition and check accuracy of markup, without going to the expenses of actually preceding a document. The problem was resolved in 1978 -1986 by SGML i.e. Standard General Markup language). It had become defector standard for interchange information of large complex document and has been used in domains of military applications and airplane maintains. It is complex, document interpretation is hard and hence user friendly language is required. In 1989, HTML was proposed by researchers team Tim Berners-Lee. It is user-friendly language. It inherits some features from SGML. But it suffers the limitation of standardization. Some of the tags displayed the different effect for different browser. E.g. 'Blink' elements cannot be rendered on some computer. Another important limitation, it has fixed set of markup tags which was unable to adopt the new further information of the application. And in 1990s, the web became more and more optimistic for single deliver platform, hence needed a solution for this standardization. The solution was provided by CSS or by XML. CSS allowed people to attach formatting to HTML document with property. XML is markup as a common representation. It has extensibility for document type and verification that document follow rules. Since 1998 it becomes an important component for web application. It

is logically formatted documents which reduce the difference between pure text data, and explicitly structured information traditionally stored in relational databases. Thus all above features recommended that XML is effective manager for the web application. Though, it manages the application but still it required to take the help of other softwares like RDBMS as well as HTML.

As importance of XML increased, so this thesis was studding the importance of XML in e-business. The study was performed into two parts: "XML needs into business domain" and "XML storage management and query optimization ". In the first part, the study was focused on the XML utilization in company. The content analysis technique and case study method were used for data collection. The data was collected from different business domain. For study twelve different business domains were selected. The business domain list was listed into table on page no: 100. The case study method was also followed for collection of business information. The data was also collected from software users as well as software developers for considering their opinion about the XML utilization in business.

The study concluded that XML is an important component in web development. It is used as database management tools, content management, and utilized for data transformation for different resources like different operating systems, different database. It also represents meta-data information like product manual, workflow procedure and business framework. The detailed analysis is presented into chapter 5 on page no:108. The XML-based business framework protocols were listed on page no: 126. These protocols were set by standard organization for easy communication. By using these protocols business interoperability become easy, it removes the participation of third party in business domain so that small scales company can perform the business easily on internet. E.g. EDI is Electronic data interchange community, which was used by the large company for selling and buying their product. This service was not affordable by the small company due to high paying cost and as large company has major stock holder of it hence they have their say for changing their standards which also affect small company business. These standards are replaced by XML. XML based e-commerce is penetrating down even to small companies. Most of the people agree that XML is easy to work with more extensible. Another example is XML utilization in Enterprise Application Integration.

This integration application is within the company itself means accounting system is integrated with inventory system, inventory is associated with billing system. XML is generated by large-scale enterprise accounting- ERP, CRM, SCM and related package. Every system communicates each other through XML only. This communication can be made in house or by third party. When vocabulary of one cannot match up, it is transferred from one vocabulary into another one. Web services allow different applications from different sources to communicate with each other through XML. It is not tied to any operating system or any programming language. i.e. 'JAVA' can talk to 'PREL'. Windows applications can talk with UNIX applications.

The protocol list suggests almost all business domains like E-commerce till food industry, agricultural industry use XML in their business.

All these observations suggest XML is an important component in today's e-world or e-business. Therefore, researchers study the storage strategy of XML. Broadly, XML data is stored into three types:-

1. The simple file system with offset based index is used for storing XML data. This method is also called navigation method. It is simple and straightforward. The XML file is read by using DOM. It stores complete document into memory which is very costly in terms of memory and query execution time. While executing query, the queried documents have to be read and parsed every time.
2. The second technique is the RDBMS: - The XML file is properly split into different tables and stored into existing relational database system. Once the data is converted into relational table, then existing database techniques can be used all advance feature of relational database like query optimization, indexing, transaction management security etc can used easily. XML-RDBMS storage is classified into: Schema less storage and Schema centric storage. In schema less storage, the XML schema is not known hence the designer need to define their own ways to represents the XML information. The designing of RDBMS schema is a major challenge because XML is recursive and hierarchical storage, while RDBMS represents relationship between different tables. Hence mapping XML characteristics into the relational table is difficult. XML is extensible, user can add their tags as per their need while in RDBMS, and once the schema is defined, it is difficult to modify the content on it. Designing of dynamic RDBMS schema is difficult job.

3. In Schema centric storage, XML schema is already known in advance. XML data is mapped into relational table. This mapping is a difficult task. Because XML features like recursive and hierarchical data represent is difficult in representation.
4. 'OODM' is one way to represent the XML data information In the OODBMS the XML node is considered as an object and the element and attributes of the XML file are considered as attributes of the object.
5. Once the data is stored into RDBMS format, then it takes the advantage of existing RDBMS components. But the main challenge of this storage is designing a proper database schema for it. This approach works well for simple query execution, but for the evaluation of complex query, it requires the multiple join operations which increase query processing cost. The other overhead of this approach is query translation. Each XML query needs to be transferred into SQL query first, and then the query execution starts.
6. The third approach is native storage which is used to store the XML file in new data format. A specialized data structure is used for storing the XML file, but all operations such as query optimization, transaction management etc have to be built from scratch.

The detailed analytical study is in chapter 3, all advantages and disadvantages of this storage. The discussion of query processing techniques is as follows:-

Query Processing techniques for this approach:-

In query processing technique, 'indexing' and 'table scan' operators are two important operators. In 'table scan' operator the query refers entire table for retrieving desired content. In order to process XML data independently, the effective indexing technique is required for retrieving and managing the XML data. The XML indexes are built on the different property of the XML file. Mainly they are built on the structural properties of the XML nodes, means on the node relationships or path values, or on the value of the XML nodes. And these indexes are classified into three types- (1) Summary index, (2) Structural join index, and (3) Sequence based index

The Summary indexes are created on the summary information of attributes and elements of associated XML document path, element set, and the paths emerging from these elements. The Structural index documents are created on structural relationship, i.e. A-D and P-C relationships. Then the indexes are built on these relationships. These types of indexes are more

preferable because they are used for solving DB and IR style of queries. In 'sequence based' index, the XML document and queries are organized in some predefined sequence and the indexes are built on it. These indexes are stored either on disk or in memory. The I/O cost is high for the disk based indexes. Therefore, the query execution time is also high. The memory based indexes resided in the computer's memory. Therefore, they gain a faster query evaluation by reducing the expensive I/O cost. These indexes are temporary indexes and small in size. The significant compression technique is required for storing these indexes. But the major drawback of these indexes is a lack of scalability for large XML database. Therefore, a good index structure is required for maintaining the XML structure which is small in size so that XML queries can be evaluated faster.

The researchers performed experiment on different XML storage property for identification of the best storage strategy for XML storage and indexing technique. The experiment was performed by using IBM DB2 and SAXON software. IBM DB2 stored XML file into native format and XML-RBMS format. SAXON stored XML file into navigational format. The experimental setup and configuration parameters are explained in chapter 4. The experimental results are noted in chapter 5. The query execution algorithm is on page no: 190. The disk space measurement and scalability measurement and indexing parameters are listed on page no: 97. In IBMS db2 two types of indexed are defined 'Path Index'. SAXON is not database products hence it cannot afford the luxury of Indexing. Tiny tree is used for XML data representation, nineteen different arrays are used for representation of XML data inside and 'Push and Pop' technique is used for query execution.

The query execution plans were measured and it suggests that the XML-RDBMs storage is best as compared with other two storage methods. The query execution time is very less than other types of storage. The query execution plan suggests the query execution parameters are similar with RDBMS parameter. It used table scan operation when entire table is needed for the query. In Index operation 'XSCAN' (XML Document Scan), XISCAN (XML Index Scan) and 'XANDOR' (XML Index Ending) are selected for execution of query when desired. Selectivity and I/O cost, cardinality values are used for query measurement. Researchers also observed that as depth of the file is increased, the query execution time is also increased. Hence the query

execution time depends upon the depth of the XML file not on the XML file types. The query execution time depends upon the structure of XML file.

Therefore the proposed model is based on XML-RDBMS storage. This model is tried to reduce the XML-RDBMS drawback and it has following features (1) Each XML node is uniquely identified (2) Numbers of join operations are reduced in query processing. (3) The XML schema is designing in such a way that it can adopt the dynamic updating of new records of XML file. The Model architecture is described on page: 170. The abstraction of the model is described as follows:-

7.2 PESS Model:-

The PESS means Path Encoding Structure Summary. The model is constructed on principles of : - (1) Assigning unique number to each XML node and (2) Nodes on similar paths are collected together.

In PESS model, the XML file is read, and unique number is assigned to each XML node. These numbers identify the relationships among the nodes. The node number is a combination of 'parent node+ self identifier + level number'. These node numbers interpret the P-C, A-D and sibling relationship. The algorithm for node_id creation is explained on page no:178. The primary index is created on node_id and experiment is performed. These node_ids are easily adding the records into relational table. The reconstruction of XML-RDBMS schema is not required. The algorithm of dynamic updating cost is explained on page no:180. Another drawback of XML-RDBMS is it cannot solve "equijoin" queries. To address this problem, PESS model designed 'Path encoding' techniques. It is nothing but similar paths are identified and assigning a number to it. So the different nodes on same paths are collected together. The algorithm of path summary is explained on page no. Once the node_id and path_id are created then XML data is stored into three tables:-

- P-table(Pah_id, Node_id, type of node)
- D-table(Node_id, value);
- N-table(Node_id, c_id)

- P -table (Pah_id, Node_id, type of node):- This table stores path information. For each path, all equivalent nodes are stored. The third column takes the value either 'E'- for element data information or 'A' – attribute node. Index is on Path_id value.
- D-table (Node_id, value):- This table stores the information of the value nodes. The table is composed of two columns 'Node_id' which stores unique node_ids of the XML file node. The 'value' column stores the node value information. Index is on Node_id value.
- N-table (Node_id, c-id) If the nodes are element nodes, their information is stored into this table. The column 'node_id" store the node identification and 'c-id' stores its children node_id. Index on Node_id.

Indexes are created on noe_id and path_id. The experimental results are listed on chapter 7. The summary of these experiments is as follows:-

- The minimum query execution time is required for 'self axis' query and the maximum time is required for 'descendent' and 'following-sibling' types of query.
- The query execution time is dependent upon query workload and I/O cost. As query workload increases, the query execution time also increases. Similarly as I/O cost increases, the query execution cost also increases.
- Impact of index structure on different types of XML file :-

For the experiment, the file structure for data centric and mix mode file was the same i.e. both the files had the same number of distinct values i.e. The total number of records in the index file is the same. For set1 (– BW file)—the depth of document centric and data centric file was four and the path structure was same. Similarly for data set 2(Depth wise File) the depth of both files was ten and path structure was similar. The data sizes of the files were different. The experiments were performed and observed that the query execution pattern was similar for both the files, but there was change in query execution time, it was increased. It increased as per the query workload i.e. the time increased as the file size increased.

- The PESS index stores only the equivalent path, so the index size is small. Therefore, the PESS model creates the memory residential index. As mentioned in the literature (chapter3) memory residential index always provides a good performance time for query execution.
- The PESS model reduces the structural join operation :-

A unique number is assigned to each XML node of the file. This unique number maintains P-C, A-D and sibling relationship. When the query retrieves the node_id, it automatically determines

the relationship between nodes, hence it does not require a separate merge join algorithm for retrieving the desired nodes.

As mentioned in chapter1 page no 13, redundant traversal of XML file in query execution increased query execution time. This is also reduced by PESS model. The prefix information of most of the queries is almost the same. i..e. the path structure of the query always starts from the root node. For each query needs to traverse the same path every time. This traversal increases query execution time. The PESS model reduces this redundant traversal by maintaining node_id and path_id creation

The data dictionary of path_id and nde_ids are collected and stored it. The query paths are match with path id and respective nodes are observed. The node_ids are interpreted as per the desired requirement.

- The information maintained in the PESS model is scalable because different files are used for storing the information. As two different tables are used for storing elements and value node information, the database files are manageable.

7.3 Limitation of PESS model

The PESS model tries to solve most of the issues mentioned in chapter1, but we have to still work on:

- Label length: - The label length, i.e. node_id, is long. It is directly proportional to depth of the tree. As the depth of the file increases, the label length also increases. For reducing the label length, a relevant compression algorithm is required.
- The node_id and path_id computational and interpretation cost is high.
- The structural join operation is reduced in containment type of query but structural join operation is need for order queries.
- Additional processing cost is required for manipulation of structural relationship.
- PESS model doesn't support 'orthogonal' property. The property suggests; once the data is converted into relational table, then it can be converted back to XML files.

7.4 Scope for future Research:-

Considering the importance of emerging industry or business like e-shopping megamall or hyper mall culture as well as small scale business, it will be necessary to understand in-depth analysis research related to cost benefit analysis for the perspective of effective customer service by focusing on XML optimization. The following points have been noted for future work:-

- In PESS model, the XML nodes are uniquely identified by node_id. These node_ids are combination of 'parent no + self identifier +level no". As depth of the file is increased, the node_id length is also increased. As length of the depth of the file is increased, the nodes label length are also increased. Hence compression algorithm is required for manipulation of node label.
- As per the literature review, it is always good to store the data into RDBMS format and design new algorithm for query execution. PESS model is implemented into relational table and existing query optimization algorithm is used for query execution. Thus designing of query optimization algorithm for PESS model is a future scope.
- In XML-RDBMS approach, mapping XML-RDBMS data, there is probability that the XML data may lose during the transformation hence the new algorithm is required for manipulating these losses of information.
- In PESS model, dynamic updation cost is considered. The new node can easily be added init but there is no provision for reusability of deleted node. So if continuous deletions take place there is probability of node_id length may be increased. Therefore the updating cost algorithm need to modify for this change.
- In XML-RDBMS schema 'orthogonality" is one of the important properties. The property suggests designing the XML-RDBMS schema such that once the data is transferred into relational table, then it can easily transfer it back to original XML file. But once the data is transferred into relational table, then it is difficult to convert it back to XML file. Hence system needs to maintain original XML file also. Therefore system storage overhead cost is increased so there is a need of developing the RDBMS schema such that can convert relational table into XML file.
- XML is recursive and hierarchical data; these characteristics are very difficult to map into relational table. To map this structural relationship into RDBMS, it always requires

additional field. This field increase calculation overhead and space overhead. Thus in future work proper designing of XML-RDBMS schema is required which will reduce calculation and space overhead.

- XML schema or DTD is defining XML document structure. Mapping DTD structural into RDBMS is difficult because XML is extensible. The new node information can be added into DTD very easily but it is very difficult to map this change into relational tables. Hence dynamic mapping algorithm from DTD to XML-RDBMS schema is a future invention.
- Once data is transferred into XML format, then some of the proposals are unable to execute equijoin operation. For execution of 'join" query, the query execution time is high; because it needs to perform first to identify structural relationship between query nodes and then need to execute query which required high query execution time. Thus query optimization algorithm needs to developed for the same.
- XML-RDBMS query processing cost is high because query needs to be converted into SQL language first, and then it is processed further. This conversion increased the cost. Thus in future research, find out the mechanism for reduction of these cost.
- XML is simple file, there is no mechanism for access control and security measurement for XML. Designing of the algorithm for these mechanism is one of the new dimensions of future research.
- For unique identification of XML node, node _ids are created. These node_ids are classified into two types: range label schema and pre-labeling the node_id. These node ids had following issues for further research.
- Major problems with Range label schema**:-**
    - Easy to understand structural relationship. Hence they can be easily solve P-C queries and content based query but it takes more time in execution of A-D relationship. The designing of new algorithm for query execution is future research scope.
- These node_id structures can effectively solve "Order sensitive queries" but it is difficult to solve descendent axis, following –sibling, preceding-sibling queries. Branch –path queries are difficult for evolution. Thus designing of new algorithm for order query execution is future research.

- In literature, the node_ids are created for identification of XML node. This node labeling schema is static means initially these new nodes are assigned to XML node. But when new XML nodes get added into XML file the entire node_id need to reconstruct again. So the updating cost is very high. This dynamic updating problem is handled by many research proposals, but still some problems are open for future research.
- Major problems with pre-labeling schema:-
  - Label computation is required for evaluating P-C, A-D and sibling relationship. This increases query processing cost.
  - Computation cost is high for determining structural relationship.
  - These node labelling schema have difficulty to solve order sensitive queries
- In Native storage, the intermediate query execution cost is high. These cost reduction is one of the major issues for future research.
- In Structural Summary index, the XML file is summarized into a specialized data structure. This structure serves as index for query evaluation but sometime this structure is not efficient for answering queries, the query has to travel a complete XML tree. Avoiding this excessive traversal is one of the future research areas.
- Lack of Scalability: - All the existing XML structural indexes are designed for answering one query at a time but they lack the scalability for answering multiple queries simultaneously. This increases query processing cost for server side and deteriorate the query performance
- XML data is also stored as document centric data, hence keyword search algorithm and proximate supportive algorithms, keyword ranking algorithm are need to support the query execution.
- XML indexes are classified into three types: structural summary index, selective index and structural index.
  - In structural summary index, summary of XML file is created and different types of indexes are crated on it. The size of index is very high so this index is difficult to store into memory. Thus this index size reduction is future scope of XML index.
  - In summary index , it is always preferred to search the query XML summary ,but for some query execution it needs to travel complete tree thus indexing benefits are not achieved

- In selectivity index implement indexing technique from RDBMS literature. As per RDBMS literature the DBA decide the index structure, it may be decided on the basis of frequently asked query with fixed structural information. But XML has property of extensibility so the structure can be extending as per the user need. XML structure is dynamic; hence defining index structure is difficult on the above criteria.
- XML index updating cost is high hence defining index of it is difficult.

- XPATH and XQUERY are two popular languages. Lots of work on query optimization on XPATH query is performed but still XQUERY query optimization is under exploration.

7.5 Suggestion:-

The thesis concludes that XML is one of the important components in software. It is used for representation of business data in international standards. So the internet communication becomes very easy and effective between inter and intra organization. XML is used not only representation of business data information but also used for information storage e.g. workflow of company procedure. It works as metadata representation and also used for web content immanent system.

All these requirements suggest XML data must be stored in proper format so it can effectively be retrieved by the user as per their need. Three models are used for storing XML information. The experiment is performed on XML storage; it suggests XML-RDBMs storage is the best option for XML data store. Once the data gets converted into relational tables, then existing RDBMS mature database management system can be used for executing all operation on it.

XML-RDBMS schema must be dynamic i.e. XML is extensible, new nodes are added as per the user need. The XML-RDBMS schema designing is such that it can adopt these dynamic changes. Another property of schema is 'orthogonality' means once XML file is converted into RDBMS tables, then it is also possible to convert RDBMS to XML file. XML represents data recursively means the same node-name are occurred in multiple times in the XML file. E.g XML has 'name' tag for the book, and 'name' tag is also used for naming publication of the book. This recursive procedure may represent in RDBMS schema. Each record in the tables must identify the relationship between two nodes. Different methods are used for maintaining these relationships. Best of these is assigning unique identification to XML node. Designing of numbering schema must satisfy the following problems:-

- Reconstruction problem: - With a given label of a node parts of the tree structure of the database can be reconstructed.
- Decision problem: - Node_id determine relations between two nodes, additional calculation is not required for determine the relationship between them.
- Numbering schema is dynamic, means when new node is added into XML file; reconstruction of entire labeling is not needed.
- Relationships between two nodes can be uniquely and quickly determined simply by examining their labels.
- The size of labels should be minimal in order to fit in memory
- The schema can be used to support all kinds of queries

Once XML file converted into relational table, then standardized indexing techniques is used for indexing relational tables. The index can be 'B+ index', 'Hash Index' etc. Hash index is best for answering contain query and B+ tree index is best for answering containment and tree traversal queries

BIBLIOGRPHY

Books

1 Abraham Silberschatz, Henry Korth, S. Sudarshan "Database System Concept" McGraw-Hill Education 27-Jan-2010 ISBM 978-007-128959-7

2 Benoît Marchal " XML By Example" International Standard Book Number: 0-7897-2242-9

3 Charles F Goldfarb, Paul Prescod The XML Handbook , , Third Edition , Pearson Education Asia

4 Elliotte Rusty Harold " XML™ Bible " ISBN: 0-7645-3236-7

5 Erik T. Ray Learning XML Publication O'REILLY

6 Hector Garcia-Molina , Jennifer Widom " Database Implementation"

7 James Keogh " J2EE: the complete reference" McGraw-Hill/Osborne, 06-Sep-2002 -

8 John O'Donahue " Java Database Programming Bible" ISBN:0764549243 John Wiley & Sons © 2002

9 Whitmer R., DOM Working Group : Document Object Model (DOM) Level 3 XPATH Specification [2003]

Journals and Periodical

1 Altinel, M., Aksoy, D., Baby, T., Franklin, M.J., Shapiro, W., And Zdonik, S.B. 1999. Dbis-Toolkit: Adaptable Middleware For Large Scale Data Delivery. In Proc. Of Sigmod

2 An XML Indexing Structure with Relative Region Coordinate, In proc. of International Conference on Data Engineering (ICDE): 313-320.

3 Atul D. Raut, Dr. M. Atique Efficient Querying of Root to Leaf Xpath Queries for XML Documents International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 1, July 2012 ISSN: 2277-3754 ISO 9001:2008

4 B.M. Monjurul Alom, Frans Henskens, Michael Hannaford DRXQP: A Dynamic Relational XML Query Processor ; 2010 Seventh International Conference on Information Technology 978-0-7695-3984-3/10 $26.00 © 2010 IEEE

5 Barbara Catania and Anna Maddalena, Athena Vakali "XML Document Indexes: A Classification " SEPTEMBER • OCTOBER 2005 Published by the IEEE Computer Society 1089-7801/05/$20.00 © 2005 IEEE IEEE INTERNET COMPUTING

6 Bart Meltzer and Robert Glushko "XML and Electronic Commerce: Enabling the Network Economy", 1998

7 Beyer , K et al : System RX:One part Relational , One Part XML SIGMOD 2005 , June 14 -16 2005b, Baltimore , Maryland , USA Copyright 2005 ACM 1-59593-060-4/05/06

8 Brian F. Cooper ,Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, Moshe Shadmon A Fast Index for Semistructured Data Proceedings of the 27th VLDB conference, Roma, Italy, 2001

9 Burton Grad "Relational Database Management Systems: The Formative Years " IEEE Annals of the History of Computing Volume 34, Number 4, october–December 2012
pp. 7-8 | 10.1353/ahc.2012.0037

10 Byron Choi, Mary Fernandez, Jerome SimeonThe XQuery Formal Semantics: A Foundation for Implementation and Optimization May 31, 2002

11 C Zhang, J Naughton, D DeWitt, Q Luo G Lohman " On Support Containment Queries in Relational Database Management Systems" in Proc. Of ACM SIGMOD , pp 425-436, 2001, pp 425-36

12 C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. 20th ACM SIGMOD Int'l Conf.Management of Data (SIGMOD '01), 2001

13 Changqing Li Tok and Wang Ling "QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates" *CIKM'05,* October 31-November 4, 2005, Bremen, Germany. ACM 1-59593-140-6/05/0010

14 Chulho Ahn, Quing Li, Ramez Elmasri, Shalli Prabhakar, Niroj Manandhar, Do Youn Kim " A Survey of Three Types of XML Indexing Techniques" ACM Transactions on Computational Logic, Vol. 37, No. 4, 12 2005, Pages 1 -24.

15 Chun Zhang, Je_rey Naughton, David DeWitt, Qiong Luo , Guy Lohman On Supporting Containment Queries in Relational Database Management Systems , SIGMOD '01 Santa Barbara, California USA Copyright 2001 ACM 089791886/ 97/05 pp 425-36

16 D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML via XML Fragments. In the proceedings of the 26th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 151-158, 2003.

17 D. Chamberlin, J. Robie, D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Source. In WebDB (Informal Proceedings), pages 63-62. 2000

18 D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDMBS," IEEE Data Eng. Bull., vol. 22, pp. 27-34, 1999.

19 D.D. Chamberlin, "XQuery: An XML Query Language," IBM Systems J., vol. 41, no. 4, 2002.

20 E. Cohen, H Kaplan and T. Milo, "Labeling Dynamic XML Trees", in Proceedings of the ACM SIGMOD-SIGACT-SIGART, 2002, pp. 272-81

21 F. Peng and S.S. Chawathe, "XPath Queries on Streaming Data," Proc. 22nd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03), 2003

22 F. Tian, D.J. DeWitt, J. Chen, and C. Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies," SIGMOD Record, vol. 31, no. 1, pp. 5-10, 2002.

23 Gang Gou and Rada Chirkova "Efficiently Querying Large XML Data Repositories: A Survey" IEEE Transactions On Knowledge And Data Engineering, Vol. 19, No. 10, October 2007

24 H. Wang and X. Meng, "On the Sequencing of Tree Structures for XML Indexing," Proc. 21st IEEE Int'l Conf. Data Eng. (ICDE '05), 2005

25 Hye-Kyeong Ko and SangKeun Lee A Binary String Approach for Update in Dynamic Ordered XML Data IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 22, NO. 4, APRIL 2010

26 Ibrahim Dweib , Ayman Awadi and Joan Lu MAXDOR: Mapping XML Document into Relational Database The Open Information Systems Journal, 2009, 3, 108-122

27 Ioana Manolescu-Goujot INRIA Futurs - LRI - PCRI, France Gemo group XML query processing: storage and query model interplay

28 J. Chu-Carroll, J. Prager, K. Czuba, D. Ferrucci, and P. Duboue. Semantic Search via XML Fragments: A High-Precision Approach to IR. In the proceedings of the 29th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 445-452, 2006

29 J. Kamps, M. Marx, M. Rijke, and B. Sigurbjörnsson. Best-Match Querying from Document-Centric XML. In proceedings of the 7th International Workshop on the Web and Databases (with ACM SIGMOD/PODS 2004), Paris, France, pages 55-60, 2004.

30 J. Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In CIKM, pages 533–542, 2004.

31 J. Lu, T. W. Ling, C. Y. Chan, and T. Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In *Proceedings of VLDB*,pages 193–204, 2005.

32 J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. F. Naughton, and I. Tatarinov. A General Techniques for Querying XML Documents using a Relational Database System. SIGMOD Record, 30(3):20 26,2001.

33 J.-K. Min, C.-H. Lee, and C.-W. Chung, "XTRON: An XML data management system using relational database," Information and Software Technology, vol. 50, pp. 462-479, 2008.

34 Jonathan RobieXML Processing and Data Integration with Xquery Published by the IEEE Computer Society 089-7801/07 2007 IEEE

35 Jong P. Yoon ,Vijay Raghavan,Venu Chakilam "Bitmap Indexing-based Clustering and Retrieval of XML Documents " 2001

36 Jong P. Yoon, Vijay Raghavan, Venu Chakilam, Larry Kerschberg BitCube: A Three-Dimensional Bitmap Indexing for XML Documents Jong Yoon, et. al., BitCube, Journal of Intelligent Information Systems, Vol. 17, November 2001

37 Justin Zobel, Alistair Mo®at, and Kotagiri Ramamohanarao. Guidelines for Presentation and Comparison of Indexing Techniques. ACM SIGMOD Record, 25(3):10-15, 1996.

38 K. Beyer, R. J. Cochrane, V. Josifovski, J. Kleewein, G. Lapis, G. Lohman, B. Lyle, F. O¨ zcan, H. Pirahesh, N. Seemann, T. Truong, B. Van der Linden, B. Vickery, and C. Zhang. System RX: One part relational, one part XML n SIGMOD, 2005.

39 Kamil Toman "Storing And Indexing XML Data" ,2002

40 Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In CIKM, pages 533–542, 2004.

41 M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi. Efficient schema-based XML-to-Relational data mapping. Avaliable online at www.sciencedirect.com since 15/12/2006]20/1/2006

42 M. Mani and D. Lee. XML to Relational Conversion using Theory of Regular Tree Grammars. In Proceedings of the 28th VLDB Conference, Hong Kong, China, 2000.

43 M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, “XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases,” vol. 1, pp. 110-141, 2001.ACM Trans. Internet Technology,

44 Mag.a Katharina "Flexible and Selective Indexing in XML Databases" ,Johannes Kepler UniversitÄat Linz, Altenberger Stra¼e 69, 4040 Linz, Austria ,August 2008

45 Michael Kay "Ten Reason Why Saxon XQuery is Fast" Bulletine of the IEEE Computer Scociety Technical Committee on Data Engineering 2008

46 Mikael Fernandus Simalango "XML Query Processing and Query Languages: A Survey" Property of Amikelive.com – Technical Paper Series 25/10/2008

47 Mustafa Atay , Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi Efficient schema-based XML-to-Relational data mapping www.elsevier.com/locate/infosys

48 N. Bruno, N. Koudas, and D. Srivastava, “Holistic Twig Joins: Optimal XML Pattern Matching,” Proc. 21st ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’02), 2002

49 N. Zhang, V. Kacholia, and M.T. Ozsu, “A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML,” Proc. 20th IEEE Int’l Conf. Data Eng. (ICDE ’04), 2004

50 Ning Zhang M. Tamer O¨ zsu Ihab F. Ilyas Ashraf Aboulnaga FIX: Featurebased Indexing Technique for XML Documents VLDB ‘06, September 1215, 2006, Seoul, korea. ACM 1595933859/ 06/09.

51 Ntima Mabanza "Analyzing the Impact of XML Storage Models on the Performance of Native XML Database Systems – A Case Study" 978-0-7695-3984-3/10 © 2010 IEEE DOI 10.1109/ITNG.2010.207

52 P. O’Neil, E. O’Neils , S. Pal , L . Cseri, G Schaller and N. Westbury, “ORDPATHS:Insert-Friendly XML Nodes Labels”, in Proceedings of the AMC SIGMOD, 2004, pp 903-8

53 P.F. Dietz, “Maintaining Order in a linked List” , in Proceedings of the ACM Symposium of Theory of Computing, 1982, pp 122-7

54 Q.Li and B. Moon , “ Indexing and Querying XML Data for Regular Path Expressions”, in Proceedings of the VLDB, 2001 ,pp 361-70

55 R. Goldman, and J. Widom, “Data Guides : Enabling Query Formulation and Optimization in Semistructured Databases”, Proceedings of the VLDB, 1997, pp. 436-445.

56 S. Abiteboul et al. The Lorel Query Language for Semistructured Data. In International Journal on Digital Libraries, 1(1):68-88. 1997

57 S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

58 S. Sipan, K. Verma, J. Miller, and B. Aleman-Meza. Designing a high-performance database engine for the 'Db4XML' native XML database system. The Journal of Software-69, pages 87-104, 2004Systems and Software

59 Samini Subramaniam , Su-Cheng Haw , Poo Kuan Hoong Mapping and Labeling XML Data For Dynamic Update, Second International Conference on Computer Research and Development , 978-0-7695-4043-6/10 $26.00 © 2010 IEEE DOI 10.1109/ICCRD.2010.173

60 Songting Chen , HuaGang Li , Junichi Tatemura ,WangPin Hsiung , Divyakant Agrawal , K. Selc¸uk Candan Twig2Stack: Bottomup Processing of Generalized TreePattern Queries over XML Documents VLDB '06, September 1215, 2006, Seoul, Korea.

61 Steven A. Demurjian, David K. Hsiao, David K. Hsiao,Douglas S. Kerr , Robert C. Tekampe , and Robert J. Watson. "Performance measurement methodologies for systems. Proceedings of the 1985 ACM annual conference on database the range of computing: mid-80's perspective: mid-80's perspective, Denver, Colorado, United States, 1985, Pages: 16-28

62 Su Cheng Haw, and G. S. V. Radha Krishna Rao "Query Optimization Techniques for XML Databases " World Academy of Science, Engineering and Technology 22 2006

63 T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In SIGMOD, pages 455–466, 2005.

64 T. Fiebig, S. Helmer, C. Kanne, G. Moerkotte, J. Neumann, and R. Weld. Anatomy of a native XML base management system. VLDB Journal- Volume 11, Number 4, pages 292-314, 2002

65 T. Yu, D. Srivastava, L. Lakshmanan, and H. Jagadish. A Compressed Accessibility Map for XML. ACM Transactions on Database Systems (TODS) Volume 29 – Issue 2, pages 363-402, 2004

66 Venu Datla and Katerina Goˇseva–Popstojanova "Measurement–based Performance Analysis of E–commerce Applications with Web Services Components" Proceedings of Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05) 0-7695-2430-3/05 the 2005 IEEE International Conference on e-Business ngineering (ICEBE'05) 0-7695-2430-3/05

67 W. Lian, N. Mamoulist, W-l. David Cheung, and S.M. Yiu, "Indexing Useful Structural Patterns for XML Query Processing", IEEE Transactions on Knowledge and Data Engineering, 17(7), 2005, pp. 997-1009

68 Wang Lian, Nikos Mamoulis, David Wai-lok Cheung, Member, IEEE Computer Society, and S.M. Yiu Indexing Useful Structural Patterns for XML Query Processing IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 7, JULY 2005

69 Weidong Kou , David Lauzon , William O'Farrell , Kelvin Cheung , Richard Gregory , Kostas Kontogiannis at el " End-to-end E-commerce Application Development Based on XML" IEEE Computer Society Technical Committee on Data Engineering , pp 1-8 2000

70 Xiaojie Yuan , Xiangyu Hu, Dongxing Wu, Haiwei Zhang+ and Xin Lian' XML Data Storage and Query Optimization in Relational Database by XPath Processing Model' , © 2013 ACADEMY PUBLISHER

71 Xiaojie Yuan Xin Wang Chenying Wang Efficient XPath Evaluation Using a Structural Summary Index, 978-0-7695-3336-0/08 © 2008 IEEE DOI

72 Y. Chen, S.B Davidson, and Y. Zheng "BLAS: An Efficient XPath Processing System", in Proceeding of the ACM SIGMOD, 2004, pp47-58

73 YANLEI DIAO, MICHAEL J. FRANKLIN, HAO ZHANG,PETER M. FISCHER Path Sharing and Predicate Evaluation for High-Performance XML Filtering *published in ACM TODS, December 2003*

74 Yanlei Diao, Peter Fischer, Michael J. Franklin, Raymond YFilter: Efficient and Scalable Filtering of XML Documents Proceedings of the 18th International Conference on Data © 2002 IEEE

75 Yi Chen , Susan B. Davidson, Yifeng Zheng "A bi-labeling based XPath processing system" Information Systems 35, 2010, doi: 10.1016/j.is.2009.05. ACM

76 Yi Jiang , Zhi min Zeng, Dong Zhan Zhang "An Efficient Encoding and Labeling Based Upon Continued Fraction for Dynamic XML Data", in World Congress on Software Engineering , IEEE DOI 10.1109/WCSE. 2009.150, PP 324-328

77 Yingjie Fan , Shuyun Wang , Chenghong Zhang Haibing Ma4, Yunfa Hu An Efficient Structural Index for Branching Path Queries The Ninth International Conference on Web-Age Information Management , IEEE 2008 pp119-126

78 YoshiKawa, M. and Amagasa, T. (2001). XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology,

79 Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos. From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. In *Proceedings of VLDB*, pages 237–248, 2003

80 Z. Liu, and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In proceedings of the 2007 ACM SIGMOD international conference on Management of Data, Beijing, China, pages 329-340, 2007

81 Zhewei Jiang, Cheng Luo, Wen-Chi Hou, Qiang Zhu, and Dunren Che Efficient Processing of XML Twig Pattern : A Novel One-Phase Holistic Solution

Reports

1 Iman Elghandour , Ashraf Aboulnaga , Daniel C. Zilio, Fei Chiang "XML Index Recommendation with Tight Optimizer Coupling", Technical Report CS-2007-22
July 11, 2007

2 Justin Zobel, Alistair Mo®at, and Kotagiri Ramamohanarao. Guidelines for Presentation and Comparison of Indexing Techniques. ACM SIGMOD Record, 25(3):10-15, 1996.

3 M Al-Nadawi, Dr Siobhan North, Dr Barry Eaglestone Indexing XML Databases: "Classifications, Problems Identification and a New Approach" , The University of Sheffield Department of Computer Science technical report 15th November, 2007

4 Mikael Fernandus Simalango "XML Query Processing and Query Languages: A Survey" Property of Amikelive.com – Technical Paper Series 25/10/2008

5 Mohammad Samir, Patrick Martin "XML Structural Indexes " Technical report No. 2009-560 June 2009 School of Computing Queen's University Kingston, Ontario, Canada K7L3N6

6 Mohammad, S., and Martin, P., 2009. Index Structures for XML Databases. In Li, C., and Ling, T. W. (Eds.). Advanced Applications and Structures in XML processing: Label Streams, Semantics Utilization and Data Query Technologies. IGI Global. pp. 98-124

7 Mohammed Al-Badawi, Dr. Siobhán North, Dr. Barry Eaglestone , research memorandum Indexing XML Databases: Classifications, Problems Identification and a New Approach, 15th November, 2007 in The University of Sheffield
Department of Computer Science

8 XML and E-Commerce Opportunities and Challenges , Harbinger Corporation ,March 14, 2000

9 XML-based Supply Chain Integration: A Review and a Case Study by Juha-Miikka Nurmilaakso ,A thesis to fulfill the requirements for the degree of Licentiate of Science in Technology. HELSINKI UNIVERSITY OF TECHNOLOGY 2013

Conference Proceedings

1 A. Gabillon. An Authorization Model for XML Databases. In Proceeding of the 2004 workshop n secure web service, Fairfax, Virginia, USA, pages 16-18, 2004

2 A. Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A.N. Rao, F. Tian, S. Viglas, Y. Wang, J.F. Naughton, and D.J. DeWitt, "Mixed Mode XML Query Processing," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

3 A. Schmidt, M.L. Kersten, M. Windhouwer, and F. Waas, "Efficient Relational Storage and Retrieval of XML Documents," Proc. Third Int'l Workshop Web and Databases (WebDB '00), 2000.

4 Alan Halverson, Josef Burger, Leonidas Galanis, Ameet Kini, Rajasekar Krishnamurthy,Ajith Nagaraja Rao, Feng Tian, Stratis D. Viglas, Yuan Wang, Jeffrey F. Naughton, David J. DeWitt Mixed Mode XML Query Processing Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

5 B. Ludascher, P. Mukhopadhyay, and Y. Papakonstantinou, "A Transducer-Based XML Query Processor," Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), 2002.

6 B. Sigurbjörnsson, J. Kamps, and M. Rijke. Processing Content-Oriented XPath Queries. In the Proceedings of the thirteenth ACM international conference on Information and knowledge management, Washington D.C., USA, pages 371-380, 2004

7 B.M. Monjurul Alom, Frans Henskens, Michael Hannaford DRXQP: A Dynamic Relational XML Query Processor ; 2010 Seventh International Conference on Information Technology 978-0-7695-3984-3/10 $26.00 © 2010 IEEE

8 Bo Zhang , Zhihua Geng , Aoying Zhou SIMP: Efficient XML Structural Index for Multiple Query Processing The Ninth International Conference on Web-Age Information Management 978-0-7695-3185-4/08 $25.00 © 2008 IEEE DOI 10.1109/WAIM.2008.25

9 Brian F. Cooper ,Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, Moshe Shadmon A Fast Index for Semistructured Data Proceedings of the 27th VLDB conference, Roma, Italy, 2001

10 C. Chung, J. Min, and K. Shim. APEX: An Adaptive Path Index for XML Data. In proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pages 121-132, 2002.

11 C. Li, T.W. Ling, and M. Hu, "Efficient Processing of Updates in Dynamic XML Data," Proc. Int'l Conf. Data Eng. (ICDE), pp. 13-22, 2006

12 C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. 20th ACM SIGMOD Int'l Conf.Management of Data (SIGMOD '01), 2001

13 D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML via XML Fragments. In the proceedings of the 26th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 151-158, 2003.

14 H. He and J. Yang, "Multiresolution indexing of XML for frequent queries", In Proc. of the 20th Intl. Conf. on Data Engineering, Boston, April 2004, pp. 683–694

15 H. Jiang, H. Lu, W. Wang, and J.X. Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data," Proc. 13th Australasian Database Conf. (ADC '02) , 2002.

16 H. Su, E.A. Rundensteiner, and M. Mani, "Semantic Query Optimization for XQuery over XML Streams," Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05), 2005.

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

18 Haifeng Jiangy Hongjun Luy Wei Wangy Jerey Xu Yuz Path Materialization Revisited: An Efficient Storage Model for XML Data (AICE2000), Canberra. Conferences in Research and Practice in Information Technology, Vol. 1

19 J. Chu-Carroll, J. Prager, K. Czuba, D. Ferrucci, and P. Duboue. Semantic Search via XML Fragments: A High-Precision Approach to IR. In the proceedings of the 29th anuual international ACM SIGIR conference on Research and development in information retrieval, pages 445-452, 2006

20 J. Kamps, M. Marx, M. Rijke, and B. Sigurbjörnsson. Best-Match Querying from Document-Centric XML. In proceedings of the 7th International Workshop on the Web and Databases (with ACM SIGMOD/PODS 2004), Paris, France, pages 55-60, 2004.

21 J. McHugh and J. Widom, “Query Optimization for XML,” Proc. 25th Int’l Conf. Very Large Data Bases (VLDB ’99), 1999.

22 Jong P. Yoon, Vijay Raghavan, Venu Chakilam, Larry Kerschberg BitCube: A Three-Dimensional Bitmap Indexing for XML Documents Jong Yoon, et. al., BitCube, Journal of Intelligent Information Systems, Vol. 17, November 2001

23 L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In the proceedings of 2003 ACM SIGMOD international conference on Management of data, California, USA, pages 16-27, 2003.

24 M. Mani and D. Lee. XML to Relational Conversion using Theory of Regular Tree Grammars. In Proceedings of the 28th VLDB Conference, Hong Kong, China, 2000.

25 M. Nicola, and B. Linden. Native XML Support in DB2 Universal Database. In proceedings of the 31st international conference on VLDB, Trondheim, Norway, pages 1164-1174, 2005.

26 M. Rys. XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005. In Proceeding of the 2005 ACM SIGMOD International conference Baltimore, Maryland, pages 958-962, 2005on Management of Data,

27 Matthias Nicola , Bert van der Linden "Native XML Support in DB2 Univeral Database " Proceedings of the 31st VLDB Conference , Trondheim , Norway,2005

28 Michael Kay "Ten Reason Why Saxon XQuery is Fast" Bulletine of the IEEE Computer Scociety Technical Committee on Data Engineering 2008

29 Mohammad Samir, Patrick Martin “XML Structural Indexes “ Technical report No. 2009-560 June 2009 School of Computing Queen's University Kingston, Ontario, Canada K7L3N6

30 Mohammad, S., and Martin, P., 2009. Index Structures for XML Databases. In Li, C., and Ling, T. W. (Eds.). Advanced Applications and Structures in XML processing: Label Streams, Semantics Utilization and Data Query Technologies. IGI Global. pp. 98-124

31 N. Bruno, N. Koudas, and D. Srivastava, “Holistic Twig Joins: Optimal XML Pattern Matching,” Proc. 21st ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’02), 2002

32 N. Fuhr, and K. Groβjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In the Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, Louisiana, USA, pages 172-180, 2001.

33 N. Zhang, V. Kacholia, and M.T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML," Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004

34 Nicolas Bruno, Luis Gravano, Nick Koudas, Divesh Srivastava Navigation- vs. Index-Based XML Multi-Query Processing Proceedings of the 19th International Conference on Data Engineering (ICDE'03) 1063-6382/03 $ 17.00 © 2003 IEEE

35 P. Rao and B. Moon, "PRIX: Indexing and Querying XML Using Prufer Sequences," Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004

36 Patrick O'Neil, Elizabeth O'Neil1,Shankar Pal, Istvan Cseri, Gideon Schaller,Nigel Westbury "ORDPATHs: Insert-Friendly XML Node Labels" SIGMOD 2004, June 13–18, 2004, Paris, France. Copyright 2004 ACM 1-58113-859-8/04/06

37 Q. Chen, A. Lim, and K. Ong. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data. In proceedings of the 2003 ACM SIGMOD international conference on Management of data, CA, USA, pages 134-144, 2003.

38 R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In proceedings of the 18th international conference on Data Engineering, Washington DC, USA, pages 129-141, 2002.

39 R. Kaushik, R. Krishnamurthy, J.F. Naughton, and R. Ramakrishnan, "On the Integration of Structure Indexes and Inverted Lists," Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'04), 2004.

40 Ryan H. Choi, Raymond K. Wong Efficient XQuery Join Processing in Publish/Subscribe Systems, Copyright c 2009, Australian Computer Society, Inc. This paper appeared at the 20th Australasian Database Conference (ADC 2009), Wellington, New Zealand, January 2009.

41 S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Y. Wu, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02), 2002

42 S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In Proceedings of the 6th annual ACM international information and data management, Washington DC, USA, pages 31-38, 2004workshop on Web

43 S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

44 S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V. Zolotov, "Indexing XML Data Stored in a Relational Database," Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04), 2004.

45 S. Paparizos, Y. Wu, L.V.S. Lakshmanan, and H.V. Jagadish, "Tree Logical Classes for Efficient Evaluation of XQuery," Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), 2004.

46 Samini Subramaniam , Su-Cheng Haw , Poo Kuan Hoong Mapping and Labeling XML Data For Dynamic Update, Second International Conference on Computer Research and Development , 978-0-7695-4043-6/10 $26.00 © 2010 IEEE DOI 10.1109/ICCRD.2010.173

47 Steven A. Demurjian, David K. Hsiao, David K. Hsiao,Douglas S. Kerr , Robert C. Tekampe , and Robert J. Watson. "Performance measurement methodologies for systems. Proceedings of the 1985 ACM annual conference on database the range of computing: mid-80's perspective: mid-80's perspective, Denver, Colorado, United States, 1985, Pages: 16-28

48 Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang. C., 2002. Storing and Querying Ordered XML Using a Relational Database System. In Proc. of the ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, pp. 204-215.

49 X. Li and G. Agrawal, "Efficient Evaluation of XQuery over Streaming Data," Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05), 2005

50 X. Yan, P. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In proceedings of the 2005 ACM SIGMOD international conference on Management of data, Baltimore, Maryland, pages 766-777, 2005

51 Xiaodong Wu et "A Prime Number Labeling Scheme for Dynamic Order XML Trees" in 20th International Conference on Data Engineering (ICDE04) 2004 IEEE pp 66-78

52 Xiaojie Yuan , Xiangyu Hu, Dongxing Wu, Haiwei Zhang+ and Xin Lian' XML Data Storage and Query Optimization in Relational Database by XPath Processing Model' , © 2013 ACADEMY PUBLISHER

53 Yanlei Diao, Peter Fischer, Michael J. Franklin, Raymond YFilter: Efficient and Scalable Filtering of XML Documents Proceedings of the 18th International Conference on Data Engineering (ICDE'02)
1063-6382/02 $17.00 © 2002 IEEE

54 Yi Chen , Susan B. Davidson, Yifeng Zheng "A bi-labeling based XPath processing system" Information Systems 35, 2010, doi: 10.1016/j.is.2009.05. ACM

55 Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos. From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. In *Proceedings of VLDB*, pages 237–248, 2003

56 Z. Liu, and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In proceedings of the 2007 ACM SIGMOD international conference on Management of Data, Beijing, China, pages 329-340, 2007

Reference Site

1 Chemical XML web site www.cxml.org

2 ebXML web site: www.ebxml.org

3 http:// www.xmlns.com

4 http://about.reuters.com/researchandstandards/events/2001/05/xml-europe/how-and-where-xml-is-changing-the-markets-paper.htm

5 http://webdesign.about.com/od/xml/a/aa060401a.htm

6 http://www.arobotext.com

7 http://www.datachannel.com

8 http://www.enigma.com

9 http://www.ipnet-solutions.com

10 http://www.kinecta.com

11 http://www.russell.com

12 http://www.sequoiasoftware.com

13 http://www.service-architecture.com/articles/xml/finance_xml.html

14 http://www.usfca.edu/fac-staff/morriss/651/tech_projects/XML/how_useful_is_xml_in_e-commerce.htm

15 http://www.wava.com

16 http://www.webxsystems.com/Pharma.htm

17 http://www.xml.com/pub/a/2001/08/15/xml-ecom.html

18 J. Robie et al. XQL (XML Query Language). http://www.ibiblio.org/xql/xql-proposal.html. 8/1/1999

19 Michael Kay "XSLT and Xpath Optimizatin " "http:// saxon.sf.net"

20 Microsoft's BizTalk : www.microsoft.com/biztalk/default.asp

21 Online Computer Library Center, 'Introduction to the Dewey Decimal Classification", http://www.oclc.org/dewey/

22 Open Financial Exchange : www.ofx.net Special issue of Communications of the ACM on e-services, June 2003

23 http:// www.commerce.net

24 W.E. Kimber , "HyTime and SGML: Understanding the HyTime HYQ Query Language" ,http://ftp2.de.freebsd.org/pub/sgml/ifi.uio.no/HyTime/HyQ. Kimber

25 W3C Consortium, http://www.w3.org, 2006

26 W3C Consortium, XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/, 2006.

27 W3C Consortium, XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/, 2006

28 W3C Website. Extensible Markup Language (XML) 1.0 (Fourth Edition). Online:http://www.w3.org/TR/REC-xml/, Accessed on: 30/10/2006

29 W3C Website. Object Data Model (DOM). Online: http://www.w3.org/DOM/, Accessed on: 15/10/2007

30 W3C Website. XML Query Language (XQL). *Online:http://www.w3.org/TandS/QL/QL98/pp/xql.html*, Access on: 07/10/2007

31 W3C Website. XQuery 1.0: An XML Query Language. Online: http://www.w3.org/TR/xquery/,Access on: 30/01/2007

Annexure-IV

IBMDB2.9 and SAXON Introduction

IBM DB2.9:-

IBM DB2 is used for implementation of XML data in RDBMS format and with the new data type ".xml". In 2006, the IBM announced "Viper" which is codename for DB2.9. The IBM claimed that this was the first relational database to store XML file with relational database system, native database system and in text file format. [12]. IBM DB2 introduced three storage types. These are (1) External File storage (2) Relational File Storage (3) Native Storage.

IBM DB2 Storage:-

The XML data is stored in external file i.e. as text file with '.XML' as extension as show in Fig 4.1 . This external file is accessed through either writing our own application, or XML wrapper class.

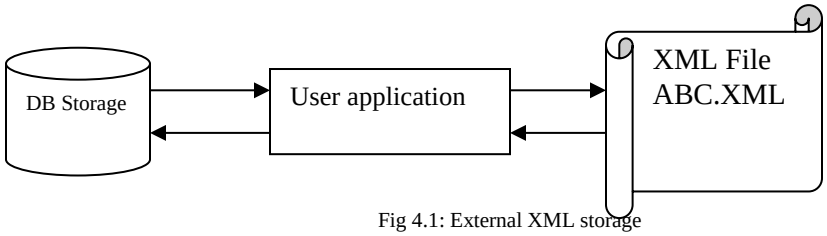


Fig 4.1: External XML storage

This type of storage uses the following data types for storing XML file:-

- varchar (up to 32,672 byte, 32704 bytes on DB2 for z/OS))
- varchargraphics (It doubles up to 16,336 double-byte characters long, 16,352 on z/OS)
- clob (up to 2Gb -1 byte)

This is easy storage method for data-management, and more secure. But it has the following drawbacks:-

(1) XML documents cannot be validated against the schema.

(2) The identification of the XML document is difficult only by looking at the catalog metadata.

(3) Indexing is not possible on XML data.

(4) Hence, XML query execution is not possible.

In XML-RDBMS format XML data is shredded and elements and attribute tables are created. The shredding processes are classified into two types:-

(1) Partial Shredding: - Only a subset of the elements or attributes from each incoming XML document is shredded into relational tables. This method is helpful for those applications, where all XML document values are not useful

(2) Hybrid XML storage: - As the name suggests it is a combination of relational and native database storage. This storage is preferred when XML document is complex. All information i.e. selection of shredding methods and a number of tables to be used for storage, is recorded into a file called as Document Access Definition File (DAD) file. The DAD file specifies how the XML document structure relates to DB2 data that resides into relational tables. DAD maps XML nodes either by RDB node Mapping or SQL Mapping. RDB nodes specify the relationship between tables and columns in the relational model which are in turn, associated with elements and attributes of the respective values in the XML documents. The SQL mapping is a template based language. The user specifies an SQL SELECT statement and states where the results of query should be placed within the template. The SQL mapping can only be used to transfer data from relational tables to an XML document, and cannot be used for dividing XML document into the relational tables.

XML Shredding can be useful when:-

2. Incoming XML data is only for feeding an existing relational table.
3. The primary goal is to enable existing relational applications to access XML data.
4. The XML documents do not represent logical business objects that should be preserved.
5. The structure of XML data is such that it can easily be mapped to relational tables.
6. Rarely needs to reconstruct the shredded documents.
7. Querying and updating the data with SQL is more important than insert performance.

Shredding is not useful when

- XML data is complex and nested, and difficult to map to a relational schema.
- Mapping XML format to relational schema leads to a large number of tables.
- XML Schema is highly variable or tends to change over time.
- The user has to manage XML documents as intact business objects.
- The application frequently needs to reconstruct the shredded documents or parts of them.

IBM DB2 is introduced new native data type ".xml" [16] [10] [9] which supports XML data as hierarchical data format and maintain structural relationship. This storage format is called as

native storage and also called as “coarse grain” storage. The second format of storage is fine grain storage. In fine grain storage, each XML element node is stored into a column. If that element information is not available into next XML element node then it is stored as ‘NULL’ value

(1) Coarse-grained Storage: - Only one table is used for storing XML file. The table has two columns: the first column is used ‘document_id” which identify the XML document and second column store entire XML document with data type or ‘XML’ or ‘CLOB’/’BLOB’. The ‘.XML’ storage, stores XML in tree structure or hierarchical format so all types of XML queries can performed by these storage. The ‘CLOB’ or ‘BLOB storage, stores XML file into text format hence XML file is considered as text file and cannot execute XML queries on it. The probable file structure is explain in Table 4.3

Name	Data type	Key
DocuementID	Number	Primary Key
DocumentName	VARCHAR	----
Document ,	CLOB or XML	

Table 4.3: Table structure for XML file

(2) Fine-Grained: - XML document be decomposed into elements, attributes, and character data region – each of which grouped together into table. The additional table is used for storing parent/child relationship. Following command is used for storing information.

```
Insert into element (1, 1, elementname, elementvalue, 1)
Select element name, element value
From cd1,
XMLTABLE ('$d/CATALOG/CD’ passing CDINFO as “d"
COLUMNS
Element name VARCHAR (32) PATH ‘name ()',
Element value archer (1000) PATH '(. [self::element ()]/text ()')) as T ;
```

Index Structure:-

IBM DB2 creates three kinds of indexes: DOCID index, NODEID index, XML value indexes. NODEID, DOCID are implicit index. A DOCID index is a regular index on the hidden DOCID column. The 'NOEID index' in created on each row of the table. It maps logical nodeID to physical RecordIDs. Once query is submitted to DB2 the query compiler compares the query predicates. Query execution plan is created with available index.

DB2 introduced three query operator are used for the query execution: XSCAN (XML Document Scan), XISCAN (XML Index Scan), XANDOR (XML Index Ending). XSCAN operator evaluates query predicates and extracts document fragments and values. XISCAN operator performs lookups or scans on XML indexes. The XANDOR operator evaluates two or more equality predicates simultaneously by driving multiple XISCANs [9]

SAXON:-

SAXON is open source XML query processer system implemented by Michal Kay la et [17]. It is implemented in Java and implements navigation approach. SAXON is navigational implementation of the XML file.

SAXON query processor developed by Michael Kay. It has open-source and also closed-source commercial versions. August 2009 a SAXON version 9.2 is available with three versions: home edition (HE), professional edition (PE), and enterprise edition (EE). The home edition is open source and free, the other versions are available under commercial licenses. For Our experiment we select the HE edition.

SAXON stored XML file in Node Info format. This Node info is implemented into either Linked tree or Tiny Tree. Linked tree structure is a link list. The link list nodes contain parent's node information and list of their children. The 'Tiny Tree' represents documents using six principal arrays of integers. The total size of these six integer array 19 bytes per node. These arrays contain one entry for each node [18]:-

(1) Node kind: - consider three type of node element, text and comment node. For attribute and namespace node additional sets of array

(2) Node code: - The name code contains a unique identifier for the URI/local-name pair in 20 bits. The ‘Name pool’ is an array which is used to allocate new codes.

(3) next sibling pointer

(4) depth of tree

(5) two values :- In case of elements points to the first attributes and the first namespace node

(6) Remaining two nodes are Pointer to text node: - which will points to the textual content in a text buffer.

Query exaction algorithm is written in java. It is open sources and it creates implicit index on object_id. Saxon is not database product. It cannot afford luxury of data storage and memory residential indexing.

ANNEXURE-I

Comparative Study of XML Storage

XML-RDBMS Approach			
Proposal of Index	**XML -RDBMS data representation**	**Advantages**	**Disadvantage**
EDGE Approach			
Edge Approach	· Edge approach(Label,Source,Target,Flag,Valur)	This is first proposal which overcome the drawback of file system and ORDBMs sysetm	It involves a number of join operations. This approach may fail to process large twig queries efficiently
	o The key idea here is using an attribute pair (Source; Target) the query can answer	Simplest mapping technique which stores an XML document in single Edge table.	1) Level information is not stored in the table therefore it is quite difficult to determine the ancestor-descendant relationship between the nodes
		Insertion and deletion of an object or attribute is simple.	2) Unable to process a query with the combinations of parent/child and ancest or descendant edges.
Binary Approach	o Edge(Source, Ordinal, Target, Label, Flag, Value)	Sinling realtionship maintainance is simple	
	o The Ordinal attribute records the ordinal of the edge among its siblinkgs		
	o A Flag value indicates whether the target nodeis an inter-object reference (ref) or points to a value (val).		
Universal Approach	· Generates a single Universal table to store all the edges. The structure of the Universal table is as follows,	· Stores XML document in single Universal table.	· Causes redundancy for multi-valued attributes and the table is not normalized

	o Universal(source, ordinaln1, flagn1, targetn1, ordinaln2, flagn2, targetn2 , : : :, ordinalnk, flagnk, targetnk)	· Deletion of an object is simple	· -Deleting an attribute in XML document requires massive updates of ordinal attribute in the Universal table
	· It has many fields which are set to NULL. And has a great redundancy . i.e Universal table is denormalized data.		
NODE Appraoch			
Extension of Inverted List	· Inverted index is extended for T-index and E-index.	Efficiently determine A-D and P-C relationships and support both structural and full-text queries	· Cannot processes "twig / Tree query "
	· Text words are index in a T-index similar to that used in a traditional IR system.	· It utilized hardware cache memory hence performance of query execution is high	· For query processing the execution time is high because number of joins operation is involved in it .
	· E-index is created for element index. o E-index (docno,begin:end, level) for an element .	· Because RDBMS database utilized existing features database engine like query optimization, query execution, scalability and concurrency control and recovery .	This Approach doesnot support dynamic updates operatin
	o T-index (docno,wordno,level).	Effective Merge join algorithm is used which is effectiev for multiple predicates	
s-XML Model	- Two tables which are the ParentTable and ChildTable.	Eficient for dynamic Updates	- Calculation required to determine ancestor/descendant relationship.
	§ *ChildTable (IdNode, Level, Child, LParent)*	- Only two tables involved therefore there will be less joins required for query processing.	
	§ *ParentTable (IdNode, LParent, SelfLabel)*	- -Order information and parent child information stored efficiently	
	§ Each node will be labeled as (l,[np,dp],[n,d]),	- -Queries involving parent-child and ancestor-descendant edges can be processed easily though ancestor-descendant relationship	

MAXDOR[12]	§ A master table for documents is needed. It is called "documents",	o Perform Query and record the information of document Update efficiently without reconstruction of the XML file.	Performace with other object not yet check.
	§ Documents(documentID, documentName, docElement, totalTokens, schemaInfo)	o Extracts structural information efficiently and makes use of it for the mapping process	§ Only one table is maintain for XML data storage . Redundancy may exist.
	A second table to store the actual contents for all documents	Reduced join operation	§ Partial reconstruction of table is needed when new node is added in the XML file.
	§ Tokens(documentId, tokenId, LS, Par, Rs, tokenLevel, tokenName, tokenValue, tokenType).	§ Reconstruction of original XML documents is easier and faster.	§ Query computation cost is high for containment query and twig query . Since only one table is maintain ; lots of inner join operation is required to rewrite the query .
	o A Four dimensional labels is used to label the XML document contents. This label solved all types of queries	§ Maintaining document structure.	§ Order queries and content matching can effectively solved. (Partial Matching ?)
	o In FDLS, labels, as follows:	§ Preserve the ordering nature of XML data.	
	o Node (NodeID, left-sibling, parent, right-sibling).	§ Ability to perform semantic search.	
Compressed XPath Query Tree (CXQT)	o The schema of XML table is:	Address the problem of consistent storage and efficient query for XML data	
	(nodeid,type,name,vale,path) where nodeid is primary key	Effectively execute the Queries	
Path Materialization			

XPARENT[3]	§ Four tables are used for maintain relationship:-	It proposed a flexible multi-level query translation schema for model-mapping-based approaches	§ For A-D style of queries additional table is required which lead the problem of additional disk space and update overhead.
	1) LabelPath(ID, Len, Path) 2) DataPath(Pid, Cid) 3) Element(PathID, Did, Ordinal) 4) Data(PathID, Did, Ordinal, Value)	efficiently solved P-C Queries	
		Dynamic XML File storage	
		reduced join operation for containment style of queries	
		Can preserve Original order of the tree	
XREL[1]	XML data is represented into four relational schemas:	startno, endno provide a unique identofocation to the XML node	Updating node, deleting node is possible but node update cost is high
	- Element(docID, pathID, start, end, index, reindex)	§ -Region information stored in the table which allows the processing of a query with the combination of ancestor/descendant and parent-child edges	For performance improvement instead of absolute data type is required
	- Attribute(docID, pathID, start, end, value) , - Text(docID, pathID, start, end, value), - Path(pathID, pathexp).	§ -Since table schema is based on the node types, insertion and deletion of an attribute or object are simple.	· Too many tables which require excessive joins for complex query and query with various combinations
		§ -Maintains path information in separate table which will be referenced by other tables and eventually reduces database size.	The approach does not use a special fulltext search system, it may not achieve high performance on query retrieval.
Reverse Path Approach	o Schema (ORDPATH, TAG, NODTYPE, VALUE, PATHID)	o This technique is beneficial to the applications which know the expected query workload and will benefit by indexing only the paths occurring in the queries.	o When New node is added into XML , Index updating cost is high. Database need to be reconstruct
	Where § ORDPATH :- preserve structural relationship within XML instance	PM Approach cannot solve "like %" Pattern queries	o The approach reduced string matching task but doesnot discussed effective implementation .
	§ TAG :- XML tag information . Instead of storing string value, it stores	PM Approach cannot solve recursive XML queries	

	integer value of TAG.		
	§ NODE_TYPE :- stores type of node.	creaing Index on BLOB column (instead of B+index on it)	
	§ VALUE :- it stores the node's value , if exists otherwise it is NULL.		
	§ PATH_ID :- It stores path value from the root to the node. This column represents all paths in the tree (similar to dataGuide)		
BLASÉ	o (plable ,start,end,level,data)	o The approach solved complex XPath query by using less join operation.	
	o Where plabel – is P-label of n.	§ Reducing number of joins and disk accesses required for complex XPath queries.	
	o Start and end are start and end tag positions of n in the XML document.	§ Optimizing join operations.	
	o Level is the level of n.	efficient implementation of SPM SPM	
	o Data is used to store the value of n if there is any value otherwise data is set to null.		

Summary information of Node Labeling Schema :-

Label Schema	Advantage	Disadvantage
PrePost coding or interval coding	- Storing (start,end) pairs requires only modest storage space - Only two number comparison determine ancestor-descendent parent-child relationship in const time - Process '/' or '//' axes ,all other axes	While Updating XML tree , re-labeling all ancestor and descendent node Searching cost of a node is very high thus query performance time is high.
Floating Point schema	- Represent node by floating point number . - Using number comparison determine parent-child, ancestor-descendent relationship . - It process '/' and '//axes type of queries - Node code itself suggest ancestor and descendent relationship	The storage of floating point number in computer memory is limited by accuracy , it is still unable to avoids the re-labeling of XML element.
Prime	- Only 'Prime number' is used to label node. - Can determine parent relationship easily - New node insertion avoid re labeling process. - Can answer '/' and '//' axis queries.	Code is difficult to determine ancestor-descendent relationship Updating cost is high Query performance is low
ORDPATH	- Represent node by all number - Even numbers are reserved for node update. - It avoid re labeling of node while insertion - Code can handle	Wastage of half of number Fails when frequently insertion of node is happen It not orthogonal poor in query performance uses labels that do not

		reflect ancestor and their by looses some of its expressivity neither support decision of the next sibling relation nor reconstruction of sibling relation nor reconstruction of sibling or child node therefore not suitable for structural queries.
QED	- Only quarter number are used to represent code . - Can avoid completely re-labeling - Perform lexicographical comparison while inserting new node. - Code is orthogonal - Can handle '/' and '//' queries	When an XML node is inserted with new node continuously, the code length of the node grows too rapidly. Label Size is large Node update and query performance is not good Difficult to determine sibling and parent-child relationship.
CDBS	- Code is compact - Insert node without relabeling - Orthogonal to specific labeling schema - Query processing time is less as comparison with Floating-point, ORDPATH, v-binary and f-binary - Query performance is good	Cannot completely avoid relabeling node. Determining ancestor - descendent , parent-child relationship is difficult (by first lookup)
IBSL	- Update cost (XML node modification) is low. - No re labeling required - Decrease label size when large	Label size increased as XMl tree grow in deeper.

	number of nodes are inserted and deleted without sacrificing query performance - Existing label is reused - Determining ancestor -descendent , parent-child relationship is difficult (by first lookup)	For processing twig query 'query performance decrease.' Because of label size is long Not an orthogonal
CEF	- Supports all the structural query to determine the ancestor-descendent , parent–child relationship and document order relationship - It is orthogonal schemes , thus it can applied broadly to different labeling schema or other application to efficiently process the updates - Supports update, it can insert node between ant two consecutive node without recoding existing label - Provide solution on high cost of re-labeling XML node when updating XML document - It control rapid growth of code length is under control when continuously inserting nodes at XML node. - Can answer '/' and '//'query	-Label size increase as XML tree grow in depth wise as well as breath wise Query performance is low Node updating cost is very high

Annexure- II

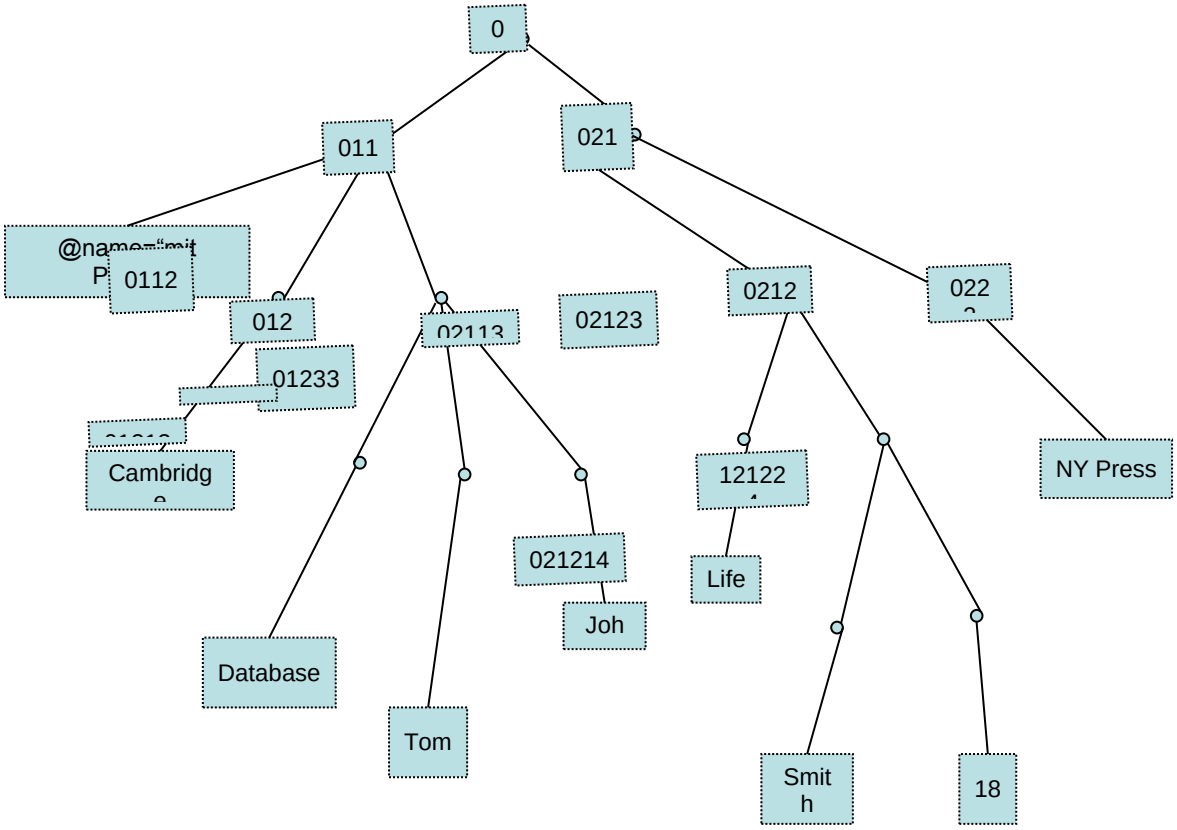
Case Study of PESS Model and Query Execution Plan

Lets Following XML file

```
<publishers>
    <publisher>
        <address> Cambridge </address>
          <book>
              <title>Databases</title>
              <author> Tom </author>
              <author> John </author>
        </book>
     </publisher>
     <publisher>
          <book>
             <title> Life </title>
            <author>
                    <name> Smith </name>
                    <age> 18 </name>
             </author>
          </book>
          <name> NY  Press </name>
     </publisher>
</publishers>
```

All above algorithms can diagrammatically expresses as :-

(a) XML Data Model Creation :-



Creating RDBMS schema:-

(b) Creating of Path table-

a. Creating Element Table :-

Element	Code
Publisher	1
Address	2
Book	3
Title	4
Author	5
Name	6
Age	7

Creating path_table :-

Path-desc	Path_id
/publisher	1
/publisher/publisher	11
/publisher/publisher/address	112
/publisher/publisher/book	113
/publisher/publisher/book/title	1134
/publisher/publisher/book/author	1135
/publisher/publisher/book/author/name	11356
/publisher/publisher/book/author/age	11357
/publisher/publisher/name	116

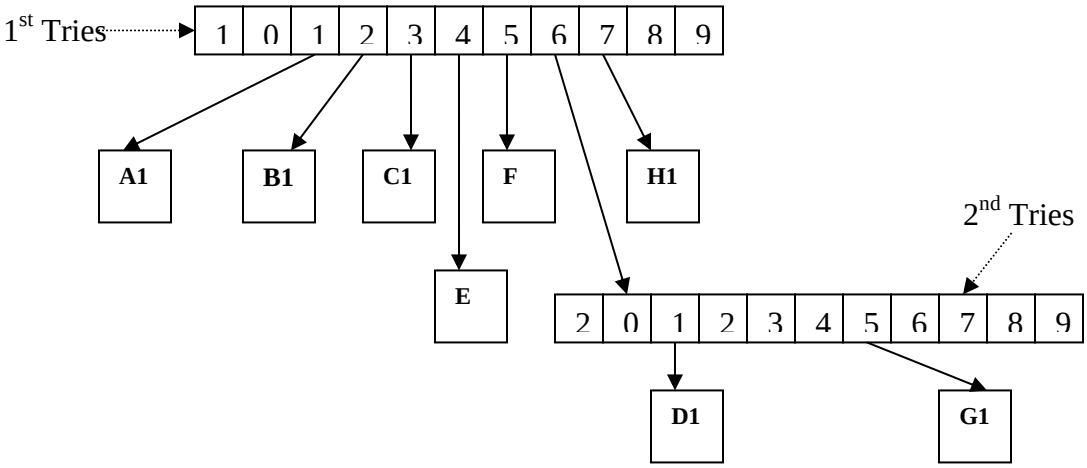
- Creating Data_node table(Node_Id, Value)

Node_id	Value
0112	“Cambridge”
01213	“Database”
01223	“TOM”
01233	“John”
0222	“N Y Press”
02113	“Life”
021214	“Smith”
021224	18

Creating Rtable (PNode , CNode) :-

PNode	CNode
0	011
0	021
011	0112
011	0122
0122	01213
0122	01223
0122	01233
021	0212
021	0222
0212	02113
0212	02123
02123	021214
02123	021224

Algorithm 4(page no) :- Generate M-way tries on Path id:-



Index File

Index Id	Index On	
I1	Index on Path-id	Create index on path id :- CREATE INDEX PIndex ON cd10 (Pathid ASC)
I2	Index on Node-id	Create index on path id CREATE INDEX NIndex ON cd10 (Nodeid ASC)

Annexure-III

Sample of XML files and their Node Structure used for Experiment

```
<?xml version="1.0"?>
 <CATALOG>
                                    <CD>
                <TITLE>Empire Burlesque</TITLE>
                <ARTIST>Bob Dylan</ARTIST>
                <COUNTRY>USA</COUNTRY>
                <COMPANY>Columbia</COMPANY>
                <PRICE>10.90</PRICE>
                <YEAR>1985</YEAR>
  </CD>
</CATALOG>
```

Name of XML File :- CD.XML (Data centric document) (C2)			
Level :- 3			
Detail Description :- contain 100 node in file at level1			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	1	100	100
3	6	6*(100)	600
Total Number of Node			701

SET 1.3:- Total number of records 1000

Name of XML File :- CD.XML (Data centric document) (C3)			
Level :- 3			
Detail Description :- contain 1000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	1	1000	1000
3	6	6*(100)	6000
Total Number of Node			7001

SET1.4:- Total number of records 10000

Name of XML File :- CD.XML (Data centric document) (C3)			
Level :- 3			
Detail Description :- contain 1000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	1	900	900
3	6	6*(900)	5400
Total Number of Node			6301

SET 1.5:- Total Number of records 50,000

Name of XML File :- CD.XML (Data centric document) (C3)			
Level :- 3			
Detail Description :- contain 1000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	1	50000	50000
3	6	6*(50000)	300000
Total Number of Node			350001

SET 1.6:- Total Number of records 1, 00,000

Name of XML File :- CD.XML (Data centric document) (C3)			
Level :- 3			
Detail Description :- contain 1000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	1	99999	99999
3	6	6*(99999)	599994
Total Number of Node			699994

SET2:- WSINFO :-

```
<?xml version="1.0" encoding="UTF-8"?>
<current_observation>
  <credit>NOAA's National Weather Service</credit>
  <credit_URL>http://weather.gov/</credit_URL>
    <date>
   <date1>01/07/2011</date1>
   <image>
    <url>http://weather.gov/images/xml_logo.gif</url>
    <title>NOAA's National Weather Service</title>
    <link>http://weather.gov</link>
  </image>
  <suggested>
                <suggested_pickup>15 minutes after the hour</suggested_pickup>
                <suggested_pickup_period>60</suggested_pickup_period>
          </suggested>
   <location1>          <location>Olathe, Johnson County Executive Airport, KS</location>
                        <station_id>KOJC</station_id>
                        <area>
                                <latitude>NA</latitude>
                                <longitude>NA</longitude>
                                 <evalu>    <elevation>NA</elevation>        </evalu>
                        </area>
          </location1>
  <observation_time>Last Updated on May 18, 8:53 am CDT</observation_time>
  <weather>Fair and Breezy</weather>
  <temparature>
     <temperature_string>68 F (20 C)</temperature_string>
                <measurement>
                    <temp_measure>
                                <temp_f>68</temp_f>
                                <temp_c>20</temp_c>
                                <relative_humidity>63</relative_humidity>
                     </temp_measure>
                            <wind_measure> <wind_string>From the South at 21 Gusting to 28 PH</wind_string>
                        <wind-detail><wind-direction>  <wind_dir>South</wind_dir>  <wind-chill>
          <windchill_string>Not Applicable</windchill_string>
          <windchill_f>Not Applicable</windchill_f>
          <windchill_c>Not Applicable</windchill_c>
 <visib> <visibility>8.00 mi.</visibility>    </visib>
 </wind-chill></wind-direction>
          <wind_degrees>170</wind_degrees>
          <wind-mp>
                <wind_mph>20.7</wind_mph>
          <wind_gust_mph>28</wind_gust_mph>
          </wind-mp></wind-detail>
           </wind_measure>
           <pressure>
                <presuure-detail>
                <pressure_string>29.78&amp;quot; (1007.4 mb)</pressure_string>
                <pressure_mb>1007.4</pressure_mb>
                <pressure_in>29.78</pressure_in>
                <dowpoint>
                    <dewpoint_string>55 F (13 C)</dewpoint_string>t
                     <dewpoint_f>55</dewpoint_f>
                    <dewpoint_c>13</dewpoint_c>
                    <heat> <heat_index_string>75 F (24 C)</heat_index_string>
                       <heat_index_f>75</heat_index_f>
                      <heat_index_c>24</heat_index_c></heat>
          </dowpoint>    </presuure-detail>  </pressure>
                </measurement>
          </temparature>
  <companyinfo>
</current_observation>

          <two_day_history_url>http://www.weather.gov/data/obhistory/KOJC.html</two_day_history_url>
                                <ob_url>http://www.nws.noaa.gov/data/METAR/KOJC.1.txt</ob_url>
                                <disclaimer_url>http://weather.gov/disclaimer.html</disclaimer_url>
                                <copyright_url>http://weather.gov/disclaimer.html</copyright_url>
                                <privacy_policy_url>http://weather.gov/notice.html</privacy_policy_url>
          </companyinfo>
 </date>
 <date>
```

SET 2.1:- Number of records 10

Name of XML File :- WS.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 10 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	12	12
3	8	(8*10)	80
4	15	(15 *10)	150
5	6	(6*10)	60
6	7	(7 * 10)	70
7	4	(4*10)	40
8	4	(4*10)	40
9	3	(3*10)	30
Total Number of Node			483

SET 2.2: Number of records 100

Name of XML File :- WS100.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 100 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	105	105
3	8	(8*105)	840
4	15	(15 *105)	1575
5	6	(6*105)	630
6	7	(7 * 105)	735
7	4	(4*105)	420
8	4	(4*105	420
9	3	(3*105)	315
Total Number of Node			5041

SET 2.3:- Number of records 1000

Name of XML File :- WS1000.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 1000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	1005	1005
3	8	(8*1005)	8040
4	15	(15 *1005)	15075
5	6	(6*1005)	6030
6	7	(7 * 1005)	7035
7	4	(4*1005)	4020
8	4	(4*1005	4020
9	3	(3*1005)	3015
Total Number of Node			48241

SET 2.4:- Number of records10, 000

Name of XML File :- WS10000.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 10000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	10000	1005
3	8	(8*10000)	80000
4	15	(15 *10000)	150000
5	6	(6*10000)	60000
6	7	(7 * 10000)	70000
7	4	(4*10000)	40000
8	4	(4*10000)	40000
9	3	(3*10000)	30000
Total Number of Node			4,71,006

SET 2.5:- Number of nodes 50,000

Name of XML File :- WS50000.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 50,000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	49990	49990
3	8	(8*49990)	599880
4	15	(15 *49990)	749880
5	6	(6* 49990)	299940
6	7	(7 * 49990)	349930
7	4	(4*49990)	199966
8	4	(4*49990)	199966
9	3	(3*49990)	149970
Total Number of Node			23,49,530

SET 2.6:- Total number of records 1, 00,000

Name of XML File :- WS100000.XML (Data centric document) (C2)			
Level :- 10			
Detail Description :- contain 1,00,000 node in file			
Level No	Children per node	Total number of node	Total number of Children
1	1	1	1
2	12	1,00,003	1,00,003
3	8	(8*1,00,003)	8,00,024
4	15	(15 * 1,00,003)	15,00,045
5	6	(6* 1,00,003)	6,00,018
6	7	(7 * 1,00,003)	7,00,021
7	4	(4* 1,00,003)	4,00,012
8	4	(4* 1,00,003)	4,00,012
9	3	(3* 1,00,003)	3,00,009
Total Number of Node			50,00,145

3.2: Documents-centric Store 100 records

Mix-mode File

```
<?xml version='1.0'?>
<memoinfo>
<memo1>
<header>
        <from>John Doe</from>
        <to>Jack Smith</to>
        <subject>
                <name>XML at WhizBang</name>
                 <year>2</year>
        </subject>
</header>
<body>
      <para1>Have you heard of this new technology, XML? It looks promising.
          We could use XML to simplify our e-commerce and launch new services.
          Check this web site. They have just  released  XML by Example  with lots of useful
          information and some great examples. I have already  ordered two copies! We
          could use XML to simplify our e-commerce and launch new services. It is also useful
          for the web site: you complained it was a lot of work, apparently XML can simplify
          the maintenance.
     </para1>
</body>
```

Mix-Mode approach (For storing Data Level 10)

<**memoinfo>**
<memo>
<header>
<from> J ohn Doe</from>
<to>Jack Smith</to>
<subject>
<name>XML at WhizBang</name>
<year>1</year>
</subject>
</**header>**
<body>
<para1>Have you heard of this new technology, XML? It looks promising. It is similar to HTML but it is extensible. All the big names (Microsoft, IBM, Oracle) are backing it.
<para2>We could use XML to simplify our e-commerce and launch new services. It is also useful for the web site: you complained it was a lot of work, apparently XML can simplify the maintenance.
<para3>Check this web site <url>http://www.w3.org/XML</url> for more information. Also visit Que (<url>http://www.mcp.com</url>). They have just released XML by Example with lots of useful information and some great examples. I have already ordered two copies!
<para4>We could use XML to simplify our e-commerce and launch new services. It is also useful for the web site: you complained it was a lot of work, apparently XML can simplify the maintenance.
<para5>Check this web site **<url>**http://www.w3.org/XML**</url>** for more information. Also visit Que (<url>http://www.mcp.com</url>). They have just released XML by Example with lots of useful information and some great examples. I have already ordered two copies!
<para6>We could use XML to simplify our e-commerce and launch new services. It is also useful for the web site: you complained it was a lot of work, apparently XML can simplify the maintenance.
<para7>Check this web site **<url**>http://www.w3.org/XML**</url>** for more information. Also visit Que (<url>http://www.mcp.com</url>). They have just released XML by Example with lots of useful information and some great examples. I have already ordered two copies!
<para8>We could use XML to simplify our e-commerce and launch new services. It is also useful for the web site: you complained it was a lot of work, apparently XML can simplify the maintenance.
<para9>Check this web site **<url>**http://www.w3.org/XML**</url>** for more information. Also visit Que **(<url>**http://www.mcp.com**</url>).** They have just released XML by Example with lots of useful information and some great examples. I have already ordered two copies!
<signature>June**</signature>**
<para10>We could use XML to simplify our e-commerce and launch new services. It is also useful for the web site: you complained it was a lot of work, apparently XML can simplify the maintenance.
<para11>Check this web site **<url>**http://www.w3.org/XML**</url>** for more information. Also visit Que (<url>http://www.mcp.com</url>). They have just released XML by Example with lots of useful information and some great examples. I have already ordered two copies!
</para11>
</para10>
</para9>
</para8>
</para7>
</para6>
</para5>
</para4>
</para3>
</para2>
</para1>
</body>

XML three Product Comparision Study

	IBM dB2	SAXON	eXit
Storage	Three Types of Storage	The SAXON is an open source XML query processor developed by Michael Kay. It is not a database product	A unique number is assigned to each XML node; these node_ids maintain the hierarchical relationship among the XML nodes. The 'level-m' numbering schema is used for assigning the numbers to the nodes.
	External File Storage		
	XML data is sored in externla file		
	XML data file is stored by using user defined data	**Three Compnents can be used in SAXON**	
	Realtional Database storage	**tree constructo**r :- XML docuement is parsed and constrct XML tree in memeory	
	Partial Shreading :- Only a subset of elements or attributes from each incoming documents are shreadded into relational table	**tree navigator** :- This class is used desired XML node in the XML tree	It store XML information into four files: Collection.dbx, DOM.dbx, element.dbx, words.dbx
	Hybride Shreading :- It is combination of relatinal and native database storage	**The style sheet compiler** :- It analyses the XML document prior to execution	
	Native XML sorage:-	**The style sheet interprete**r: - This is the core component of the saxon	Collection.dbx:- For every relationship in an XML file a object is created . The 'collection_id' is a unique identifier for each hierarchical relationship.
	Special type ".xml" is used for storing XML file	**The *outputter* componen**t:- It is used to control the output process	
	Classify in three types:-		
	Coarse-grain :- Two columns table is used for storing the XML data. The first column stores the identification of XML document, and the second column stores XML document itself.		dom.dbx:- For each document node, node_id is created and its address is stored into data pages.
	Medium Grain :- The XML document is very large. It cannot be stored into a row, so the document is sliced and each row of the table stores a slice of the document. This storage is called as medium grain		element .dbx:- The 'name_id' is a unique node identifier for each element and attribute node
	Fine Grain :-Individual components of the XML document are stored in columns of multiple tables of the database. It is nothing but the RDBMS storage.		words.dbx:- This is nothing but the collection of text node information

XML three Product Comparision Study

INDEX	Two types of Index :-	It is not a database product, hence it doesn't maintain persistent indexes nor collect statistical data which is used by its optimizer	Three types of Index :-
	System Generated Index:- Indexes are "DOCID " index or "NODE ID" index		**Structural Index** :- Index on elements and attributes of the document.
	User defined Index :- These indexes are "Value Index" and " Full text Index"		Range Index :- These indexes are created on text nodes and attributes of the document.
	Three operators/functions are used for defined XML index XSCAN, XISCAN, and XANOR		Spatial Index
Query Processor	the query compiler compares the query predicates with existing index definitions and determines if any available indexes can be used to execute the query. This process is known as "index matching".	The query processing is divided into three phases : parsing, static analysis and run-time execution	Query Execution Steps:-
	It lists the set of eligible indexes for a given query.	XML query parsing - XML query tokens are recognized , symbols are analysis and syntacticla checkinging is performed	Step1:- The query is split into chain of basic unit
	This set of index file is an input to the cost-based optimizer which finds out query execution cost and decides the effective execution pla	**Static Analysis**: - The syntax tree is input to the static analysis phase	Step2:- elements are extracted from element.dbx file
	Three functions are used for it. They are: **XSCAN, XISCAN, and XANOR.**	**Run-time analysis**: - The process is divided into two phases 'dynamic context processing'and 'dynamic evaluation'.	Step 3:- Different nodes are joined by using 'ancestor-decendent path-join' algorithm.
	DB2 uses the **XSCAN** operator to traverse the XML document trees and if needed, to evaluate predicates and extract document fragments and values.	**The dynamic context processing** :- It process the memory requirement for query evaluation.	
	XISCAN operator performs lookups or scans on XML indexes.	**The dynamic context processing** :- It process the memory requirement for query evaluation.	
	The **XANDOR** operator evaluates two or more equality predicates simultaneously, by driving multiple XISCANs.		