A TABLE-BASED CLUSTERING STRATEGY FOR XML DATA STORAGE AND QUERYING

NATTAPONG THAWORNKOOL

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2005**

ISBN 974-04-6337-1

Thesis
Entitled

A TABLE-BASED CLUSTERING STRATEGY FOR XML DATA STORAGE AND QUERYING

…….
Mr. Nattapong Thawornkool
Candidate

…….
Asst. Prof. Charnyote Pluempitiwiriyawej, Ph.D.
Major-Advisor

…….
Assoc. Prof. Damras Wongsawang, Ph.D.
Co-Advisor

…….
Assoc. Prof. Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies

…….
Assoc. Prof. Supachai Tangwongsan, Ph.D.
Chair
Master of Science Programme in Computer Sciece
Faculty of Science

Thesis
Entitled

A TABLE-BASED CLUSTERING STRATEGY FOR XML DATA STORAGE AND QUERYING

was submitted to the Faculty of Graduate Studies, Mahidol University
for the degree of Master of Science (Computer Science)

on

20 May, 2005

…….
Mr. Nattapong Thawornkool
Candidate

……
Asst. Prof. Charnyote Pluempitiwiriyawej, Ph.D.
Chair

……
Assoc. Prof. Damras Wongsawang, Ph.D.
Member

……
Lect. Songsri Tangsripairoj, Ph.D.
Member

……
Lect. Songpol Ongwattanakul , Ph.D.
Member

……
Assoc. Prof. Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies
Mahidol University

……
Prof. Amaret Bhumiratana, Ph.D.
Dean
Faculty of Science
Mahidol University

ACKNOWLEDGEMENT

I would like to express my sincere gratitude and deep appreciation to my advisor, Dr.Charnyote Pluempitiwiriyawej, for his guidance, invaluable advice, supports, supervision and continued encouragement which enables me to complete this thesis successfully.

I am very grateful to my reviewing committee member, Dr.Damras Wongsawang, Dr. Songsri Tangsripairoj, and Dr. Songpol Ongwattanakul for their constructive guidance and suggestions.

I wish to thank to the Department of Computer Science of Mahidol University for providing the facilities and library, and other graduated friends for their valuable advice and kindness during my study.

Finally, I am grateful to my family for their love, encouragement, moral support and being my willpower to study and finish my thesis.

Nattapong Thawornkool

A TABLE-BASED CLUSTERING STRATEGY FOR XML DATA STORAGE AND QUERYING

NATTAPONG THAWORNKOOL 4337388 SCCS/M

M.Sc. (COMPUTER SCIENCE) MAJOR IN COMPUTER SCIENCE

THESIS ADVISORS : CHARNYOTE PLUEMPITIWIRIYAWEJ, Ph.D,
DAMRAS WONGSAWANG, Ph.D.

ABSTRACT

XML, a data format, has become the de facto standard for interchanging data among sources. As the processing of XML data is increasing, the requirement of efficient XML data storage and querying is apparent. Native and non-native XML storage are two aspects of storing XML data that facilitate efficient XML query processing. The element-based and the subtree-based clustering strategies are used to cluster XML data in the native storage with respect to the type of XML data elements and the sub-tree structure, respectively.

In this thesis, a new *table-based clustering strategy* to support efficient native storage and querying of XML data is described. The strategy combines element-based clustering and the subtree-based clustering. It categorizes the XML data with respect to the type of data elements and classifies the tree-like structure of XML data regarding the cardinalities of relationships between the data elements. The clustering technique stores XML data into a set of tables. With table-based clustering, the size of XML files can be reduced by half. Additionally, table-based clustering has been evaluated with a good performance for complex queries.

KEY WORDS : EXTENSIBLE MARKUP LANGUAGE / DATA CLUSTERING /
XML DATA STORAGE AND QUERYING /
TABLE-BASED CLUSTERING STRATEGY

79 P. ISBN 974-04-6337-1

กลยุทธ์การจัดกลุ่มข้อมูลแบบเทเบิลเบสสำหรับการจัดเก็บและการสอบถามข้อมูลเอกซ์เอ็มแอล
(A TABLE-BASED CLUSTERING STRATEGY FOR XML DATA STORAGE AND QUERYING)

ณัฐพงศ์ ถาวรกูล 4337388 SCCS/M

วท.ม. (วิทยาการคอมพิวเตอร์)

คณะกรรมการควบคุมวิทยานิพนธ์ : ชาญยศ ปลื้มปิติวิริยะเวช,Ph.D, ดำรัส วงศ์สว่าง,Ph.D.

บทคัดย่อ

เอกซ์เอ็มแอลกลายเป็นสื่อกลางในการแลกเปลี่ยนและรับส่งข้อมูลระหว่างแหล่งข้อมูลต่างๆ เมื่อการประมวลผลข้อมูลเอกซ์เอ็มแอลเพิ่มมากขึ้น ความจำเป็นในการจัดเก็บและการสอบถามข้อมูลเอกซ์เอ็มแอลอย่างมีประสิทธิภาพกลายเป็นประเด็นที่ได้รับความสนใจและมีการศึกษากันอย่างกว้างขวาง ซึ่งแบ่งออกได้เป็น 2 แนวทางด้วยกัน คือ นอนเนทีฟ เอกซ์เอ็มแอลและเนทีฟ เอกซ์เอ็ม ในวิทยานิพนธ์นี้ได้มุ่งเน้นไปที่เนทีฟ เอกซ์เอ็มแอลซึ่งสามารถแบ่งกลยุทธ์ตามการจัดกลุ่มข้อมูลได้เป็น 2 แบบคือ เอลิเม็นเบสซึ่งจัดกลุ่มข้อมูล ตามชนิดของเอลิเม็นและ ซับทรีเบส ซึ่งจะทำการจัดกลุ่มข้อมูลตามความสัมพันธ์ของเอลิเม็นซึ่งเกิดจากการแตกทรีออกเป็นทรีย่อยๆ

วิทยานิพนธ์นี้อธิบายกลยุทธ์การจัดกลุ่มข้อมูลแบบเทเบิลเบส เพื่อรองรับการจัดเก็บแบบเนทีฟและการสอบถามข้อมูลเอกซ์เอ็มแอล กลยุทธ์ดังกล่าวซึ่งมีลักษณะผสมผสานระหว่าง การจัดกลุ่มแบบเอลิเม็นเบสและซับทรีเบส กลยุทธ์เทเบิลเบสทำการจัดกลุ่มข้อมูลตามชนิดของเอลิเม็นและแบ่งแยกกลุ่มของข้อมูลตามโครงสร้างต้นไม้ ซึ่งแบ่งแยกตามความสัมพันธ์เชิงจำนวนระหว่างเอลิเม็นและจะทำการจัดเก็บข้อมูลเอกซ์เอ็มแอลโดยใช้โครงสร้างแบบตาราง จากผลการทดลอง พบว่ากลยุทธ์การจัดเก็บแบบเทเบิลเบสสามารถลดขนาดเอกสารเอกซ์เอ็มแอลได้ประมาณครึ่งหนึ่ง นอกจากนั้นการสอบถามข้อมูลเอกซ์เอ็มแอลทำได้อย่างมีประสิทธิภาพโดยเฉพาะการสอบถามข้อมูลที่เกี่ยวกับเอลิเม็นหลายๆชนิด

79 หน้า ISBN 974-04-6337-1

CONTENTS

		Page
ACKNOWLEDGEMENTS		iii
ABSTRACT		iv
LIST OF TABLES		ix
LIST OF FIGURES		x
CHAPTER		
I	**INTRODUCTION**	1
	1.1 Problem Statement	1
	1.2 Goal of The Thesis	2
	1.3 Thesis Organization	3
II	**BACKGROUND INFORMATION ON XML**	4
	2.1 General Information about XML	4
	2.2 Aspects of Storing XML Data	5
	2.3 Schema Description	6
	2.3.1 DTD	6
	2.3.2 XML Schema	7
	2.4 XML Parsers	9
	2.4.1 DOM Parser	9
	2.4.2 SAX Parser	9
III	**RELATED RESEARCH**	11
	3.1 Non-Native XML	12
	3.1.1 The Edge Approach	12
	3.1.2 The Monet Approach	13
	3.1.3 The DTD Driven Approach	14
	3.2 Native XML	15
	3.2.1 Lore Approach	15
	3.2.2 Timber Approach	16
	3.2.3 NATIX Approach	17

CONTENTS (Cont.)

		Page
	3.2.4 Orientstore Approach	18
	3.3 Comparative Study of XML Storage Strategies	19
IV	**TABLE-BASED CLUSTERING STRATEGY**	21
	4.1 Structure and Node Relationship Information Tree (SNRIT)	24
	4.2 Schema Mapping	26
	4.3 Clustering Contents	30
	4.4 Summary	31
V	**IMPLEMENTATION OF TABLE-BASED CLUSTERING ENGINE**	32
	5.1 SNRIT Creator	33
	5.2 Cluster-table Configuration Generator	37
	5.3 Node Configuration Generator	41
	5.4 Cluster-Content Creator	43
	5.5 Data Storage Repository	45
	5.6 Loader	46
	5.7 Query Parser	46
	5.8 Query Processor	47
	5.9 Summary	50
VI	**EXPERIMENTS**	52
	6.1 Description of the Test Data	52
	6.2 Description of the Test Query	53
	6.3 Measurement	53
	6.4 Clustering Size Comparison	54
	6.5 Downsizing Ratio Comparison	55
	6.6 Clustering Time Comparison	56
	6.7 Querying Time Comparison	57
	6.8 Reconstruction Time Comparison	58
	6.9 Summary	59
VII	**CONCLUSION AND FUTURE WORK**	60

CONTENTS (Cont.)

Page

REFERENCES 62

APPENDIX 64

BIOGRAPHY 79

LIST OF TABLES

	Page
Table 6.1 The data sets information	52
Table 6.2 Query Group Overview	53
Table 6.3 Comparative clustering size experiment	54

LIST OF FIGURES

		Page
Figure 2.1	Sample XML document showing basic components of XML.	4
Figure 2.2	Sample XML document contain the structured data	5
Figure 2.3	Sample XML document contain the mixed contents	6
Figure 2.4	DTD example	7
Figure 2.5	XML Schema example	8
Figure 2.6	DOM tree example	9
Figure 2.7	Behavior of SAX parser	10
Figure 3.1	(a) Sample XML document “movies.xml” and (b) its graph representation	11
Figure 3.2	(a) Edge table and (b) data table for movies.xml	13
Figure 3.3	Monet tables for movies.xml	14
Figure 3.4	DTD driven approach relational schema mapping	14
Figure 3.5	OEM data model representing movie items in movies.xml	15
Figure 3.6	TIMBER data model representing movies.xml	17
Figure 3.7	NATIX data model representing movies.xml	18
Figure 3.8	Orientstore data model representing movies.xml	19
Figure 4.1	Example XML document “movies.xml ” contains movie list	21
Figure 4.2	Example of storing XML document into table storage	22
Figure 4.3	Table-based clustering approach	23
Figure 4.4	SNRIT example	24
Figure 4.5	A table represented SNRIT	26
Figure 4.6	The mapping process snapshot	27
Figure 4.7	Cluster table example	28
Figure 4.8	Node Configuration example	29
Figure 4.9	Cluster Table and contents example	30
Figure 5.1	The system architecture and its main components	32
Figure 5.2	SNRIT Creator components	33

LIST OF FIGURES (Cont.)

	Page
Figure 5.3 XML document tree example	33
Figure 5.4 XML structure tree example	34
Figure 5.5 Range of Node generation algorithm pseudo code	35
Figure 5.6 The augmented structure tree with the ranges	36
Figure 5.7 The augmented structure tree with the frequencies	36
Figure 5.8 Cluster-table configuration generator component	37
Figure 5.9 A tabular data represented SNRIT	38
Figure 5.10 Grouping node algorithm	39
Figure 5.11 Grouping node example	40
Figure 5.12 Cluster Table example	41
Figure 5.13 Node configuration generator component	42
Figure 5.14 Node configuration example	43
Figure 5.15 Cluster-Content Creator component	44
Figure 5.16 Storage XML document example	45
Figure 5.17 Parse tree example	47
Figure 5.18 Query Processor component	47
Figure 5.19 The query operation example	48
Figure 5.20 Data configuration example	49
Figure 5.21 Construct tree example	50
Figure 5.22 Query result example	50
Figure 6.1 Chart of downsizing ratio comparison including its data	55
Figure 6.2 Chart of clustering time comparison including its data	56
Figure 6.3 Chart an querying time comparison including its data	57
Figure 6.4 Chart of reconstruction Time Comparison	58

CHAPTER I
INTRODUCTION

1.1 Problem Statement

On the Internet, the Extensible Markup Language (XML [1]) has become the standard for interchanging data among varieties of sources. XML provides a simple data format which is machine-readable. Moreover, it can capture data semantics and represent them in a human-understandable format. The advancement of the Internet makes the processing of XML data important. As the processing of XML data is increasing, the requirement of efficient XML data storage and querying is apparent.

In XML, both data and its structure are represented as text and stored into a document file. The structure is represented in form of attributes and element tags which can be nested. To process and query XML data, the entire document needs to be parsed and the data must be memory-resident. Loading the entire document into the memory may be impractical while loading parts of document may slow down query performance when having periodic multiple access of data items.

Two aspects of storing XML data to facilitate efficient query processing have been discussed. One is the Non-native XML. Another is the native XML.

In Non-Native XML [6, 7, 8, 9] aspect, a conventional DBMS is used to manage and manipulate XML data. Thus, only the data with rigid structure can be stored. The XML data needs to be transformed regarding the data model used in the DBMS. The relational DBMS is mature and widely used. With this aspect, the DBMS can utilize small disk spaces and provide efficient query processing [10]. However, it has to deal with the transformation overheads of query and those of data. If the relational DBMS is used, the XML queries need to be transformed into SQL queries. Then, the data is accessed through relational operations and transformed into XML.

The most expensive relational operation is the JOIN operation which is applied to capture the parent-child relationship between XML data elements.

In Native XML [15, 16, 17, 18] aspect, a specific data management system is used to manage and manipulate XML data. Hence, the data with flexible structure can be stored. With this aspect, the data is clustered and stored according to its internal structure. The query processing is relatively efficient since the transformation overheads of query and those of data are small. In this aspect, two clustering strategies have been discussed: Element-based clustering and Subtree-based clustering.

In element-based clustering strategy, XML data is grouped with respect to its type. For each element type, a chunk of storage spaces is allocated. This strategy performs fast for processing of node queries which access multiple elements of a particular type. However, it is time-consuming for processing of path queries which access multiple types of element according to path expression.

In subtree-based clustering strategy, XML data is grouped with respect to its tree like structure. A tree is used to represent data in an XML document. Each node represents an element or an attribute. The tree is partitioned into one or more sub-trees. Each sub-tree may contain different types of elements. The elements in the sub-tree are stored in a particular chunk of disk space. A connector between parent and child sub-trees needs to be maintained in order to rebuild the original tree. This strategy performs fast for path queries but it consumes time for processing of node queries.

1.2 Goal of The Thesis

This thesis describes a new *table-based clustering strategy* to support efficient storage and querying of XML data. Our strategy combines the element-based clustering and the subtree-based clustering. It categorizes the XML data with respect to the types of data elements and classifies the schema tree regarding to the cardinalities of relationships between the data elements. Our clustering technique stores XML data into a set of tables. Data elements in the same cluster support fast

access for simple queries. Data elements from different clusters can be accessed from the enclosed paths which stored in one of the column of the table.

Our clustering strategy requires the *Structure and Node Relationship Information tree* (*SNRIT*) which represents element-nesting structure and ancestor-descendant relationship of elements. Our SNRIT plays an important role in the schema mapping process. It is useful to help the system to capture the structure of XML document.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2: Provides background information about XML

Chapter 3: Overviews the research that is relevant to this thesis.

Chapter 4: Describes the table-based clustering strategy and introduces the Structure and Node Relationship Information Tree which supports efficient XML data storage and querying.

Chapter 5: Focuses on the implementation of the table-based clustering engine.

Chapter 6: Provides the evaluation result of the table-based strategy which respect to the element-based and the subtree-based strategies.

Chapter 7: Concludes the thesis with a summary of accomplishment and an outline of issues to be considered in the future.

CHAPTER II
BACKGROUND INFORMATION ON XML

This chapter provides readers information on XML. It gives the general information about XML, the aspects of storing the XML datas, the schema description, and the XML parsers.

2.1 General Information about XML

Extensible Markup Language (XML) [1] has become a standard for marking up data in a structured document. Its specification has been published in February 1998. XML document is only a plain text document with its nested and self-describing structure that provides a human-understandable and machine-readable data format. Currently, it has become the standard for representing and exchanging data format. It consists of element tags, attribute name, and contents. The element tags and attribute names are used to represent semantic information of data contents.

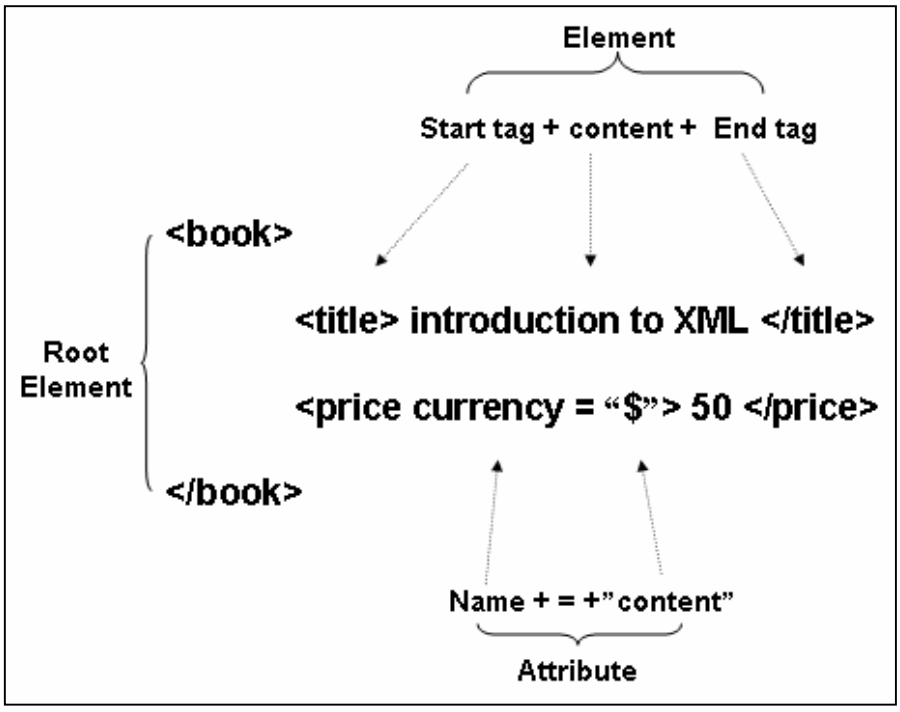


Figure 2.1: Snapshot of XML document showing basic components of XML

As show in Figure 2.1, XML document consists of two parts of information: the content and the markup. The content part is the text data area whereas the markup part represents the semantic of the content. The semantic of the content can be in the form of element or attribute. The element consists of a sequence of a start tag, a content, and an end tag. The start tag consists of a sequence of '<', element title, and '>' such as <book>, <title>, <price>. The end tag consists of a sequence of '</', element title, '>' such as </book>, </title>, </price>. The text data located between the start tag and the end tag is content. The attribute consists of a sequence of attribute name, '=', '"', content, '"' such as 'currency="$"'. The attribute has to be enclosed into the start tag of element. Conceptually, an XML document contains nested elements which form a hierarchical structure. The element at the top level is the root. Each XML document must have only one root.

2.2 Aspects of Storing XML Data

XML has widely used in a variety of application domains such as bioinformatic, supply-chain, and medical information. The information is normally stored in an XML document. Storing data in and getting data from XML documents has two aspects: the document-centric and the data-centric.

…<university> Mahidol University </university> is one of the most prestigious universities in <country>Thailand</country>, recognized for its outstanding achievements in teaching, research and professional services…….....

Figure 2.2: Snapshot of XML document in document-centric aspect

In *document-centric aspect*, the information stored in the document has more flexible structure in the sense that the document may contain a bunch of mixed contents in which XML element tags are embedded in large text fragments to represent semantic of some keywords. The document-centric aspect is usually used for human consumption. It has been appeared in many application domains such as email, advertisements, and PDF document. Figure 2.2 shows a snapshot of an XML document. We can see that the element tags <university> and <country> are

embedded in the large text fragment. They represent the semantic of “Mahidol University” and “Thailand” respectively.

```
<university>
   <name>Mahidol</name>
   <address>
      <street>112 Rama Six</street>
      <province>Bangkok</province>
      <zip>10123</zip>
   </address>
</university>
```

Figure 2.3 : Snapshot of XML document in data-centric aspect

In *data-centric aspect*, the information stored in the document has a fairly regular structure and has no mixed contents. It is usually used for machine consumption. The data-centric aspect has been appeared in many application domains such as sales orders, flight schedules, and scientific data. Figure 2.3 shows a snapshot of a XML document which contains a university database. The record represents the information about Mahidol University which located at 112 Rama six streets Bangkok 10123. In this thesis, we are focusing on storing and querying XML data with the data-centric aspect.

2.3 Schema Description

XML document contains information which is classified as semi-structured data which has a flexible structure. The information which describes the structure is called schema description. It is used to describe the document structure with a list of legal elements and attributes. It can be declared inline in the document or in an external file. Two approaches of declaring a schema for XML documents are Document Type Definition (DTD) and XML Schema.

2.3.1 DTD

DTD [5] is used to describe structure of XML document. It contains a set of rules which defines elements, attributes, and relationships between elements. In the

DTD, the elements and attributes are described with the declarations. The form of declarations is:

<!KEYWORD ELEMENT_NAME LIST_OF_ PARAMETER>

The KEYWORD is reserved for either ELEMENT or ATTRIBUTE. The list of the parameters is defined depending on KEYWORD. If the ELEMENT keyword is used, the parameter is either pattern or data type. If the ATTLIST keyword is used, the attribute name must follow the element name which the attribute is associated with. The rest of parameters are formed as a sequence of data type, and a constraint.

```
<!ELEMENT book (title, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA #REQUIRED>
```

Figure 2.4: DTD example

Figure 2.4 shows an example of DTD. It describes the fact that an XML document consists of three element types: book, title, and price. The element of type book contains only one element of type title followed by only one element of type price. The element of type title and that of type price contain information of type #PCDATA (parse character data). The element of type price carries one attribute whose name is currency. Such attribute has type of CDATA (character data). The keyword #REQUIRED indicates that the attribute currency is mandatory.

2.3.2 XML Schema

Like DTD, XML Schema [12, 13, 14] describes the structure of the XML document. A main difference between the DTD and the XML Schema is that XML Schema uses XML syntax whereas DTD does not. The XML Schema specification offers everything a DTD can do. Moreover, It supports namespaces and richer data types. XML Schema supports custom data types and seven primitive data types which are string, boolean, decimal, float, double, time, and date.

XML Schema needs to be defined apart from XML document whereas DTD can be internally or externally defined. The document structure consists of a variety of

elements (e.g. element, complexType, and simpleType) that determine the appearance of elements and their content in XML documents. The simpleType identifies the element that can contain text only whereas the complexType identifies the element that contains text, sub-elements, attributes, or their combination. Each element must have a prefix which references to the XML Schema namespace. Conventionally, the prefix xsd: references to "http://www.w3.org/2001/XMLSchema" which appears in the schema element. However, any prefix can be used to identify the elements and the data types as belonging to the vocabulary of any XML Schema language.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="book">
  <xsd:complexType>
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="price">
    <xsd:complexType>
     <xsd:simpleContent>
      <xsd:extension base="xsd:string">
       <xsd:attribute name="currency" type="xsd:string"/>
      </xsd:extension>
     </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Figure 2.5: XML Schema example

Figure 2.5 shows a snapshot of XML Schema file that is equivalent to the DTD file in Figure 2.3. The book element is a complex element which contains a sequence of a title and a price element. Whereas the title element is simple element that has type of string, the price element becomes complex element. It has type of string. Additional, the price element carries the currency attribute that has type of string.

In this research, we use schema information as a hint to help fragment and cluster XML data. The schema information can be pre-defined in DTD or XML Schema files. If the schema information is not pre-defined, it can be directly extracted from XML document.

2.4 XML Parsers

Since data in XML documents is text, it needs to be parsed before being accessed. Parsing the document requires XML Parsers which can be classified into two types: DOM (Document Object Model) and SAX (Simple API for XML).

2.4.1 DOM Parser

DOM [2] is a hierarchical structure which represents data in XML documents. To access data in XML document, DOM parser parses once the XML document and builds up the DOM tree in main memory. Having received request, it navigates along the DOM tree and returns matched data.

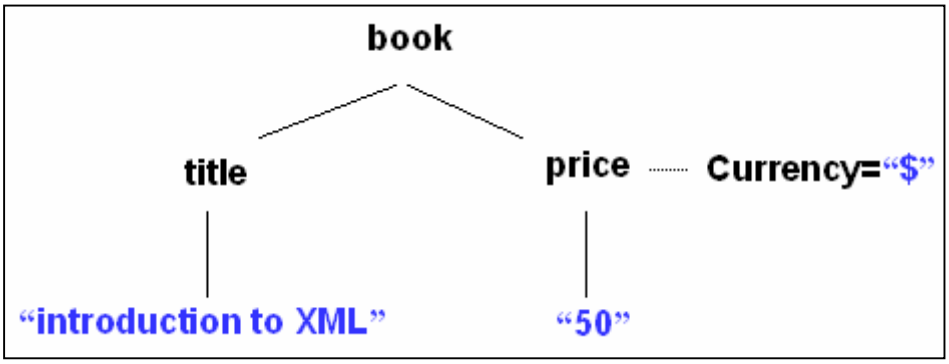


Figure 2.6: DOM tree example

Figure 2.6 show a DOM tree which represents the XML document in Figure 2.1. The tree consists of a set of nodes. Each node represents an element or data content such as book, title, and "introduction to XML" which is data content of title element. The attribute is stored together with its data as a unit and is carried by its element holder. The price element carries a currency attribute.

2.4.2 SAX Parser

SAX [3] is a simple API for parsing XML documents. It provides classes and interfaces for event-based XML parsing. SAX has been widely used in Java Programming community. Parsing the XML document, SAX parser reads data items and notifies the event handler to respond the parsing events. The three main events are the start of element, the end of element, and the character data.

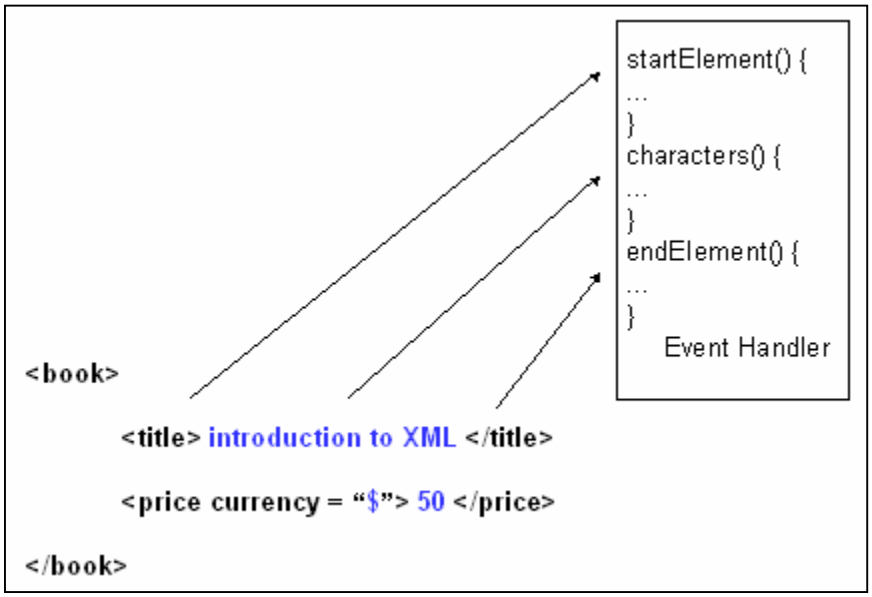


Figure 2.7: Behavior of SAX parser

Figure 2.7 shows how to trigger the event handler. Assuming that the parser parses the document through element tag <title>, it triggers the event handler. The event handler launch the function startElement() to response the request. Then, the event handler launch the function characters() to response the "introduction to XML" data content and launch the function endElement() to response the end tag </titile>.

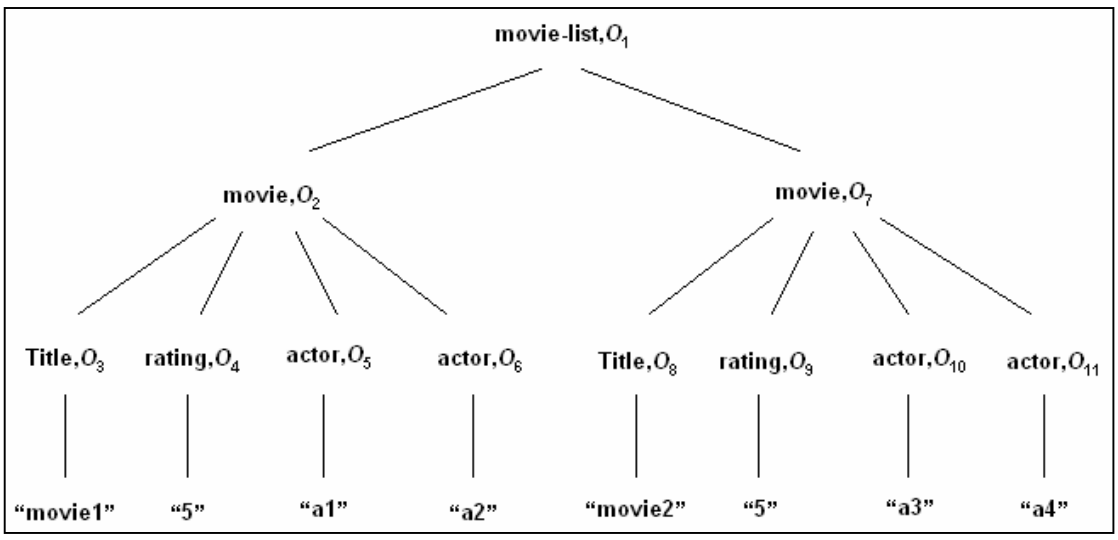
One of the issues that we need to consider is when the SAX parser and the DOM parser should be used. It has been suggested that the DOM parser should be used when the XML documents (e.g. email, e-newspaper) are document-centric whereas the SAX parser should be used when the XML documents (e.g. sale order, medical information) are data-centric. In this thesis, since we are focusing on the data-centric XML documents, the SAX parser is used.

CHAPTER III
RELATED RESEARCH

This chapter provides readers information on the research works that are relevant to this thesis. The relevant research works can be classified into two categories: Non-Native XML and Native XML. Since our research works are focusing on clustering and querying XML data from Native XML storage, the existing clustering strategies are further investigated and discussed.

```
<movie-list>
    <movie>
        <title> movie1 </title>
        <rating> 5 </rating>
        <actor>a1</actor>
        <actor>a2</actor>
    </movie>
    <movie>
        <title> movie2 </title>
        <rating> 4 </rating>
        <actor>a3</actor>
        <actor>a4</actor>
    </movie>
</movie-list>
```

a)



b)

Figure 3.1: (a) Sample XML document "movies.xml" and (b) its graph representation

To illustrate how XML data is actually stored with either the Non-Native or the Native XML storages, we use the XML document “movie.xml” shown in Figure 3.1 throughout this chapter. An XML document can be represented as a directed graph that consists of a set of nodes and a set of edges. Nodes are used to represent XML elements and attributes whereas edges are used to represent parent-children links between elements. Every node is assigned an identification number. The identification number starts with *O* character and follows with generated numeric value in the dept-first search manner. The data contents are embedded in double quote and linked to its parent.

3.1 Non-Native XML

In Non-Native XML strategy [6, 7, 8, 9], XML documents are stored in a DBMS data model, which supports only structured data. Two types of DBMS are the relational database management system (RDBMS) and the object oriented database management system (OODBMS). The RDBMS is mature and widely used. It stores and maps XML data contents to relational tables. An advantage of RDBMS is that it takes low space and efficient in query processing. Unfortunately, RDBMS rely on SQL that the parent-children relationship between elements is managed by SQL joins. This produces very complex SQL query and adds overhead involving many joins for complex path expressions. Three approaches have been applied for the Non-Native XML strategy: the edge approach, the Monet approach, and the DTD driven approach.

3.1.1 The Edge Approach

The Edge Approach [8] fragments the directed graph of an XML document and organizes those fragments into two tables - the edge table and the data table-- as show in Figure 3.2. Each record in the edge table corresponds to each edge in the directed graph. It consists of the node ID, the position (POS) of the node with respect to its sibling, the node label (LAB), the node type (TYPE), and the parent node (PAR_ID). The node type indicates whether the node is a reference node (REF) or a cdata node (CDATA). The reference node refers to the node with no data. The cdata node refers to the node with a data item which is stored in a text node. All text nodes

are classified and separated into another table callesd as data table. Node id is used to link data with its holder. This strategy provides the benefit that sub-elements of one XML element are stored close to each other. The drawback is that elements with the same tag name are not placed close together. Consequently, queries such as “find out the number of movies” will incur a large number of I/Os.

ID	POS	LAB	TYPE	PAR_ID
O_1	1	movie-list	REF	O_0
O_2	1	movie	REF	O_1
O_3	1	title	CDATA	O_2
O_4	2	rating	CDATA	O_2
O_5	3	actor	CDATA	O_2
O_6	4	actor	CDATA	O_2
O_7	2	movie	REF	O_1
O_8	1	title	CDATA	O_7
O_9	2	rating	CDATA	O_7
O_{10}	3	actor	CDATA	O_7
O_{11}	4	actor	CDATA	O_7

a)

ID	DATA
O_3	movie1
O_4	5
O_5	a1
O_6	a2
O_8	movie2
O_9	4
O_{10}	a3
O_{11}	a4

b)

Figure 3.2: (a) Edge table and (b) data table for movies.xml

3.1.2 The Monet Approach

The Monet approach [7] fragments the XML documents and groups the XML data into semantically homogeneous units based on the binary association. There are three association types: parent-children, node-attribute, and node-value. The parent-children association is an edge that links between element nodes. The node-value association is an edge that links between an element and a child text node. The node-attribute association is an edge that links between an element node and an attribute value node. Each association is represented as a binary table as show in Figure 3.3. Each table is made up of two columns. For the parent-children and the node-attribute associations, the two columns are the ID which represents an element or an attribute, and the PAR_ID which refers to the parent of the current node. For node-value association, the two columns are the ID and the DATA which represents the data content. The Monet approach provides the benefit that elements with the same tag name are placed close together. However, using the Monet approach may end up with a large number of tables.

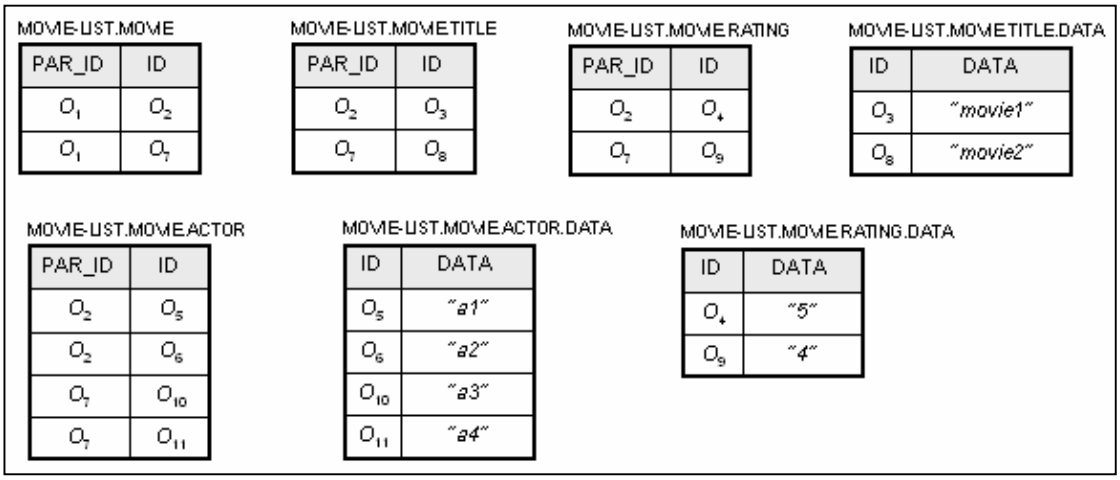


Figure 3.3 : Monet tables for movies.xml

3.1.3 The DTD Driven Approach

The DTD driven approach [9] uses the schema-mapping technique. It requires the existence of DTD. The big challenge is deciding when an element is grouped into its parent (known as inlining method) and when a separated table must be created. A separated table is used to capture the parent-children relationship between an element and a set of child elements with the same tag. Each record in a table is assigned an identification number (ID) and contains a parentID column to identify its parent. An element which can appear only once in its parent is inlined as a column of the table representing its parent.

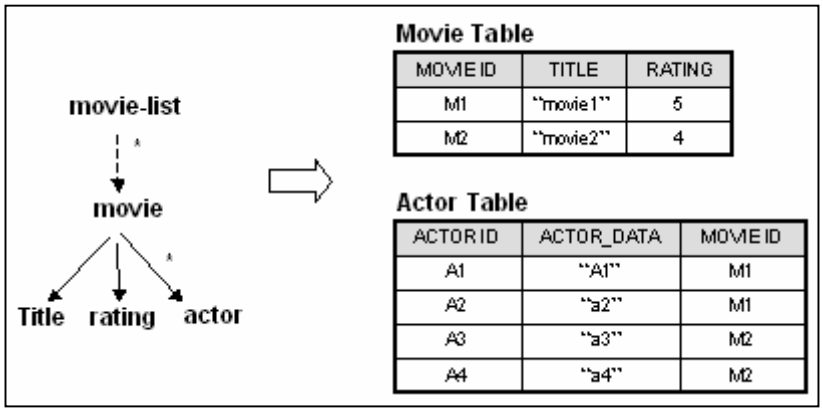


Figure 3.4: DTD driven approach relational schema mapping

Figure 3.4 shows the relational tables which are transformed from “movies.xml” document. The left of the figure displays the DTD tree. It shows

structure of “movies.xml” document. The “*” is used to represent the cardinality relationship between elements which means the parent can have the child with the same tag name from 0 to indefinition. The right of the figure illustrates a pair of tables. The Movie table contains information about movies such as title and rating. The Actor table contains information about actors who plays in a particular movie.

3.2 Native XML

Native XML [15, 16, 17, 18] requires a data management system which is specifically designed for storing and querying of a set of XML documents. It fragments XML document and groups XML data into its internal structure. Four approaches have been applied for the Native XML strategy: the Lore approach, the TIMBER approach, the NATIX approach, and the Orientestore approach.

3.2.1 Lore Approach

Lore [18] (Lightweight Object Repository) uses Object Exchange Model (OEM) to represent data in XML documents. It reserves the nested structure of the XML document using labeled directed graph. In the OEM, the elements and attributes are looked alike objects.

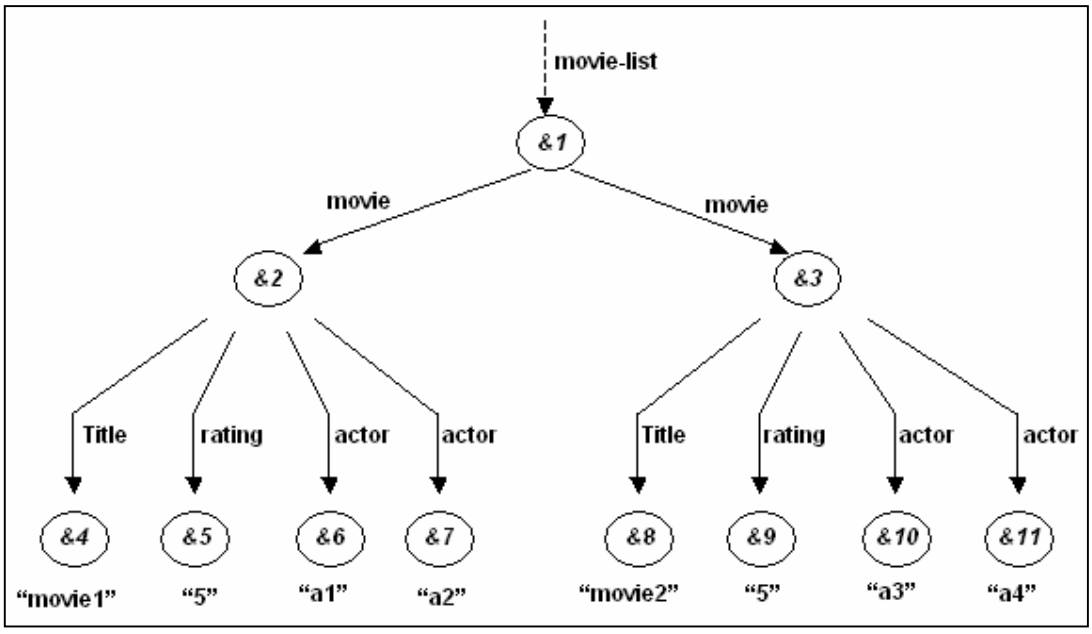


Figure 3.5: OEM data model representing movie items in movies.xml

Figure 3.5 shows an OEM graph which represents information about movies in “movies.xml” document. The nodes in the graph are objects. Each object is assigned an object identifier (oid) such as &1, &2 and so on. The object can be classified into two types: simple and complex. The simple objects (e.g. objects with id &4, &5, and &6) can only contain data. The complex objects (e.g. objects with id &1, &2, and &3) contain named links to other OEM objects. The named link is a special label that is used to identify the class of object (element type). For example, Object &2 is complex and it link to the sub-objects &4, &5, &6, and &7. Object &4 is simple and contains data "movie1". To store OEM objects into storage, each object is stored as a single record. Each record consists of oid, object type, object value, and its link information. Lore stored the records into storage based on depth-first search order.

3.2.2 TIMBER Approach

The TIMBER [15] is developed at the University of Michigan. It is built on top of SHORE (responsible for disk management). Like Lore, TIMBER uses tree structure to represent XML data. A node represents an element or a set of attributes. An element which contains one or more sub-elements is represented as an internal node. The element content is represented as a leaf node. All attributes associated with a particular element are stored together into a single node and linked to the element node. Each node has a node identifier. The node identifier consists of three fields <start, end, level> based on inclusion relationship between an element and its sub-elements. The inclusion relationship means every descendant has the pair field <start, end> inner the pair field of its ancestor. Each node is stored as a single record. All records are stored together in the storage. And they are stored in depth-first search manner.

Figure 3.6 shows TIMBER data model which represents movies appeared in “movies.xml” document. The nodes movie-list, movie, title, rating, and actor represent element whereas the other nodes (“movie1”,”5”, and so on) represent data content. For example, the movie-list node has the node identification as <1, 30, 1 >. It contains two-child movie nodes which have the node identification as <2, 15, 2 > and <16, 29, 2 > respectively. Considering inclusion relationship between movie-list node and the first movie node, the pair field <start, end> of the movie node is equal to <2 ,15

> and the pair field of movie-list node is equal to <1 ,30 >, the movie node is descendant of movie-list node because the pair field <2 ,15 > is inner the pair field <1 ,30 >.

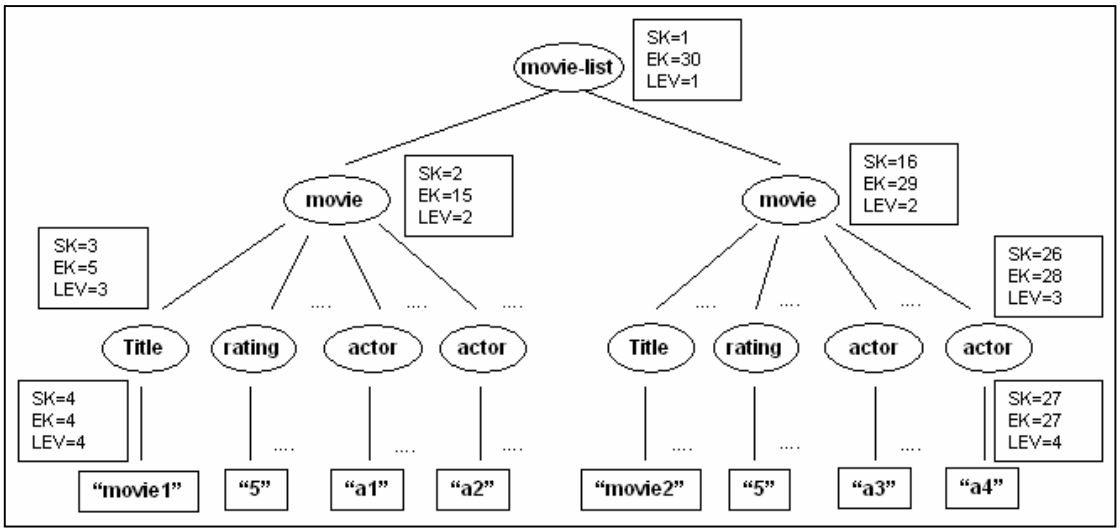


Figure 3.6 : TIMBER data model representing movies.xml

3.2.3 NATIX Approach

The NATIX [16] is a system developed at the University of Mannheim. Like Lore and TIMBER, NATIX uses tree structure to represent XML data. NATIX mainly focus on a mechanism for fragmenting and distributing the data tree into the other page since the whole data tree may not fit in a single physical page on disk storage. In NATIX, a tree of XML data is partitioned into several units. The size of each unit must be no larger than the size of disk page which can be as large as 32K. The tree partitioning can be described as follows: First, allocates a disk page for the whole tree. If the tree cannot fit in a single page, the tree is partitioned into two sub-trees. Then, if there is an existent tree that cannot fit in a single page, do the tree partitioning until every sub-trees can fit in a page. Finally, to reconstruct the original tree, the connection link needs to be maintained. There are two connectors reside separated in parent and sub-tree. The connector resides in parent tree called as Proxy whereas the connector resides in sub-tree called as Hub. The connector is a node also. Substituting all proxies by their respective sub-trees reconstructs the original data tree.

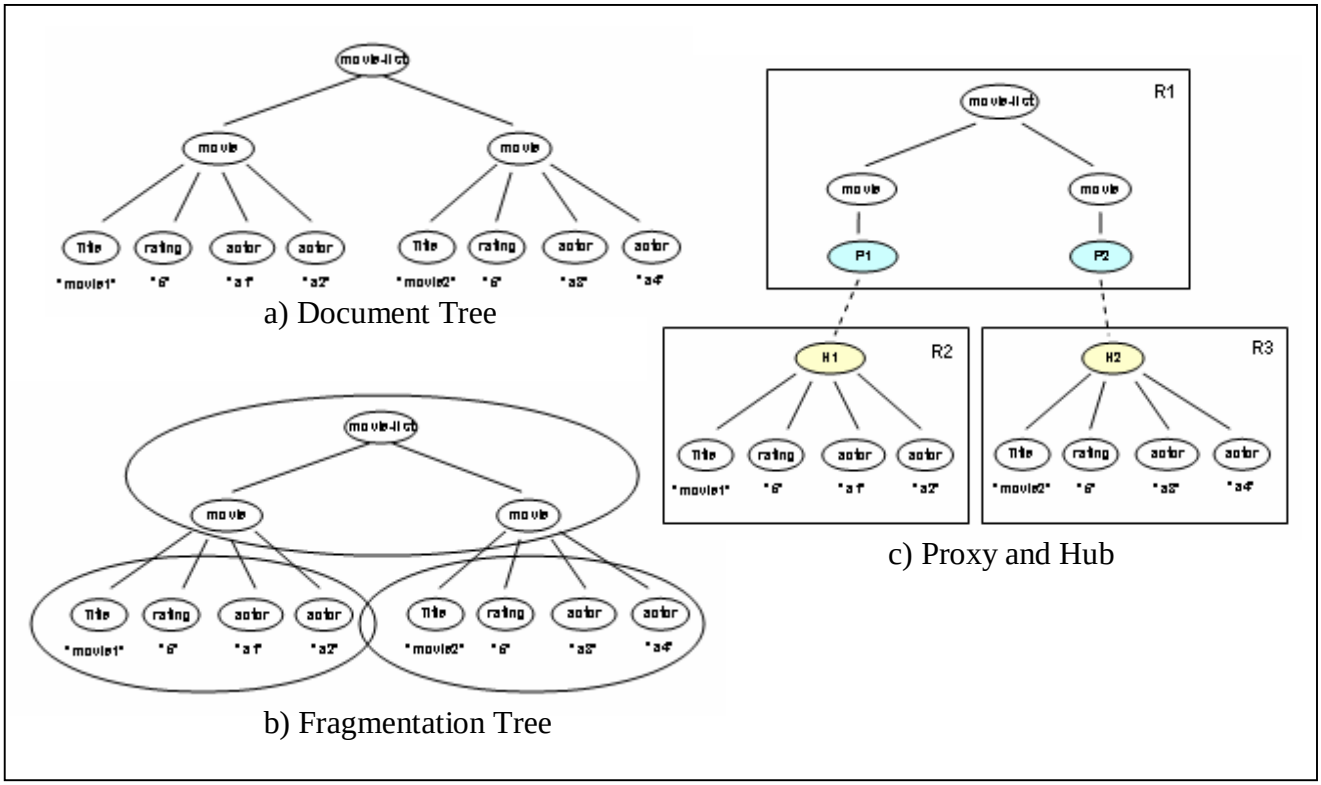


Figure 3.7 : **NATIX data model representing movies.xml**

Figure 3.7 shows NATIX data model representing “movies.xml” document. Assuming that, the whole data tree is partitioned into three sub-trees. The sub-tree in page R1 contains movie-list nodes, movie whereas the other pages contain the childe nodes of each movie. Figure 3.7c shows relationships between proxy and hub node.

3.2.4 Orientstore Approach

Like other approaches, the OrientStore [17] uses tree structure to represent XML data. OrientStore introduce two strategies for storing the XML documents. First strategy is called as Element-Based Clustering (EBC). EBC stored XML data based on element type. The element node and its own attribute and text nodes are stored altogether as a single unit. All units that conform to the same element type are placed close together. Last strategy is called as Logical Partition-Based Clustering (LPC). LPC partitiones the data tree and stores each sub-tree base on the semantic block. The semantic block is used to describe a relatively integrated logical unit. All instances of the same semantic block are placed close together.

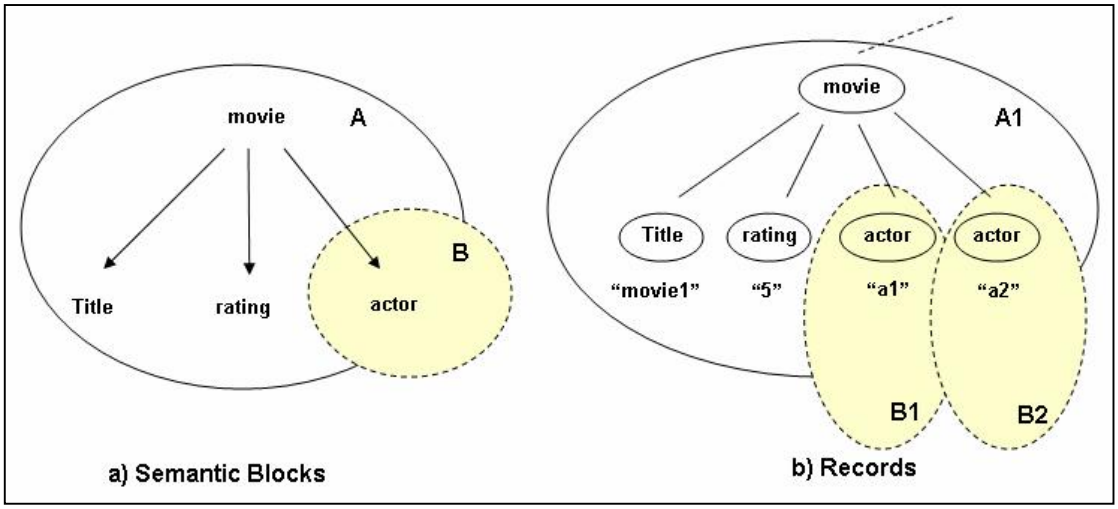


Figure 3.8 : **Orientstore data model representing movies.xml**

Figure 3.8 shows Orientstore data model which represents logical units of XML data in "movies.xml" document. Assuming that the document is divided into two semantic blocks: A and B. The semantic block A consists of element types movie, title, and rating whereas the semantic block B consists of only element type actor. In Figure 3.8b the records A1 is instance of the semantic block A whereas the record B1 and B2 are instances of the semantic block B. The record B1 and B2 are stored together in a physical page whereas the record A1 is separated in another page.

3.3 Comparative Study of XML Storage Strategies

In this chapter, we have described about a variety of strategies for storing and querying XML data. They are classified into two strategies: Non-Native XML and Native XML. Non-Native XML is a good choice for performance consideration. However, it has tradeoff with a lot of cost for DBMS installation and maintenance. Besides, it needs expert who has skill both XML technology and DBMS. Consequence, Native XML is became the interested strategy. There are two mainly classified strategies for Native XML.

The first one is element-based clustering, [Lore, TIMBER, OrientStore(EBC)]. It fragments the XML document and groups the XML data dependently based on the element type. Each element that conforms to the same element type is placed close together. The element-based clustering reaches good performance in a simple query

For example, finding the number of the movies; it is able to answer the question by retrieving only the element type movie without involving other unnecessary element types; thus reducing the number of I/Os. Unfortunately, for a large XML document, it can result in a large number of tables.

The last one is subtree-based clustering, [NATIX, OrientStore(LPC)]. It fragments the XML document into the logical units. The data tree is partitioned into several sub-trees based on logical unit. Each sub-tree that is instance of the same logical unit is placed close together. The subtree-based clustering reaches good performances in a complex query which relates about reconstruction a huge of parent-children links. Because most of parent and child nodes locate in the same cluster which incurs fewer I/Os for such query. However, for a simple query, it incurs lots of I/Os because the nodes with the same tag name are not placed close together.

CHAPTER IV
TABLE-BASED CLUSTERING STRATEGY

The underlying format of storing XML documents has a significant impact on efficiency of query processing. To improve a query response time, a simple way is to reorganize and downsize XML documents which are verbose in nature. We have developed a new table-based clustering strategy to reorganize XML data for fast access. Our table-based clustering supports Native XML storage. It combines the element-based and the subtree-based clustering strategies.

```
<movie-list>
 <movie>
  <title>movie1</title>
  <rating>5</rating>
  <actor a_name="a1">
        <award year="1998">award1</award>
  </actor>
  <actor a_name="a2">
        <award year="1998">award2</award>
  </actor>
 </movie>
 <movie>
  <title>movie2</title>
  <rating>4</rating>
  <actor a_name="a3">
        <award year="1996">award1</award>
        <award year="1997">award3</award>
  </actor>
 </movie>
 <movie>
  <title>movie3</title>
  <rating>3</rating>
  <actor a_name="a4">
        <award year="2000">award2</award>
  </actor>
  <actor a_name="a5"/>
 </movie>
</movie-list>
```

Figure 4.1 : Example XML document “movies.xml ” contains movie list

Figure 4.1 shows a simple XML document which is used to combine a scenario throughout this and next chapters. The document “movies.xml” represents a movie list which contains information about the movie including the title of movie, the rating of movie, the actor name, the award title which she won and the year when she won the award. The movie, the title, the rating, the actor and the award are represented as XML elements. The name and year awarded of the actor are represented as XML attributes.

In our table-based clustering, data in the XML document is fragmented with respect to element types. These element types are then grouped according to their structural constraints. The elements that conform to the same element type are stored together into the same table.

Movie

ID	TITLE	RATING
1	movie1	5
2	movie2	4
3	movie3	3

Actor

ID	A_NAME
1.1	a1
1.2	a2
2.1	a3
3.1	a4
3.2	a5

Award

ID	AWARD	YEAR
1.1.1	award1	1998
1.2.1	award2	1998
2.1.1	award1	1996
2.1.2	award3	1997
3.1.1	award2	2000

Figure 4.2 : Example of storing XML document into table storage

Figure 4.2 shows the table storage of XML document containing the movies list. The information about the movies is partitioned into three pieces. Each of which is represented as a table. The three tables are Movie, Actor, and Award. Every table has one column reserved for link construction among tables. Such column is called ID and is located at the first column of each table. The Movie table consists of two more columns including the Title column which is allocated for the title of movie and the Rating column which is allocated for the rating of movie. The Actor table consists of

one more columns, namely A_name which is allocated for the name of actor. The Award table consists of two more columns including the Award column which is allocated for the award name and the Year column which is allocated for the year that the actor won award.

Our table-based clustering requires SNRIT which is a structure tree that represents XML document structure. In XML, the document structures are defined by XML schema. Normally, XML schema is predefined and stored in a separated XML document such as DTD, XML Schema. However, the schema information can directly extract from XML document.

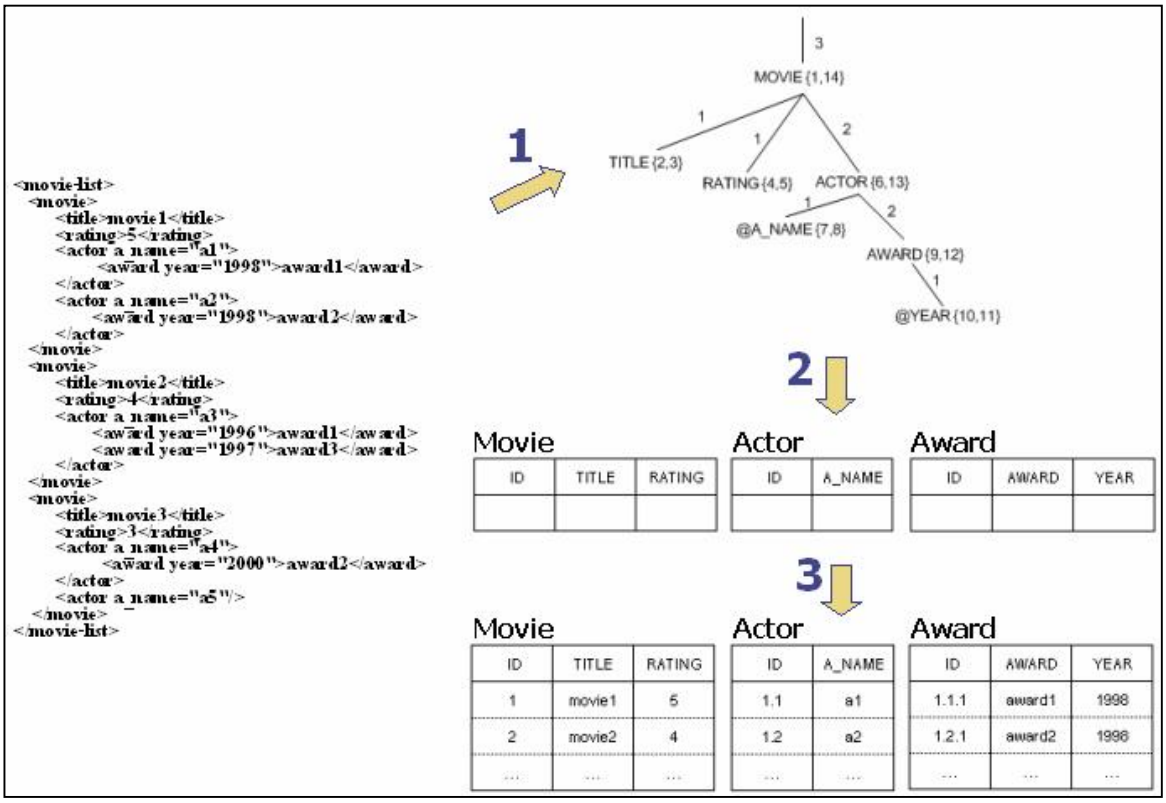
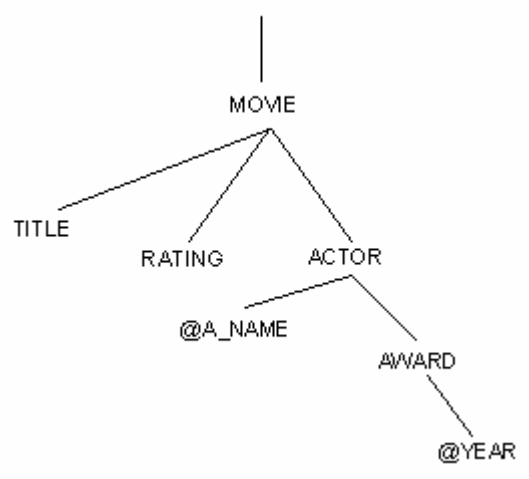


Figure 4.3 : Table-based clustering approach

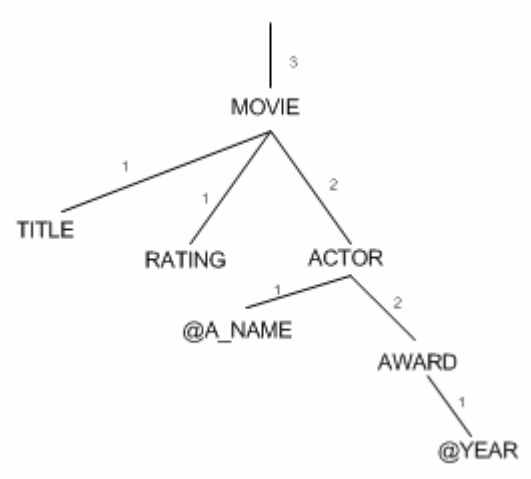
Figure 4.3 shows the table-based clustering process which consists of three steps. First, the SNRIT is created. Second, the schema mapping which maps the SNRIT and the table structure is performed. Finally, the XML elements that conform to a particular element type are placed into an appropriate table.

4.1 Structure and Node Relationship Information Tree (SNRIT)

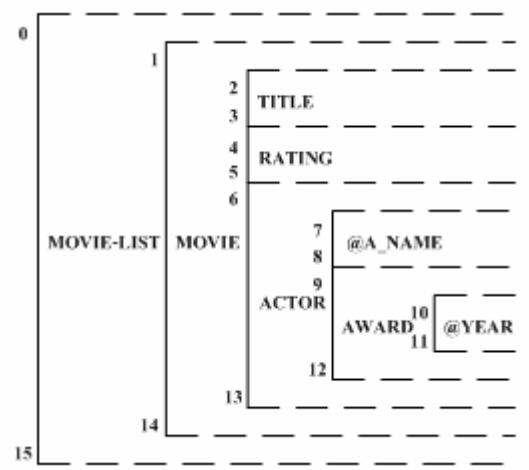
The Structure and Node Relationship Information Tree (SNRIT) is an extension of the XML structure tree. It represents element-nesting structure and ancestor-descendant relationships. The SNRIT plays an important role in the schema mapping process. It is useful to capture the structure of XML document.



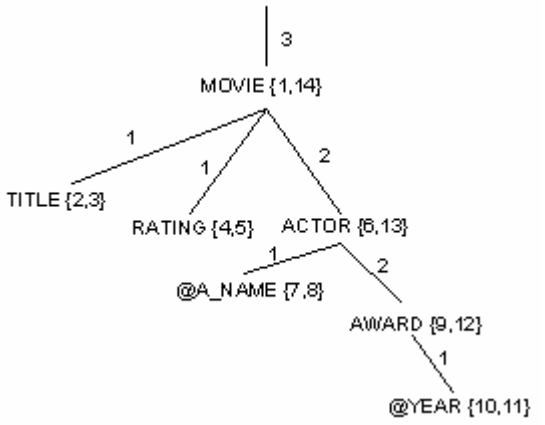
b) Structure Tree



c) Structure Tree with freequency



d) Node Range



e) SNRIT

Figure 4.4 : SNRIT example

Figure 4.4a shows XML structure tree which represents the structure of XML document “movies.xml”. When the XML document is parsed, The XML structure tree can be obtained. The structure tree consists of a set of nodes and a set of edges.

Each node represents an element type which is the substitution for element instances that conform to the same path. In Figure 4.4b, we omit the movie-list element

because it is root. The movie-list element contains three movie sub-elements (as show in Figure 4.1). All of three movie elements can form to the same path which is movie-list -> movie. Consequence, the element type movie (as show in Figure 4.4a) substitutes the three instances of movie element. It is represented with node in XML structure tree.

Each edge represents parent-children link between elements. If an element carries an attribute, the edge will represent the attribute as a child node of the element node. As show in Figure 4.1, the attribute a_name is carried by the actor element. In the structure tree(as show in Figure 4.4a), the a_name node is represented as child node of the actor node. The attribute node must be located at the left most child of the element node.

SNRIT uses the XML structure tree as basis. In SNRIT, each edge is associated with maximum occurrences of a particular type of element that conform to the same path. We briefly called it as *frequency* whose data type is numeric. For example, there are three movie elements in data source (as show in Figure 4.1). We can see that the numbers of occurrence of actor element with respect to the three movies are 2, 1, and 2. As a result, the frequency value that is associated with the edge that represents movie-actor link is equal to 2 (as show in Figure 4.4b). The frequency is useful for the schema mapping process which is discussed in section 4.2.

In addition to augmenting edge with frequency, SNRIT augments node with a range. The range consists of a pair of numeric-value fields <Start, End>. The Start and the End values are obtained from the dept-first search on the structure tree. The Start value is always lower than the End value. Every descendant has strictly a range inner the range of its ancestor. At the root, the Start value is set to 0. The Start value of descendant is always higher than the Start value of its ancestor. The End value of descendant is always lower than the End value of its ancestor. For example, as show in Figure 4.4c, movie-list node is associated with the <0, 15>. The movie node which is a child of movie-list node is associated with the <1, 14> and the actor node which is ancestor of movie-list node is associated with the <6, 13>. We refer to the range for the ancestor as the super range and refer to the nearest super range as scope. The scope of

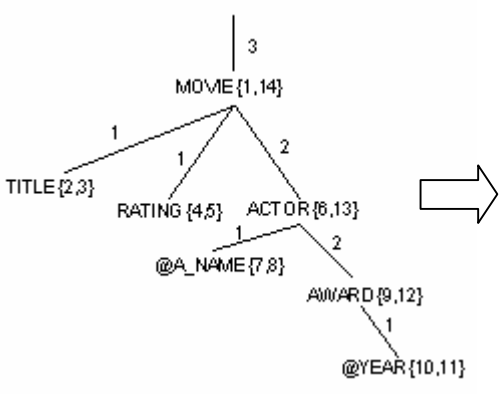
node is simply the range of its parent node. The SNRIT for the XML document about movies is shown in Figure 4.4d.

4.2 Schema Mapping

As already discussed, table-based clustering strategy uses table structure as data model. The cluster table needs configuration including defining table schema, relationship and data constraints.

To transform XML document tree to tables, it requires mapping process. The mapping process captures the structure from semi-structure of XML document and generates a cluster-table configuration.

XML documents may have the schema definition. If the schema definition is not presented, it needs to be extracted from XML document. Then, the schema definition is transformed into SNRIT. Our SNRIT is useful to help the mapping process to know how to fragment the XML document and regroup the fragmented units resulted.



NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

Figure 4.5 : A table represented SNRIT

Figure 4.5 illustrates the result of transforming the SNRIT into a table. The table consists of four columns including node labels (NODE), start range (START_R), end range (END_R), and frequency (FREQUENCY) which are directly mapped from the SNRIT respectively.

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3

a)

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1

b)

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1

NODE	START_R	END_R	FREQUENCY
ACTOR	6	13	2

c)

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1

NODE	START_R	END_R	FREQUENCY
ACTOR	6	13	2
A_NAME	7	8	1

NODE	START_R	END_R	FREQUENCY
AWARD	9	12	2
YEAR	10	11	1

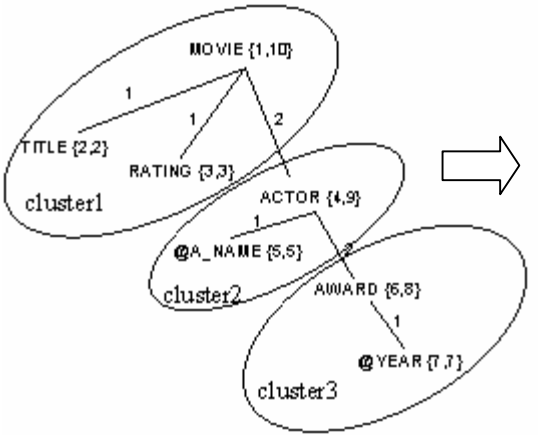
d)

Figure 4.6 : The mapping process snapshot

The mapping process navigates along the SNRIT in dept-first search manner. Note that for each a pair of parent-child node, the parent node is always located above child node in table. For example, for the movie-title pair, the movie node is located at the first and the title node is located at the second row. The mapping process mainly focuses on grouping. The main principle of grouping is the following:

1. Process one record at a time from top to bottom.
2. For each record, if frequency is more than 1, a new group is created and that node is appended into the new group.
3. If frequency is equal to 1, that node is appended into the group that its parent is member.

Figure 4.6 shows the steps of mapping process. In Figure 4.6a, showing the process at the movie record. Observe that the frequency is more than 1. Therefore, a new group is created. In Figure 4.6b, showing the process at the title record. Observe that the frequency is equal to 1. Therefore, it is appended into the group that its parent is member. In Figure 4.6c, showing the process at the actor record. Observe that the frequency is more than 1. Therefore, it is stored into a new separated group. In Figure 4.6d, showing the finish of process. There are three groups of node resulted. After that, these groups of nodes are mapped to cluster-table schema. The cluster-table configurations are shown in Figure 4.7.



id	NODE	START_R	END_R
1	MOVIE, TITLE, RATING	1	14
2	ACTOR, A_NAME	6	13
3	AWARD, YEAR	9	12

Figure 4.7 : Cluster table example

Figure 4.7 shows the cluster-table configuration. The document “movies.xml” is fragmented into three cluster tables. The first cluster table group three element types including Movie, Title, and Rating. The second cluster table group two element types including Actor, and A_name. The last cluster table group two element types including Award, and Year.

After XML structure tree are mapped to the cluster tables, each node is distributed along the cluster tables. As we have ever discussed, the table is the structured data storage. It means that each cluster table needs to define configuration for its member. The node configuration contains information about location (order), data type, and constraints. In this research, we concentrate only on the location.

NODE	TYPE	START_R	END_R	TABLE_ ID	COLUMN_ NO
MOVIE	ELEMENT	1	14	1	0
TITLE	ELEMENT	2	3	1	1
RATING	ELEMENT	4	5	1	2
ACTOR	ELEMENT	6	13	2	0
A_NAME	ATTRIBUTE	7	8	2	1
AWARD	ELEMENT	9	12	3	1
YEAR	ATTRIBUTE	10	11	3	2

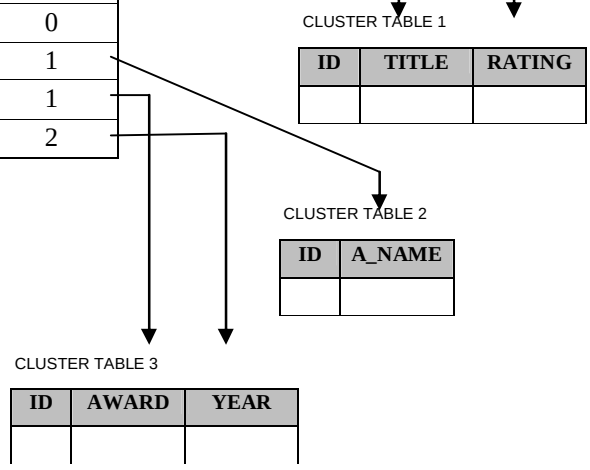


Figure 4.8 : Node Configuration example

Figure 4.8 shows node configuration which agrees with the cluster-table configuration in Figure 4.7. The node configuration consists of six fields. The first four fields are directly extracted from the SNRIT including node label (NODE), node type (TYPE), start range (START_R) and end range (END_R). There are two additional fields: TABLE_ID and COLUMN_NO which indicate a particular cluster table in which the node is a member and the number of column in the cluster table, respectively.

If the COLUMN_NO is 0, it means that the node instance contains no data content, but a group of related data that are represented as nested elements. Otherwise, it means that the node instance contain data content. The positive numeric value is used to identify location or order of space that node occupies in the cluster table. As show in Figure 4.8, for example, there are three nodes as the members of the cluster table 1. Nevertheless, only the title and the rating node are allocated space.

4.3 Clustering Contents

After the cluster-table structure has been created, the XML document has to be parsed. Whenever the clustering process encounters an element tag or an attribute name, the data content will be extracted and placed into an appropriate table with respect to the clustering constraints.

```
<movie-list>
 <movie> //1
  <title>movie1</title>
  <rating>5</rating>
  <actor a_name="a1">                          //1.1
        <award year="1998">award1</award>  //1.1.1
  </actor>
  <actor a_name="a2">                          //1.2
        <award year="1998">award2</award>  //1.2.1
  </actor>
 </movie>
 <movie> //2
  <title>movie2</title>
  <rating>4</rating>
  <actor a_name="a3">                          //2.1
        <award year="1996">award1</award>  //2.1.1
        <award year="1997">award3</award>  //2.1.2
  </actor>
 </movie>
 <movie> //3
  <title>movie3</title>
  <rating>3</rating>
  <actor a_name="a4">                          //3.1
        <award year="2000">award2</award>  //3.1.1
  </actor>
  <actor a_name="a5"/>                         //3.2
 </movie>
</movie-list>
```

CLUSTER TABLE 1

ID	TITLE	RATING
1	movie1	5
2	movie2	4
3	movie3	3

CLUSTER TABLE 2

ID	A_NAME
1.1	a1
1.2	a2
2.1	a3
3.1	a4
3.2	a5

CLUSTER TABLE 3

ID	AWARD	YEAR
1.1.1	award1	1998
1.2.1	award2	1998
2.1.1	award1	1996
2.1.2	award3	1997
3.1.1	award2	2000

Figure 4.9 : Cluster Table and contents example

Figure 4.9 shows the result of clustering “movies.xml” document. Three records are added into the cluster table 1. Five records are added into the cluster table 2. Five records are added into the cluster table 3. Whenever a new record is added, the record id will be generated. The record id contains the enclosed path which is the information about the path summary of related cluster tables. The enclosed path is

useful for reconstruction parent-children links. The enclosed path consists of a parent_path_id and a current_node_id.

parent_path_id + "." + current_node_id

The parent_path_id describes an inclusion relationship between the records in two cluster tables, one of which is a descendant of another. The record ID of the descendant has to enclose the recode ID of its ancestor. As show in Figure 4.9, the record at row 1 of cluster table 3 has inclusion relationship with the record at row 1 of cluster table 2 and it has inclusion relationship with the record at row 1 of cluster table 1. The current_node_id is a sequential generated number which is used to identify the record with respect to its parent.

4.4 Summary

In this chapter, we have developed the table-based clustering strategy. The approach consists of three main steps. First, obtains SNRIT by parsing XML document or the schema description files. Then, transform SNRIT to a set of tables using the schema mapping. Finally, consider data contents and put it into appropriate tables with respect to the clustering constraints.

CHAPTER V

IMPLEMENTATION OF TABLE-BASED CLUSTERING ENGINE

In this chapter, we describe the architecture of table-based clustering engine which supports XML storage and querying. Figure 5.1 shows the architecture of table-based clustering engine. It consists of seven components: SNRIT creator, cluster-table configuration creator, node configuration creator, cluster-content creator, document loader, query parser and query processor. The first four components support clustering and storing XML data. The last three components support querying the XML data. All components are implemented in Java (J2SE 1.4.2) environment. The functionalities of each component will be described in each section.

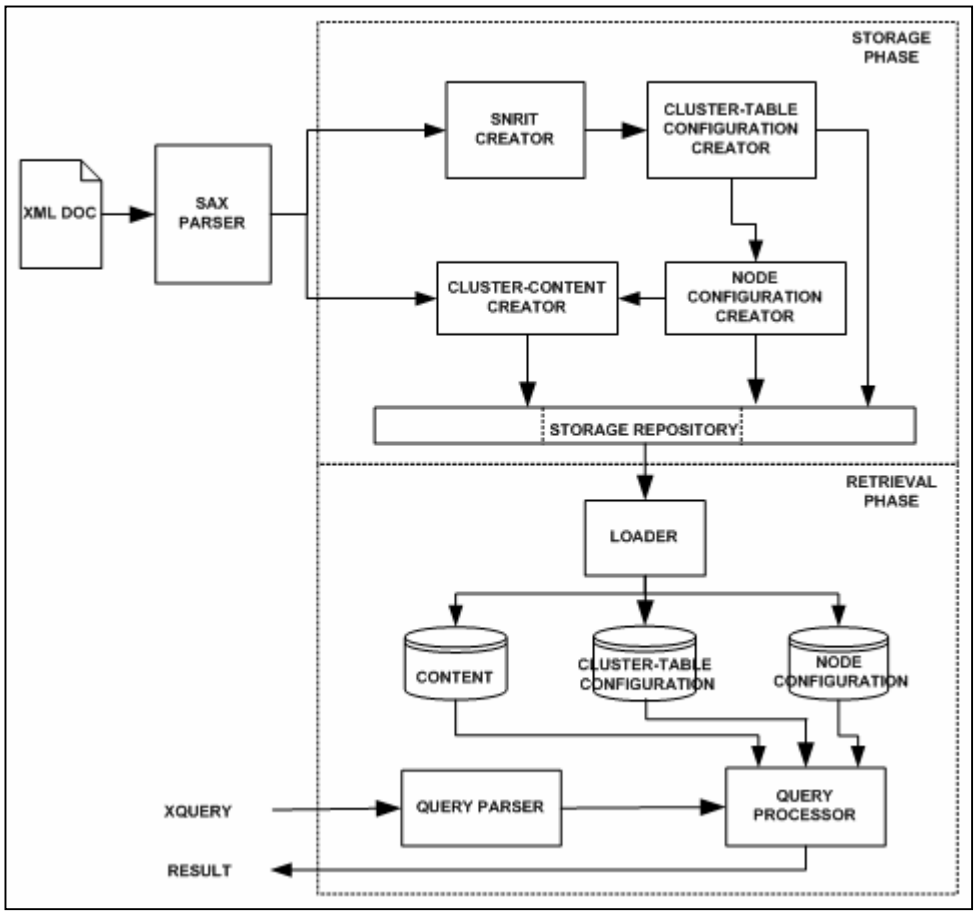


Figure 5.1 : The system architecture and its main components

5.1 SNRIT Creator

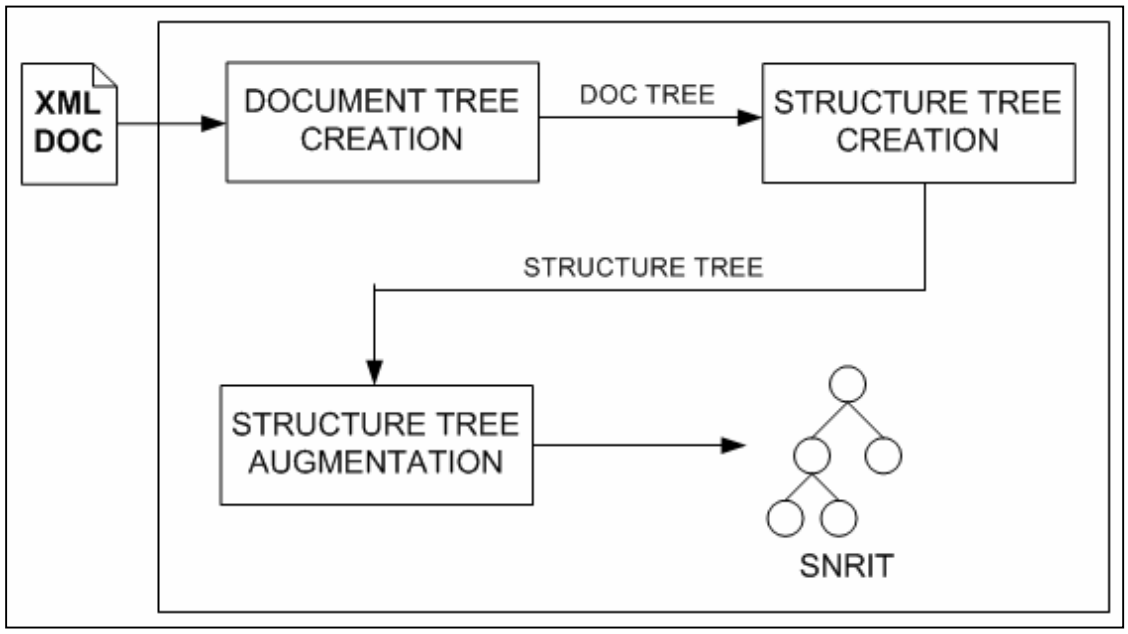


Figure 5.2 : SNRIT Creator components

The SNRIT creator takes as input an XML document and produces as output the SNRIT which is derived from the tree. Figure 5.2 shows the main three modules inside the SNRIT creator: document tree creation, structure tree creation and structure tree augmentation.

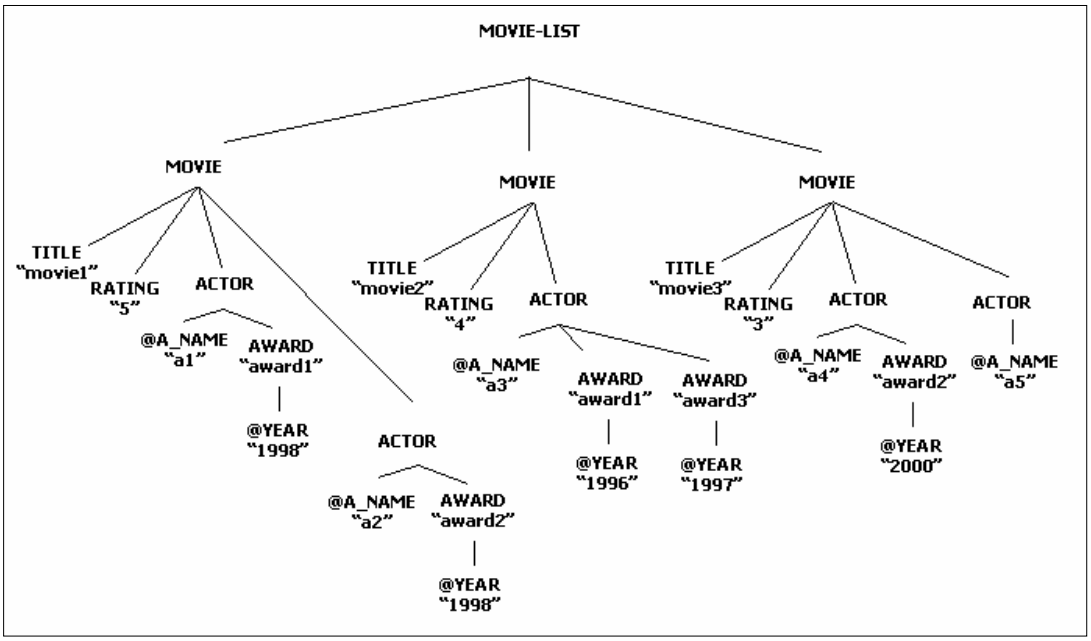


Figure 5.3 : XML document tree example

In the document tree creation module, the given XML document (Figure 4.1 page 21) is parsed and a corresponding XML document tree (Figure 5.3) is generated. To do so, the SAX parser or the DOM parser can be used. Figure 5.3 shows XML document tree. It consists of a set of nodes and a set of edges. Node represents element tag or attribute name. If node has data content, its data is enclosed with in double quote and located under node label. Edge represents parent-children link between elements.

In the structure tree creation module, the document tree is considered and a structure tree that describes relationship between element types is build. The nodes which represent a set of elements of the same type causes adding a node in the structure tree. Note that if XML Schema or DTD can be obtainable, structure tree can be derived from the schema description. As show in Figure 5.4, the document contains eight element types including Movie-List, Movie, Title, Rating, Actor, A_name, Award, and Year. Every node of document tree has to conform to element type.

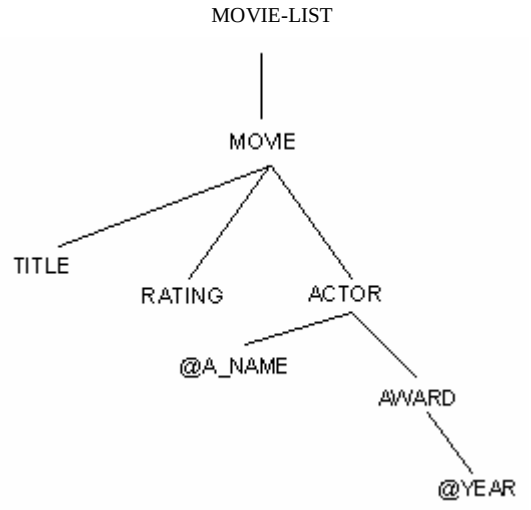


Figure 5.4 : XML structure tree example

In the structure tree augmentation module, the SNRIT is derived from the structure tree by augmenting each node with a range values. The Range is used to identify ancestor-descendant relationship between nodes. The range value that the child node holds must be sub-range of the range that the parent node holds.

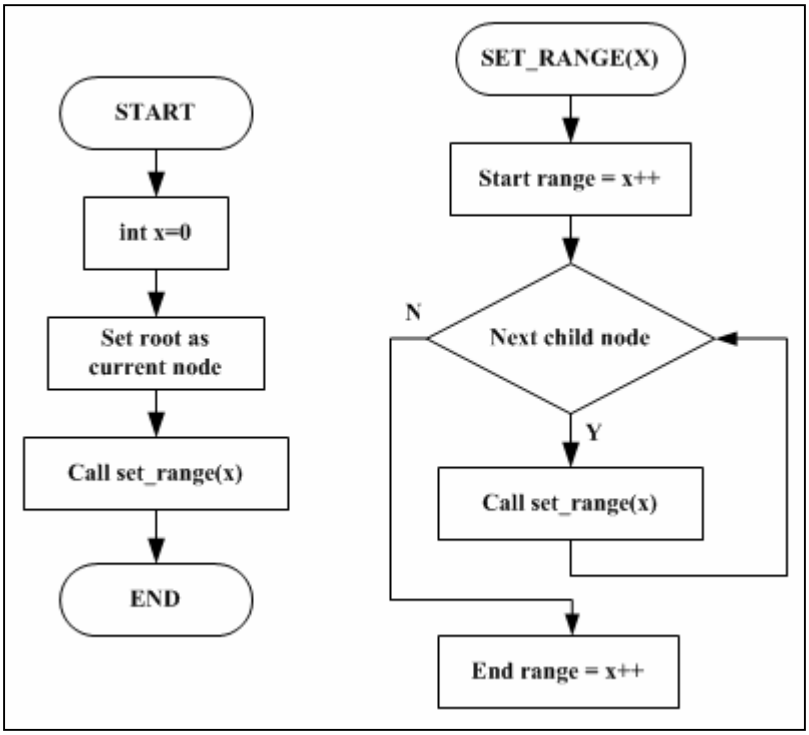
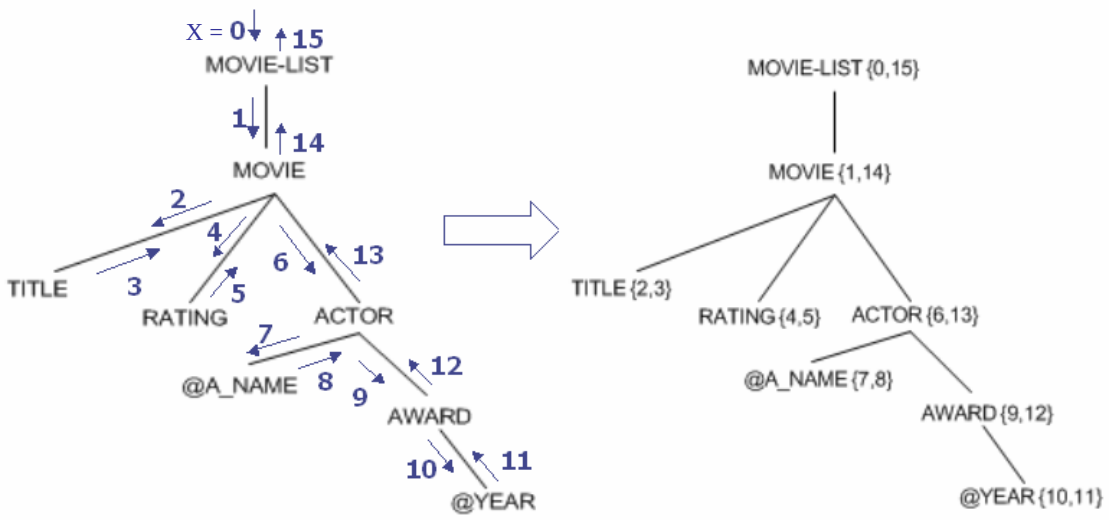


Figure 5.5 : Range of Node generation algorithm pseudo code

Figure 5.5 describes the algorithm for augmentating the range on each node. Initially, we set up a variable called *x* and set its value to 0. We process one node at a time from top to bottom. Every time process was passed to next node, the value of variable *x* is increased by 1. In case that node has child nodes, it will pass process to child node from left to right. After finished process of each child node, the process was passed back to its parent. The value of variable *x* that is received from its parent is assigned to the start range of the node. At the same time, the value of variable *x* that is passed back to its parent is assigned to the end range of the node.

Figure 5.6 shows the traversing along the structure tree and how to obtain the range value of each node. For example, movie-list node receives value 0 from variable *x*. Then, the movie-list passes value 1 to the child node which is movie node. Then, movie node passes value 2 to its first child which is title node. Because of the title node no has one child, the title node passes value 3 back to its parent node which is movie node. Consequence, the value that title node received is assigned to start range and the value that title node passed back to its parent node is assigned to end range. The range of title node is <2, 3>.

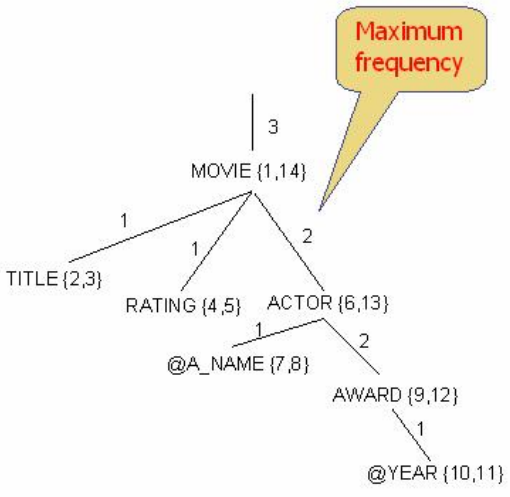


a) The steps of traversal on the structure tree **b) The structure tree is augmented with the ranges**

Figure 5.6 : The augmented structure tree with the ranges

Besides assignment the range on each node, we need to attach frequency on each edge. The frequency is the maximum occurrences of a particular type of element that conform to the same path. Figure 5.7a shows the counting of elements that conform to the path movie-list -> movie -> actor. There are three movie elements in data source. The numbers of occurrence of actor elements are 2, 1, and 2 respectively. Consequence, frequency is equal to 2. Figure 5.7b shows frequency on each edge and the SNRIT.

```
<movie-list>
  <movie>
    <title>movie1</title>
    <rating>5</rating>
  1 <actor a name="a1">
        <award year="1998">award1</award>
    </actor>
  2 <actor a name="a2">
      <award year="1998">award2</award>
    </actor>
  </movie>
  <movie>
    <title>movie2</title>
    <rating>4</rating>
  1 <actor a name="a3">
        <award year="1996">award1</award>
      <award year="1997">award3</award>
    </actor>
  </movie>
  <movie>
    <title>movie3</title>
    <rating>3</rating>
  1 <actor a name="a4">
        <award year="2000">award2</award>
    </actor>
  2 <actor a name="a5"/>
  </movie>
</movie-list>
```



a) Counting elements example **b) The structure tree is augmented with the frequency**

Figure 5.7 : The augmented structure tree with the frequencies

5.2 Cluster-table Configuration Creator

The cluster-table configuration creator takes as input the SNRIT and produces as output cluster-table configuration. Figure 5.8 shows the main four modules inside the cluster-table configuration creator: node selection, node testing, group management, and table-configuration creation.

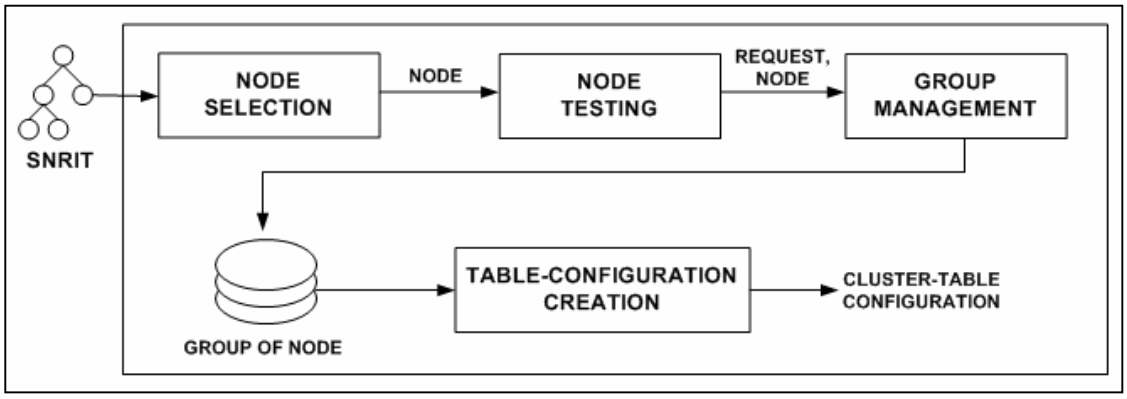


Figure 5.8 : Cluster-table configuration generator component

In the node selection module, the SNRIT is traversed in dept-first search manner to form a node list. Nodes are selected from the last one at a time. Each selected node is fed to the node testing module.

In the node testing module, the decision which group that the node should join is required. The decision is made by using the following criterion:

If frequency of node is equal to 1 :

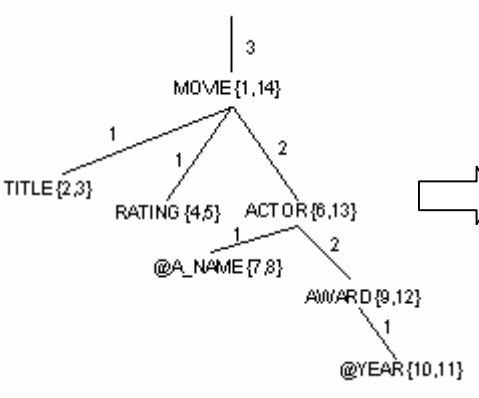
Then it will be added to the group that its parent is member.

Else a new group is created and the node becomes the first member.

Once the decision has been made, the group management module asks for information about the target group if there exists. Otherwise, a new group will be created. Then, the node is appended in the group.

After all nodes are placed into the group, the table-configuration creation module will express the cluster-table configuration which contains information about a logical

storage of XML document. We demonstrate the process of cluster-table configuration generator as following:



NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

Figure 5.9 : A tabular data represented SNRIT

- SNRIT-to-node-list Transformation

We represent the node list with the table as show in Figure 5.9. It consists of four columns including node label (NODE), Start Range (START_R) and End Range (END_R), and Frequency (FREQUENCY) respectively. Each row represents node of SNRIT. To consider two nodes that have ancestor–descendant relationship each other, the row that represents ancestor node locates above the row represented descendant node in table. For example considers the parent-children nodes Movie and Title. The Movie node is above Title node.

- Node grouping

Consider the Frequency of node, if any node has Frequency equal to 1, it shows that there is only one instance of node can occurrence in any instance of its parent. As a result, it is appended to the group that represents its parent. As show in the Figure 5.9, for example, it is noticed that Title node has Frequency equal to 1, it shows that whenever we found instance of Movie node, we could find only one instance of Title node. Consequence, Title node should be grouped into the same group with its parent.

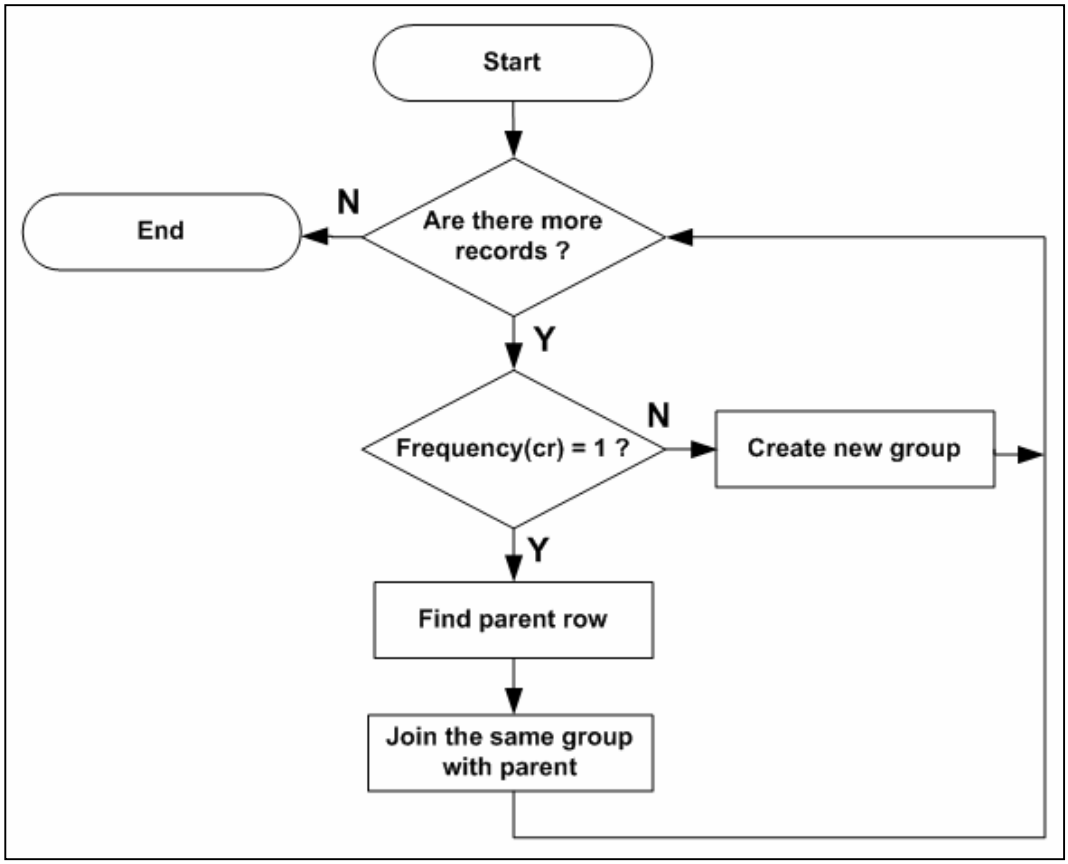


Figure 5.10 : Grouping node algorithm

Let N be the numbers of records in table that represent the node list, and cr be the current record that is being processed. Figure 5.10 shows the node-grouping algorithm. The algorithm processes one record at a time. A group will be created regarding the frequency associated with cr. If the frequency is greater than 1, a new group in which cr is placed will be created. Otherwise, its ancestor will be considered.

Figure 5.11 provides an example of running the node-grouping algorithm on the movie data set. At starting process (the record movie) as show in Figure 5.11a, the frequency is equal to 3. A new group is created and the record movie becomes the first member of group. While processing passes through record rating as show in Figure 5.11b, the frequency is equal to 1. The record rating is to be combined to the same group of its parent. We can find out parent record from record cr-1 to record 1 (candidate parent record). The parent record must valid both two rules as following:

1. Start Range (current record) > Start Range (candidate parent record)
2. End Range (current record) < End Range (candidate parent record)

Considering record title as candidate parent record.

Start Range (record rating) > Start Range (record title) : 4 > 2 valid

End Range (record rating) < End Range (record title) : 5 < 3 invalid

The node title is not parenting of node rating.

Considering record movie as candidate parent record.

Start Range (record rating) > Start Range (record movie) : 4 > 1 valid

End Range (record rating) < End Range (record movie) : 5 < 14 valid

The node movie is parent of node rating.

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3

a)

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1

b)

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1
ACTOR	6	13	2
A_NAME	7	8	1
AWARD	9	12	2
YEAR	10	11	1

NODE	START_R	END_R	FREQUENCY
MOVIE	1	14	3
TITLE	2	3	1
RATING	4	5	1

NODE	START_R	END_R	FREQUENCY
ACTOR	6	13	2
A_NAME	7	8	1

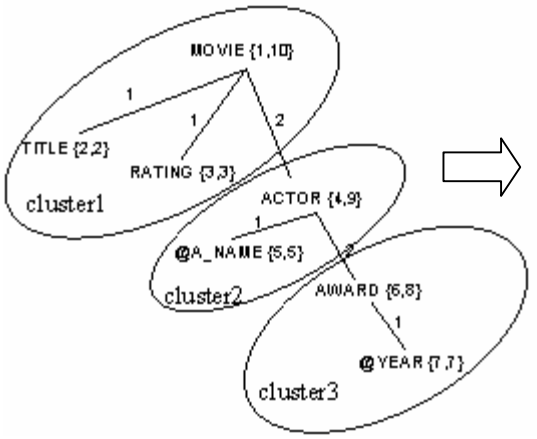
NODE	START_R	END_R	FREQUENCY
AWARD	9	12	2
YEAR	10	11	1

c)

Figure 5.11 : Grouping node example

Figure 5.11c shows complete process. There are three groups result. First group contain three records: movie, title, and rating. Next group contain two records: actor and a_name. Last group contain two records: award and year.

After finishing node grouping, the cluster-table configuration will be created. The cluster table configuration consists of information including id, node member, and relationship between clusters.



id	NODE	START_R	END_R
1	MOVIE, TITLE, RATING	1	14
2	ACTOR, A_NAME	6	13
3	AWARD, YEAR	9	12

Figure 5.12 : Cluster Table example

As show in **Figure 5.12**, the document "movies.xml" is fragmented into three-cluster table. The first cluster table has identification number (id) is equal to 1. It contains three node members including Movie, Title, and Rating. The Start Range and End Range is information about ancestor-descendant relationship between the cluster tables. The next cluster table has id equal to 2. It contain two node member including Actor and A_name. The last cluster table has id equal to 3. It contain two node member including Award and Year.

5.3 Node Configuration Creator

The node configuration creator takes as input the XML document, the SNRIT, and the cluster-table configuration and produces as output the node configuration. Figure 5.13 shows the main four modules inside the node configuration creator: paring, SNRIT matching, node checking, and node extension.

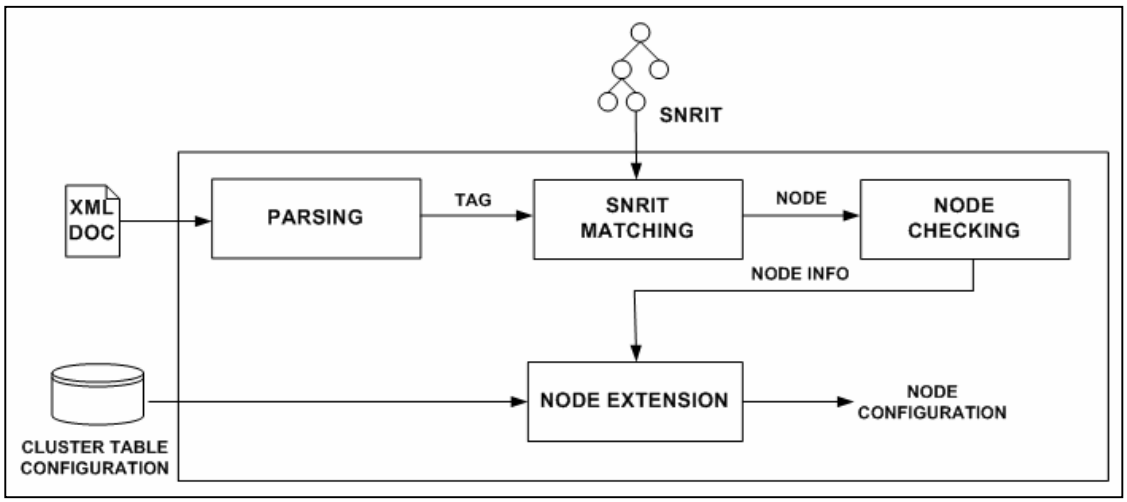


Figure 5.13 : Node configuration generator component

In the parsing module, the XML document is parsed. Having found each element tag, each tag is fed to the SNRIT matching module.

In the SNRIT matching module, the element type which each element is an instance is recovered. The element types are represented with the each node of the SNRIT. The SNRIT matching module traces the traversing around the SNRIT. Because of the tag is extracted sequentially from document which contains the huge of the nested tag. Traversing along the SNRIT is likely navigation along the path.

In the node checking module, each element type is checked if it is data node or not. If there is any one of its instance has data content, it is data node. Otherwise, it is non-data node. Initially, the node checking module will mark all node of SNRIT as non-data node. There are two states for node: data node and non-data node. If there is at least one instance of node has data content, that node is marked as data-node.

In the node extension module, the node configuration is created. The node configuration adopts some property from the SNRIT including node name (NODE), node type (TYPE), start range (START_R), and end range (END_R). Additional, there are two fields be added to each edge including TABLE_ID and COLUMN_NO. The TABLE_ID contains the cluster-table ID that the node is member. COLUMN_NO contains the order of allocated space that the cluster table allocates for that node. If node is data node, it requires space for its content. Therefore, the cluster table needs to

allocate space for that node and the COLUMN_NO is set to value that is more than 0. The COLUMN_NO field is numeric sequence generation starting from 1. If the node is non-data node, the cluster tables no needs to allocate space for that node. Consequence, The COLUMN_NO field is set to 0.

NODE	TYPE	START_R	END_R	TABLE_ID	COLUMN_NO
MOVIE	ELEMENT	1	14	1	0
TITLE	ELEMENT	2	3	1	1
RATING	ELEMENT	4	5	1	2
ACTOR	ELEMENT	6	13	2	0
A_NAME	ATTRIBUTE	7	8	2	1
AWARD	ELEMENT	9	12	3	1
YEAR	ATTRIBUTE	10	11	3	2

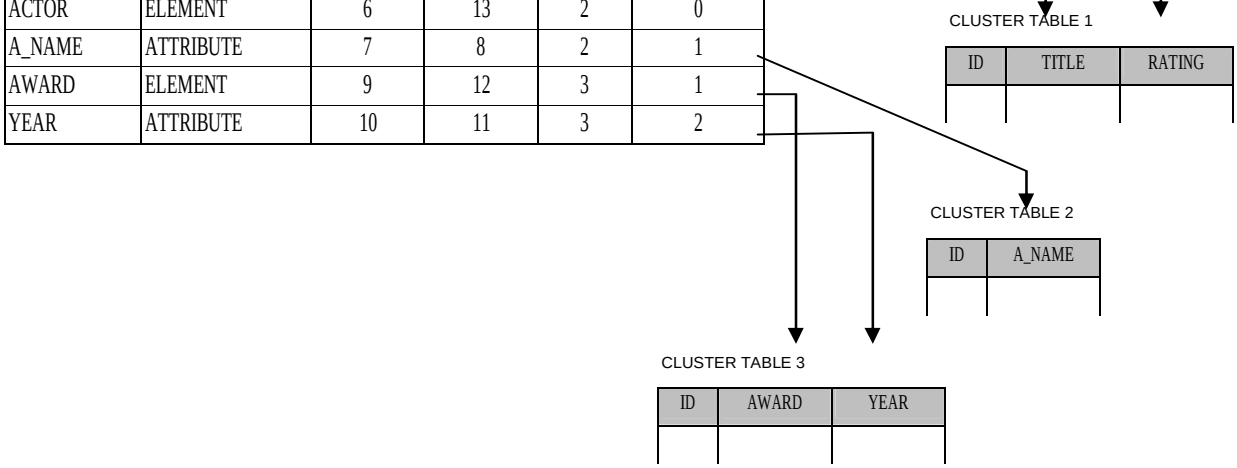


Figure 5.14 : Node configuration example

Figure 5.14 shows the node configuration. The title and the rating element types are allocated space at column 1 and 2 of cluster table 1 respectively. The a_name element type is allocated space at column 1 of cluster table 2. The award and the year element types are allocated space at column 1 and 2 of cluster table 3 respectively. We can see that the movie and the actor element types are not allocated space. Because of it is non-data node, so it is set the COLUMN_NO field to 0.

5.4 Cluster-Content Creator

The cluster-content creator takes as input the XML document, the node configuration, and the cluster-table configuration and produces as output the clustering document. Figure 5.15 shows the main four modules inside the cluster-content creator: parsing, node matching, storage management, and ID generation.

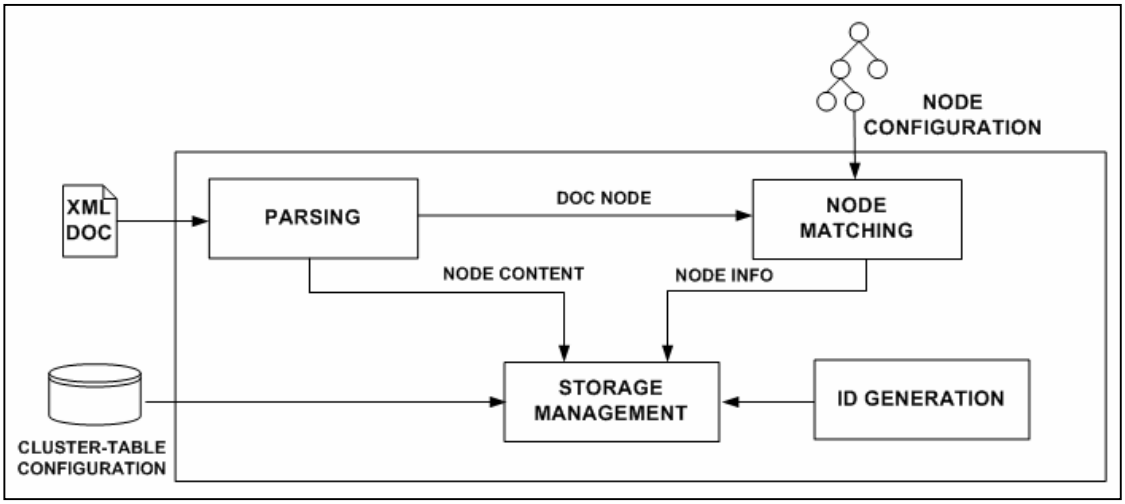


Figure 5.15 Cluster-Content Creator component

In the node matching module, the node configuration of each element is recovered. The node configuration identifies the cluster table and column number for each element.

In the storage management module, the data contents are stored into the cluster tables. If node is configured as data node, its data content is stored into record of the cluster table that node is member. The storage management module will monitor the record status of the cluster table. Whenever the record is full, it is flushed into the storage.

In the ID generation module, the record ID is assigned to the record whenever a new record is created. The ID generation module has to trace the record ID of the cluster table. Whenever a new record is added, it is able to trace the record ID of the record that is parent of a new record.

5.5 Data Storage Repository

```
<DOC>
<Clusters>
  <Cluster ID= "0" S="0" E="15"/>
  <Cluster ID= "1" S="1" E="14"/>
  <Cluster ID= "2" S="6" E="13"/>
  <Cluster ID= "3" S="9" E="12"/>
</Clusters>
<Nodes>
 <Node name= "movie-list" S="0" E="15" T="E" CID="0" CNO="0">
  <Node name= "movie" S="1" E="14" T="E" CID="1" CNO="0">
   <Node name= "title" S="2" E="3" T="E" CID="1" CNO="1"/>
   <Node name= "rating" S="4" E="5" T="E" CID="1" CNO="2"/>
   <Node name= "actor" S="6" E="13" T="E" CID="2" CNO="0">
    <Node name= "a_name" S="7" E="8" T="A" CID="2" CNO="1"/>
    <Node name= "award" S="9" E="12" T="E" CID="3" CNO="1">
     <Node name= "year" S="10" E="11" T="A" CID="3" CNO="2"/>
    </Node>
   </Node>
  </Node>
 </Node>
</Nodes>
<Contents>
<Cluster ID= "0">
  |
</Cluster>
<Cluster ID= "1">
  |1 |movie1 |5
  |2 |movie2 |4
  |3 |movie3 |3
</Cluster>
<Cluster ID= "2">
  |1.1 |a1
  |1.2 |a2
  |2.1 |a3
  |3.1 |a4
  |3.2 |a5
</Cluster>
<Cluster ID= "3">
  |1.1.1 |award1 |1998
  |1.2.1 |award2 |1998
  |2.1.1 |award1 |1996
  |2.1.2 |award3 |1997
  |3.1.1 |award2 |2000
</Cluster>
</Contents>
</DOC>
```

Figure 5.16 : Storage XML document example

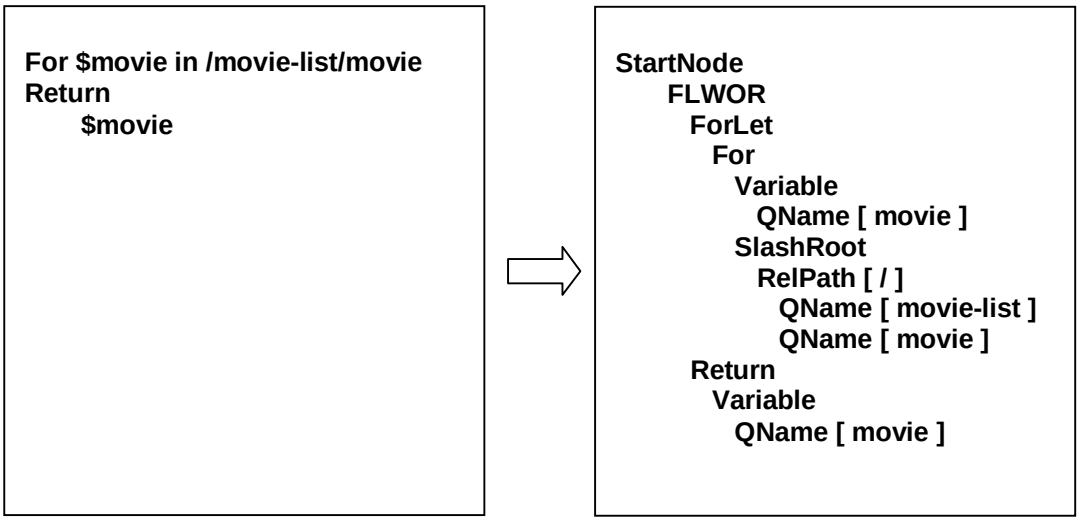
Having cluster content creator finished process; the cluster-table configuration, node configuration and the cluster tables are stored to data storage in XML file format. It is alternate to use a single document or multiple documents. As show in Figure 5.16, it is stored in a single document. It consists of three parts. The upper part is cluster-table configuration. It gives information about the cluster table. The middle part is node configuration. It gives information about structure tree that is mapped to cluster tables. Each node consists of six attributes. The attribute `name` represents node name. The attribute `S` represents Start Range. The attribute `E` represents End Range. The attribute `T` represents node type (Element, Attribute). The attribute `CID` represents cluster table that it located. And final the attribute `CNO` represents column number that it located. The lower part is sequence of the cluster tables. The first column is record ID.

5.6 Loader

The loader is component that functions in the retrieval phase. Its function is to load the cluster tables, cluster-table configuration and node configuration into main memory. Initially, node configuration and cluster-table configuration are loaded first. Whenever the cluster table is requested, loader will load the requested cluster table into main memory.

5.7 Query Parser

The query parser takes as input the user query and produces as output the parsed tree. The query parser will transform query text into internal tree structure called as parsed tree (alternatively called as query tree). The user query consists of at least one query expression which is predefined from the query language such as XPath [11], XQuery [4]. To evaluate XQuery, we use xqengine as XQuery parser. The xqengine [19] , released by Fatdog Software, is a java open source for searching collections of XML documents that uses an XQuery front end. Finally, the parse tree is converted to a query operation that is processed by query processor. Figure 5.17 illustrates a given query text and its corresponding parse tree which is produced by the engine.



a) The query text **b) The parse tree**

Figure 5.17 : Parse tree example

5.8 Query Processor

The query processor takes as input the parse tree, the node configuration, and the cluster tables and produces as output the query result. Figure 5.18 shows the main four modules inside the query processor: expression recognition, path evaluation, content extraction, and expression processing.

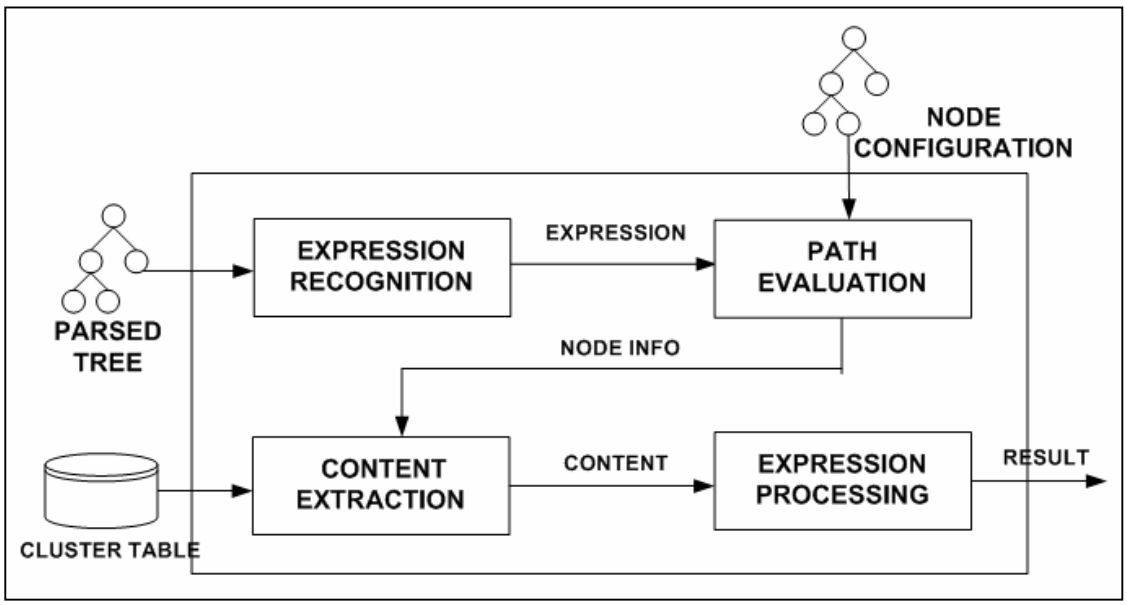


Figure 5.18 : Query Processor component

Query processor is component that responsible for evaluating the user query. The query expression makes of one or more query operation. Some query operation needs to access path and data contents. To access the data contents stored in the cluster tables, the query processor requires the node configuration.

In the expression recognition module, because of the query text is to be formatted based on the query grammar; the parse tree can be translated to query expression. Normally, the query expression is start by reserved word such as FOR, LET, WHERE. Whenever the node of parse tree is matched the reserve word, query expression is recognized.

Normally, the query expression consists of several query operations. For example is shown in Figure 5.19. If the query operation requests to access the path, the path evaluation module will be trigger. For example, path /movie-list/movie is mapped to the column 0 of cluster table 1.

Query Expression:	**Query Operation:**
For $a in /movie-list/movie	**1. Load content that match the path "/movie-list/movie".** **2. Define variable $a.** **3. Set $a reference to loaded content.**

Figure 5.19 : The query operation example

In the path evaluation module, the location of the data contents is recovered. It requires the node configuration. Each path is matched with the node configuration. Then, the data configuration is created. The data configuration defines the detail of data contents that are loaded including cluster-table ID and column NO. Finally, the data configuration is fed to the content extraction module.

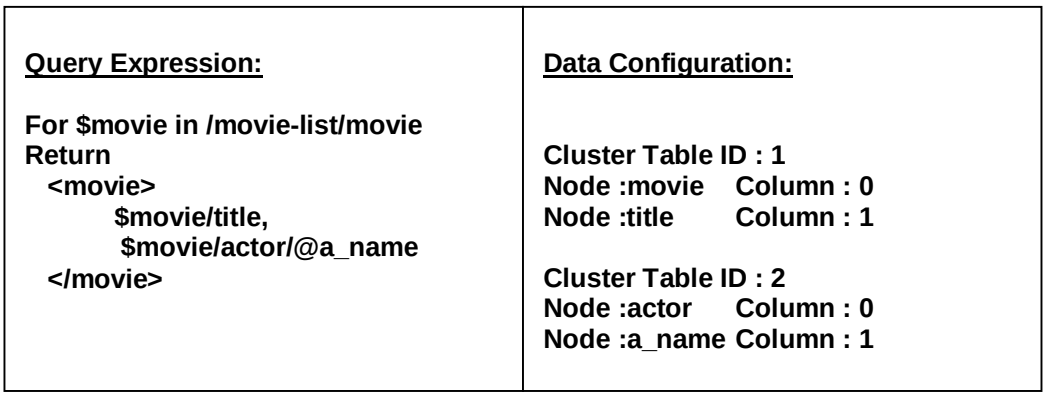


Figure 5.20 : Data configuration example

For example as show in Figure 5.20, the query expression concerns about movie information. It returns the movie title and the actor name. The requested query path is following:

Path: /movie-list/movie

Path: /movie-list/movie/title

Path: /movie-list/movie/actor

Path: /movie-list/movie/actor/@a_name

The data configuration is shown on the right of Figure 5.20. There are two cluster table are requested including cluster table 1 and 2. The column 1 of cluster table 1 and the column 1 of cluster table 2 are loaded. We can see that column 0 of cluster table 1 and column 0 of cluster table 2 are referenced also. Because of the field column NO 0 is reserved for non-data node, no have data content is loaded. It is used to reserve XML structure.

In the content extraction module, the cluster tables that are reference in the data configuration are loaded into main memory. The XML document tree is reconstructed. If one more the cluster table is loaded, ancestor-descendant link will be crated. The cluster table is shown in Figure 5.12. The ancestor-descendant link is reconstructed with comparing record ID. As already discussion, the record ID contains enclosed path. It enclosed parent path ID in its ID. Consequence, completely construction link is shown in Figure 5.21.

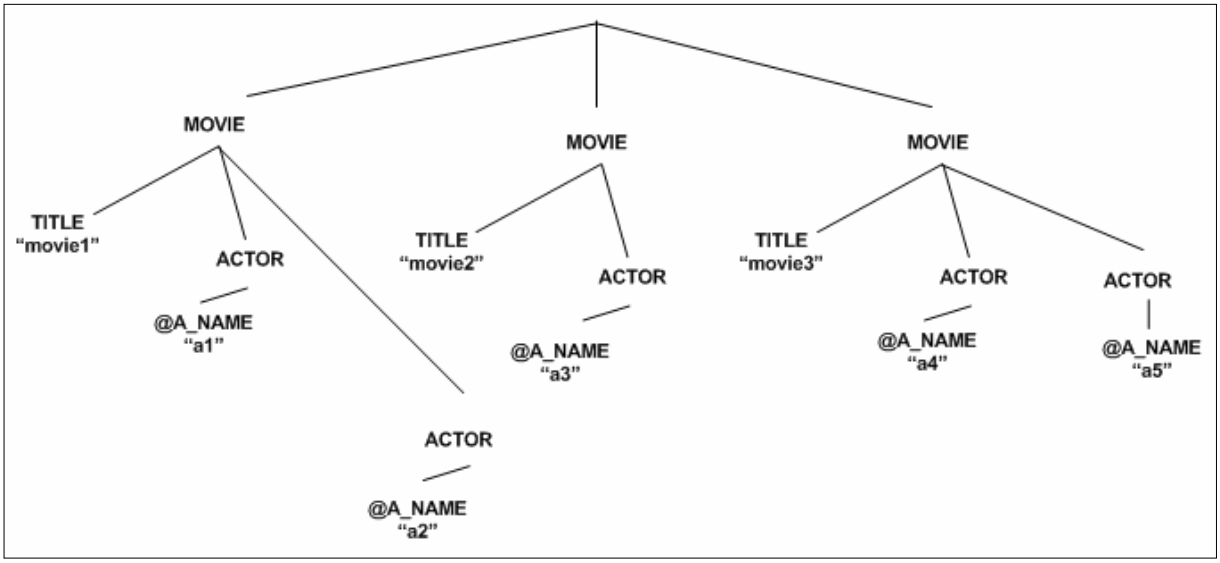


Figure 5.21 : Construct tree example

In the expression-processing module, each query expression is operated; the sequence of the query operation is executed. The query operation divided into a several type including data format, data filter, data loading, and so on. The result of the above query expression is shown in Figure 5.22.

```
<movie>
    <title>movie1</title>
    <actor a_name="a1"/>
    <actor a_name="a2"/>
</movie>

<movie>
    <title>movie2</title>
    <actor a_name="a3"/>
</movie>

<movie>
    <title>movie3</title>
    <actor a_name="a4"/>
    <actor a_name="a5"/>
 </movie>
```

Figure 5.22 : Query result example

5.9 Summary

In this chapter, we present the table-based clustering engine to reorganize XML data using our table-based clustering strategy. The engine performs two phases: Storage and retrieval. The storage phase clusters the XML data elements into a set of cluster tables. The retrieval phase evaluates user query.

The storage phase consists of four components: SNRIT creator, cluster-table configuration creator, node configuration creator and cluster-content creator. SNRIT is extended from the XML structure tree. After the SNRIT is created, the cluster-table configuration and node configuration will be configured. They are used to identify the particular table which is appropriated with each element type. Finally, the XML document is fragmented and XML data contents are grouped and placed into the several tables.

The retrieval phase consists of three components: Loader, Query Parser and Query Processor. Loader loads the node configuration and cluster-table configuration into main memory. Query parser translates the user query into parse tree. Query processor evaluates parses tree and return query results to client.

CHAPTER VI
EXPERIMENTS

In this chapter, we provide experimental results on the performance of our table-based clustering engine with respect to the element-based and the subtree-based clustering engines. The experiments are conducted on personal computer with Intel Pentium IV 2.8 GHz microprocessor with 256 MB RAM running Microsoft Windows XP Professional. The program is developed on Java2 (SDK 1.4.2) environment.

6.1 Description of the Test Data

The test data used in our experiment come from XBench database generator v1.0 [20]. We generate a set of data-centric XML documents each of which contains the information about product catalog. The schema diagram of the product catalog is summarized in Appendix B. There are 67 different types of element. The XML documents have different number of product items: hence, they have different size.

Table 6.1 : The data sets information

Document Name	Number of items	file size (KB)
Catalog1.xml	**250**	**1,058**
Catalog2.xml	**500**	**2,071**
Catalog10.xml	**2,500**	**10,617**
Catalog20.xml	**5,000**	**21,200**
Catalog50.xml	**12,500**	**52,916**

Table 6.1 provides a summary of the size of the document. The size of the document is ranging from 1 MB to 50 MB. The size of the document is varied to the number of product items.

6.2 Description of the Test Query

Table 6.2 : Query group overview

Query Group	Number of Element type	Number of Query	Query Number
G1	**1**	**5**	**Q1.1-Q1.5**
G2	**10**	**2**	**Q2.1-Q2.2**
G3	**20**	**3**	**Q3.1-Q3.3**
G4	**30**	**3**	**Q4.1-Q4.3**
G5	**40**	**3**	**Q5.1-Q5.3**
G6	**50**	**2**	**Q6.1-Q6.2**
G7	**60**	**2**	**Q7.1-Q7.2**

The test queries used in our experiment are divided into eleven groups. Each group has different numbers of element types. Each query in the same group produces different numbers of clusters that are loaded. Table6.2 gives information about the query groups including the number of element type that related with query, the number of query, and the list of query. The details of the test queries are shown in Appendix C.

6.3 Measurement

There are five measurements in our experiments.

- *Clustering size* is the size of the document which contains XML data after clustering.
- *Downsizing ratio*, as show in equation 6.1, is the size of the document after clustering with respect to the size of the document before clustering.

$$Downsizing \quad Ratio \quad = \quad \left(1 \quad - \quad \frac{DOC_{clus}}{DOC_{orig}} \right) \quad \times \quad 100 \qquad (6.1)$$

Where $\boldsymbol{DOC_{orig}}$ is the size of the original XML document and $\boldsymbol{DOC_{clus}}$ is the size of the document after clustering. The higher downsizing ratio is, the more efficient in clustering strategy.

- *Clustering time* points from the time that clustering engine reads the given XML document to the time that the clustering engine produces the document in which the data has been clustered.

- *Querying time* measures from the time that the query processor loads clustered data corresponding to the user query to the time that the query processor produces the query answer.

- *Reconstruction time* is the time that the original XML document is reproduced.

To be comparable among three strategies, we implement our table-based, the element-based and the subtree-based clustering engines in the same environment. The simple XML documents that are resulted from clustering process of three strategies are shown in Appendix A. Each experiment is tested five times, and then average to obtain result.

6.4 Clustering Size Comparison

Table 6.3 : Comparative clustering size experiment

Document Name	Clustering Size (KB)		
	Table-based	Element-based	Subtree-based
Catalog1.xml	496	808	678
Catalog2.xml	967	1,583	1,326
Catalog10.xml	4,966	8,253	6,813
Catalog20.xml	9,934	16,587	13,631
Catalog50.xml	24,832	41,779	34,113

Table 6.3 shows the clustering output size for our table-based clustering (column 2) comparing to the element-based (column 3) and the subtree-based clustering strategies (column 4). The clustering output sizes presents in unit of kilobytes (KB). We can see that, in every case, the table-based clustering strategy produces the smallest size of document whereas the element-based clustering strategy produces the largest size of document. This is because our table-based clustering

strategy groups several element types and stores together into the cluster tables. Many element types are grouped together as a record in the table. Although the subtree-based clustering strategy performs node grouping, it provide worse clustering size than our table-based clustering strategy since the path is encoded and stored into the table.

6.5 Downsizing Ratio Comparison

Document Name	Downsizing Ratio		
	Table-based	Element-based	Subtree-based
Catalog1.xml	53.11909	23.62949	35.91682
Catalog2.xml	53.30758	23.5635	35.97296
Catalog10.xml	53.22596	22.26618	35.82933
Catalog20.xml	53.14151	21.75943	35.70283
Catalog50.xml	53.07279	21.04656	35.53179

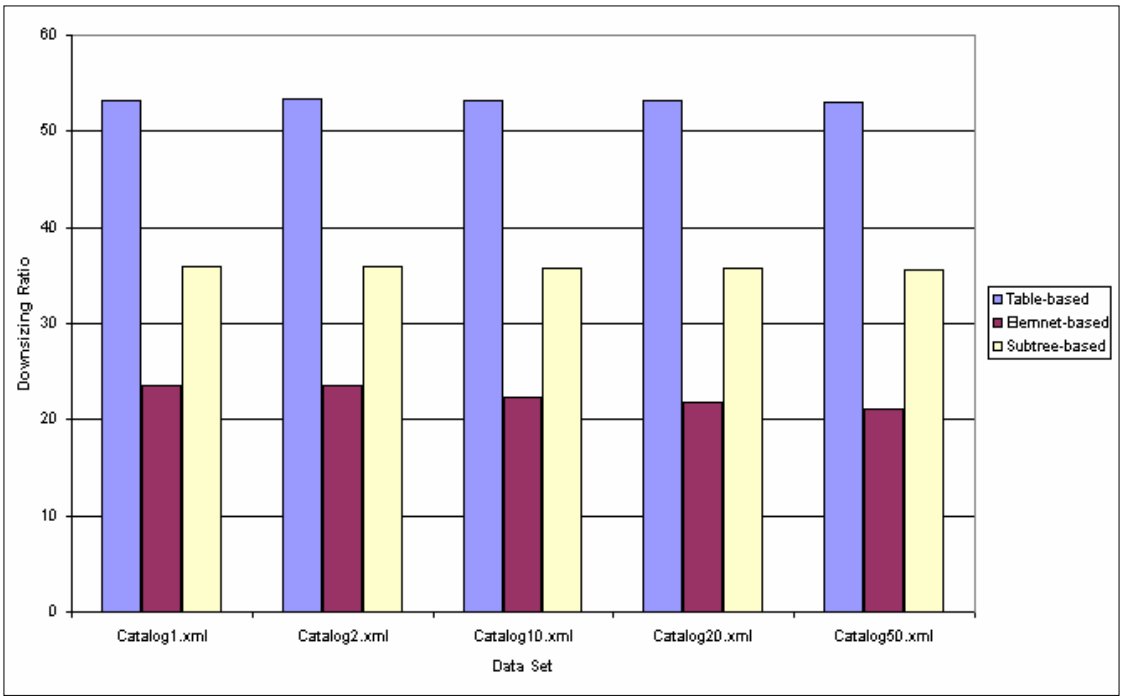


Figure 6.1 : Chart of downsizing ratio comparison including its data

Figure 6.1 shows the downsizing ratio of our table-based clustering strategy compare to the element-based and the subtree-based clustering strategies. Our table-based clustering can downsize the documents by 53% with respect to its original size. Note that our table-based clustering strategy produces the best downsizing ratio

when comparing to the element-based and the subtree-based clustering strategies respectively.

6.6 Clustering Time Comparison

Figure 6.2 shows the clustering time of our table-based clustering strategy compare to the element-based and the subtree-based clustering strategies. We can see that our table-based clustering consumes time in clustering process lower than the subtree-based and the element-based clustering respectively. Note also that, the clustering time is proportional to the size of the given document.

Document Name	Clustering Time(secs)		
	Table-based	Element-based	Subtree-based
Catalog1.xml (1MB)	0.4933	1.0767	0.6433
Catalog2.xml (2MB)	0.5500	1.5333	0.7667
Catalog10.xml (10MB)	2.9467	8.3967	4.2833
Catalog20.xml (20MB)	7.8933	20.9600	11.6667
Catalog50.xml (50MB)	28.3433	59.6767	40.1167

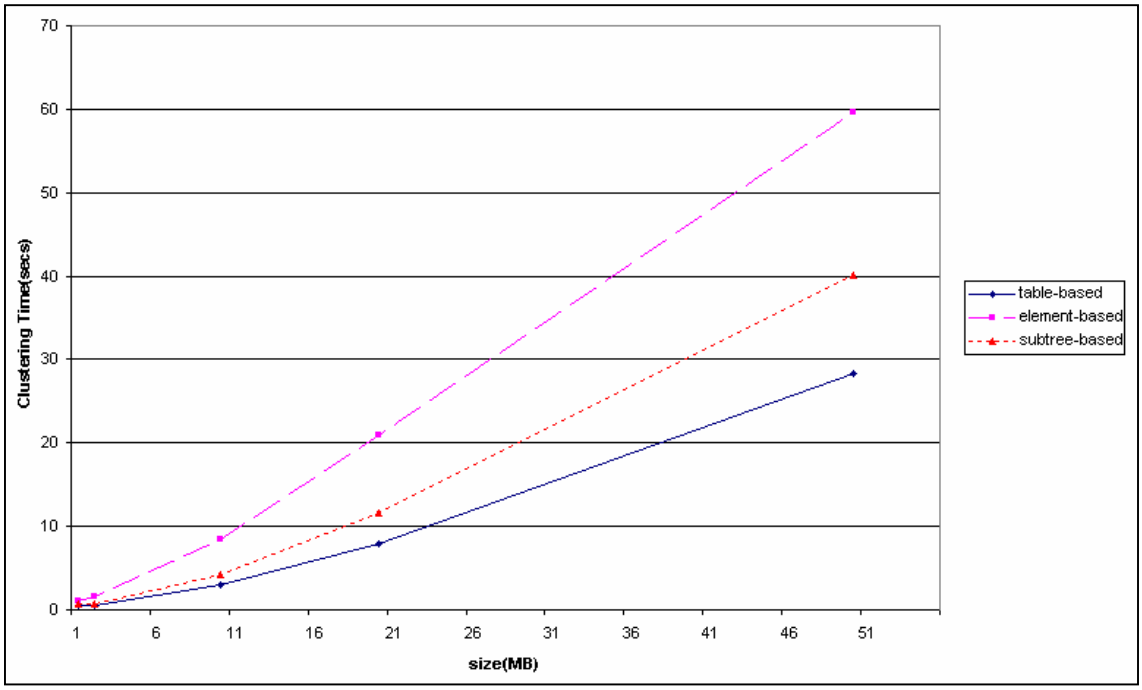


Figure 6.2 : Chart of clustering time comparison including its data

6.7 Querying Time Comparison

Query Group	Querying Time (seconds)		
	Table-based	Element-based	Subtree-based
G1	0.2513	0.1367	0.4080
G2	0.7800	0.8167	1.3433
G3	1.2878	1.5922	1.9389
G4	1.5800	2.4033	2.2356
G5	1.7611	2.9311	2.4067
G6	2.2333	3.8583	3.0433
G7	2.5583	4.6383	3.3783

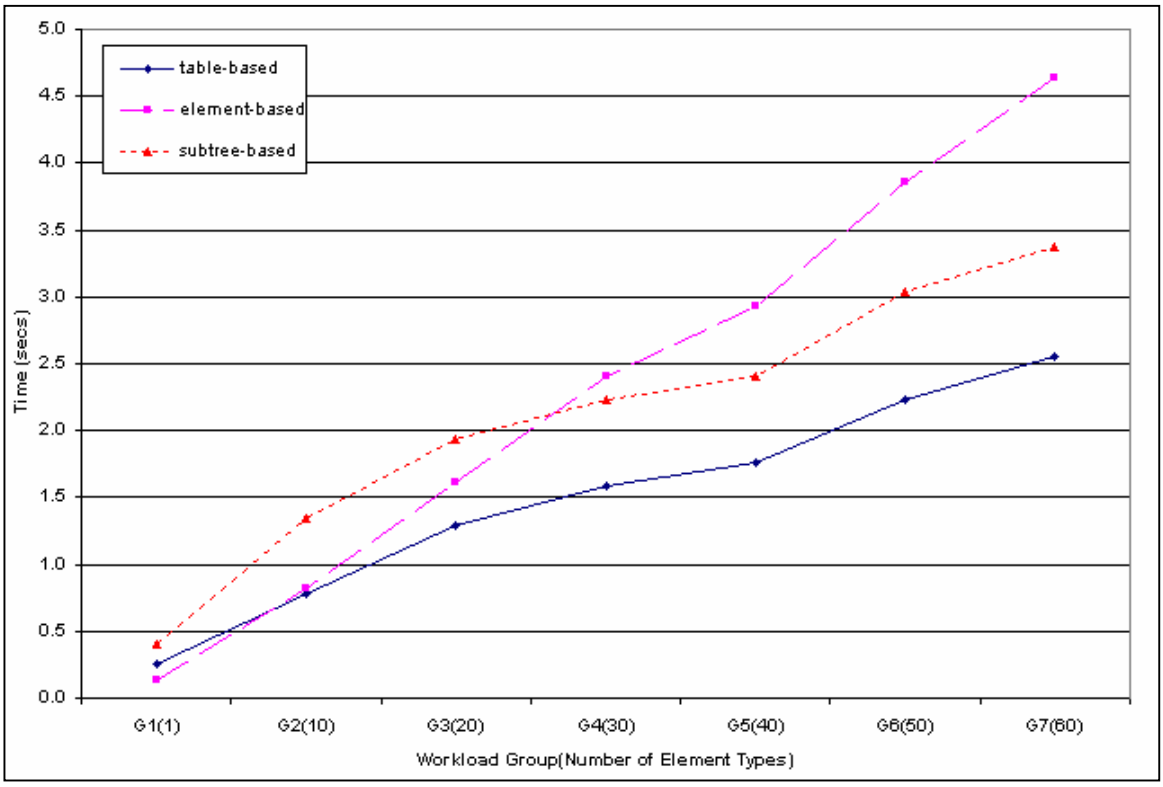


Figure 6.3 : Chart an querying time comparison including its data

To compare the querying time among our table-based, the element-based, and the subtree-based clustering strategies, we used data from catalog10.xml as a test bed. Figure 6.3 shows the time complexity for clustering data. We can see that the table-based clustering strategy performs best for complex queries in which at least

10 element types are required. This is because our table-based clustering strategy groups several element types and stores into the same cluster table. For simple queries in which less than 10 element types are required, the element-based clustering strategy performs best. This is because the element-based strategy groups several elements which are instance of the same element type together into the cluster page. Note also that, the subtree-based clustering strategy performs better than element-based clustering strategy when the query accesses more than 25 element types. This is because the subtree-based clustering strategy groups several element types and stores into the same cluster table (like our table-based clustering strategy). However, the subtree-based clustering strategy needs to take the overhead for managing sub-trees. As a result, the subtree-based clustering strategy takes the number of I/O higher than our table-based clustering strategy.

6.8 Reconstruction Time Comparison

Document Name	Reconstruction Time (seconds)		
	Table-based	Element-based	Subtree-based
Catalog1.xml	0.3433	0.5733	0.4333
Catalog2.xml	0.7200	1.2867	0.8833
Catalog10.xml	3.9933	6.6600	4.8033
Catalog20.xml	15.1233	28.0067	20.5400
Catalog50.xml	31.1033	68.6933	52.6100

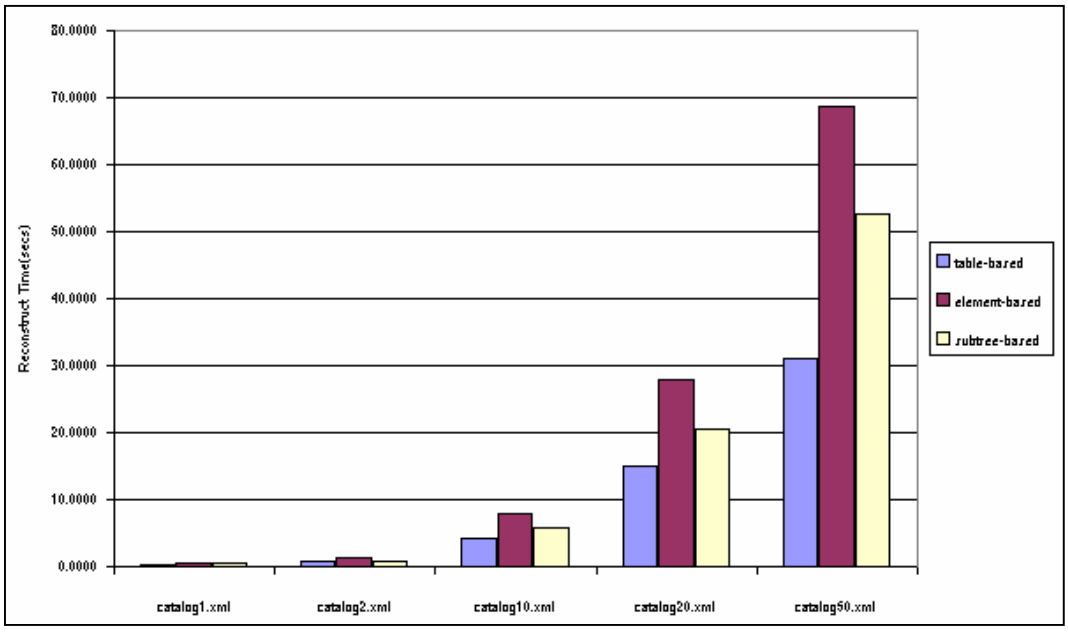


Figure 6.4 : Chart of reconstruction Time Comparison

Figure 6.4 shows the time complexity for reconstruction of XML document using different three strategies. We can see that our table-based strategy provides the best performance in all cases. This is because our table-based strategy can reduce the number of I/O and the overhead for managing sub-trees regarding the element-based and the subtree-based strategies, respectively.

6.9 Summary

In this chapter, we provided the experiment for comparative performance of table-based clustering strategy with respect to the element-based and the subtree-based clustering strategies. The experiment divides into five parts: the clustering size, the downsizing ratio, the clustering time, the querying time, and the reconstruction time.

In the storage point-of-view, our table-based clustering strategy has shown the best clustering size, the best downsizing ratio, the best clustering time, and the best reconstruction time. In the querying point-of-view, our table-based clustering strategy provides the best query response time for complex queries which access a large set of element types. Our table-based clustering strategy performs fairly well for simple queries which access a few number of element types.

CHAPTER VII
CONCLUSION AND FUTURE WORK

In this thesis, we have presented a new *table-based clustering strategy* for efficient storage and querying of XML data. Our strategy is based on the *Structure and Node Relationship Information Tree (SNRIT)* which is an XML structure tree with a frequency and a node range information.

Our table-based clustering strategy categorizes XML data elements with respect to their types and classifies the schema tree regarding the cardinalities of relationships between data elements. The data elements are stored into a set of tables. The data elements of the same type are stored into the same table to support fast access of simple queries. The relationships between data elements are indicated by the enclosed path which supports efficient accessing more complex queries.

We have developed and demonstrated an engine that uses our table-based clustering strategy. The engine consists of two phases: Storage and retrieval. In storage phase, the data element and its structure are differentiated; A given XML document is parsed and the structural information is extracted. The structural information is used to cluster data elements. In retrieval phase, a user query is parsed and evaluated. Only tables that are related to the user query are loaded.

We have compared our table-based clustering strategy with the element-based and the subtree-based clustering strategies. Regarding the experimental result, our table-based clustering can downsize the XML documents by 53% with respect to its original size. Additionally, our table-based clustering strategy produces the best downsizing ratio when comparing to the element-based and the subtree-based clustering strategies. Moreover, our table-based clustering strategy produces the good performance for querying XML documents.

To support efficient storage, the use of specific technique for each element type such as date time, number, string, character can extend. Additionally, compression technique can be considered. To support efficient query response time, index and query optimization can be the next focuses.

REFERENCES

1. W3C. “Extensible Markup Language (XML) 1.0 (Second Edition)”, 2000, Available at http://www.w3c.org/TR/2000/REC-xml-20001006.
2. W3C. Document Object Model (DOM). 2002. DOM Level 3 Core Specification, Version 1.0, available at http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20020409/.
3. David Megginson. SAX: A simple API for XML. Technical report, Megginson Technologies, 2001.
4. D. Chambelin, J. Clark, D. Florescu, J. Robie, J. Sim´eon, and M. Stefanescu. XQuery 1.0: An XML query language. W3C Working Draft, June 2001 , available at http://www.w3.org/TR/xquery .
5. J. Bosac, T. Bray, D. Connolly, E. Maler, G. Nicol, C.M. Sperberg-McQueen, L. Wood, and J. Clark. Guide to the W3C XML specification (“XMLspec”) DTD, June 1998. http://www.w3.org/XML/1998/06/xmlspec-report.
6. D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML in a relational database. Technical Report 3680, INRIA, 1999.
7. A. Schmidt, M. Kersten, M.Windhouwer, and F.Waas. Efficient relational storage and retrieval of XML documents. In Proc. of Intl.Workshop on theWeb and Databases (WebDB), 2000.
8. D. Florescu, D. Kossman, Storing and Querying XML Data using an RDMBS. IEEE Data Engineering Bulletin 22(3), 1999.
9. J. Shanmugasundaram, K. Tufte, et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999.
10. F. Tian, D. DeWitt, J. Chen, and C. Zhung. The design and performance evaluation of various XML storage strategies. http://www.cs.wisc.edu/niagra/Publications.html, 2001.

11. James Clark and Steve DeRose. XML path language (XPath) version 1.0. Technical report, World Wide Web Consortium (W3C) Recommendation, 1999.
12. World Wide Web Consortium. XML Schema Part 0: Primer. 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/.
13. World Wide Web Consortium. XML Schema Part 1: Structures. 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.
14. World Wide Web Consortium. XML Schema Part 2: Datatypes. 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.
15. H. V. Jagadish, Shurug AL-Khalifa, et al. TIMBER: A Native XML Database. Technical Report, University of Michigan, April 2002.
16. Carl-Christian Kanne. Natix: A Native XML Base Management System. PhD thesis, University of Mannheim, 2002.
17. X. Meng, D. Luo, M. Lee and J. An. OrientStore: A Schema Based Native XML Storage System. In: Proc. of VLDB 2003. Berlin, Germany. 1057-1060.
18. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. SIGMOD Record, Vol.26(3):54-66, September 1997.
19. Xqengine. Available at http://xqengine.sourceforge.net/
20. Xbench. Available at http://db.uwaterloo.ca/~ddbms/projects/xbench/

APPENDIX

APPENDIX A

EXAMPLE DOCUMENT CLUSTERING

Original XML document

```
<movie-list>
  <movie>
    <title>movie1</title>
    <rating>5</rating>
    <actor a_name="a1">
          <award year="1998">award1</award>
    </actor>
    <actor a_name="a2">
         <award year="1998">award2</award>
    </actor>
  </movie>
  <movie>
    <title>movie2</title>
    <rating>4</rating>
    <actor a_name="a3">
          <award year="1996">award1</award>
         <award year="1997">award3</award>
    </actor>
  </movie>
  <movie>
    <title>movie3</title>
    <rating>3</rating>
    <actor a_name="a4">
          <award year="2000">award2</award>
    </actor>
    <actor a_name="a5"/>
  </movie>
</movie-list>
```

1. Table-based clustering Document

1.1 Cluster-Table Configuration

```
Cluster Information
Cluster 0 (S:0, E:15, F:1)
  Member:  movie-list(S:0, E:15, F:1, C:0)
Cluster 1 (S:1, E:14, F:3)
  Member:  movie(S:1, E:14, F:3, C:0), title(S:2, E:3, F:1, C:1),
rating(S:4, E:5, F:1, C:2)
Cluster 2 (S:6, E:13, F:2)
  Member:  actor(S:6, E:13, F:2, C:0), a_name(S:7, E:8, F:1, C:1)
Cluster 3 (S:9, E:12, F:2)
  Member:  award(S:9, E:12, F:2, C:1), year(S:10, E:11, F:1, C:2)
```

1.2 Node Configuration

```
<Nodes>
 <Node name= "movie-list" S="0" E="15" T="E" CID="0" CNO="0">
  <Node name= "movie" S="1" E="14" T="E" CID="1" CNO="0">
   <Node name= "title" S="2" E="3" T="E" CID="1" CNO="1"/>
   <Node name= "rating" S="4" E="5" T="E" CID="1" CNO="2"/>
   <Node name= "actor" S="6" E="13" T="E" CID="2" CNO="0">
    <Node name= "a_name" S="7" E="8" T="A" CID="2" CNO="1"/>
    <Node name= "award" S="9" E="12" T="E" CID="3" CNO="1">
     <Node name= "year" S="10" E="11" T="A" CID="3" CNO="2"/>
    </Node>
   </Node>
  </Node>
 </Node>
</Nodes>
```

1.3 Data Content

```
<Contents>
<Cluster ID= "0">
  |
</Cluster>
<Cluster ID= "1">
  |1 |movie1 |5
  |2 |movie2 |4
  |3 |movie3 |3
</Cluster>
<Cluster ID= "2">
```

```
  |1.1 |a1
  |1.2 |a2
  |2.1 |a3
  |3.1 |a4
  |3.2 |a5
</Cluster>
<Cluster ID= "3">
  |1.1.1 |award1 |1998
  |1.2.1 |award2 |1998
  |2.1.1 |award1 |1996
  |2.1.2 |award3 |1997
  |3.1.1 |award2 |2000
</Cluster>
</Contents>
```

2. Element-based clustering Document

2.1 Cluster-Table Configuration

```
Cluster 0 (S:0, E:15, F:1)
  Member:  movie-list(S:0, E:15, F:1, C:0)
Cluster 1 (S:1, E:14, F:3)
  Member:  movie(S:1, E:14, F:3, C:0)
Cluster 2 (S:2, E:3, F:1)
  Member:  title(S:2, E:3, F:1, C:1)
Cluster 3 (S:4, E:5, F:1)
  Member:  rating(S:4, E:5, F:1, C:1)
Cluster 4 (S:6, E:13, F:2)
  Member:  actor(S:6, E:13, F:2, C:0)
Cluster 5 (S:7, E:8, F:1)
  Member:  a_name(S:7, E:8, F:1, C:1)
Cluster 6 (S:9, E:12, F:2)
  Member:  award(S:9, E:12, F:2, C:1)
Cluster 7 (S:10, E:11, F:1)
  Member:  year(S:10, E:11, F:1, C:1)
```

2.2 Node Configuration

```
<Nodes>
 <Node  name= "movie-list" S="0" E="15" T="E" CID="0" CNO="0">
  <Node  name= "movie" S="1" E="14" T="E" CID="1" CNO="0">
   <Node  name= "title" S="2" E="3" T="E" CID="1" CNO="1"/>
   <Node  name= "rating" S="4" E="5" T="E" CID="1" CNO="2"/>
```

```
    <Node  name= "actor" S="6" E="13" T="E" CID="2" CNO="0">
      <Node  name= "a_name" S="7" E="8" T="A" CID="2" CNO="1"/>
      <Node  name= "award" S="9" E="12" T="E" CID="3" CNO="1">
        <Node  name= "year" S="10" E="11" T="A" CID="3" CNO="2"/>
      </Node>
    </Node>
  </Node>
 </Node>
</Nodes>
```

2.3 Data Content

```
<Contents>
<Cluster ID= "0">
  |
</Cluster>
<Cluster ID= "1">
  |1
  |2
  |3
</Cluster>
<Cluster ID= "2">
  |1.1 |movie1
  |2.1 |movie2
  |3.1 |movie3
</Cluster>
<Cluster ID= "3">
  |1.2 |5
  |2.2 |4
  |3.2 |3
</Cluster>
<Cluster ID= "4">
  |1.3
  |1.4
  |2.3
  |3.3
  |3.4
</Cluster>
<Cluster ID= "5">
  |1.3.1 |a1
  |1.4.1 |a2
  |2.3.1 |a3
  |3.3.1 |a4
  |3.4.1 |a5
</Cluster>
```

```
<Cluster ID= "6">
  |1.3.2 |award1
  |1.4.2 |award2
  |2.3.2 |award1
  |2.3.3 |award3
  |3.3.2 |award2
</Cluster>
<Cluster ID= "7">
  |1.3.2.1 |1998
  |1.4.2.1 |1998
  |2.3.2.1 |1996
  |2.3.3.1 |1997
  |3.3.2.1 |2000
</Cluster>
</Contents>
```

3. Subtree-based clustering Document

3.1 Cluster-Table Configuration

```
Cluster Information
Cluster 0 (S:0, E:15, F:1)
  Member:  movie-list(S:0, E:15, F:1, C:0)
Cluster 1 (S:1, E:14, F:3)
  Member:  movie(S:1, E:14, F:3, C:0), title(S:2, E:3, F:1, C:1),
rating(S:4, E:5, F:1, C:2)
Cluster 2 (S:6, E:13, F:2)
  Member:  actor(S:6, E:13, F:2, C:0), a_name(S:7, E:8, F:1, C:1)
Cluster 3 (S:9, E:12, F:2)
  Member:  award(S:9, E:12, F:2, C:1), year(S:10, E:11, F:1, C:2)
```

3.2 Node Configuration

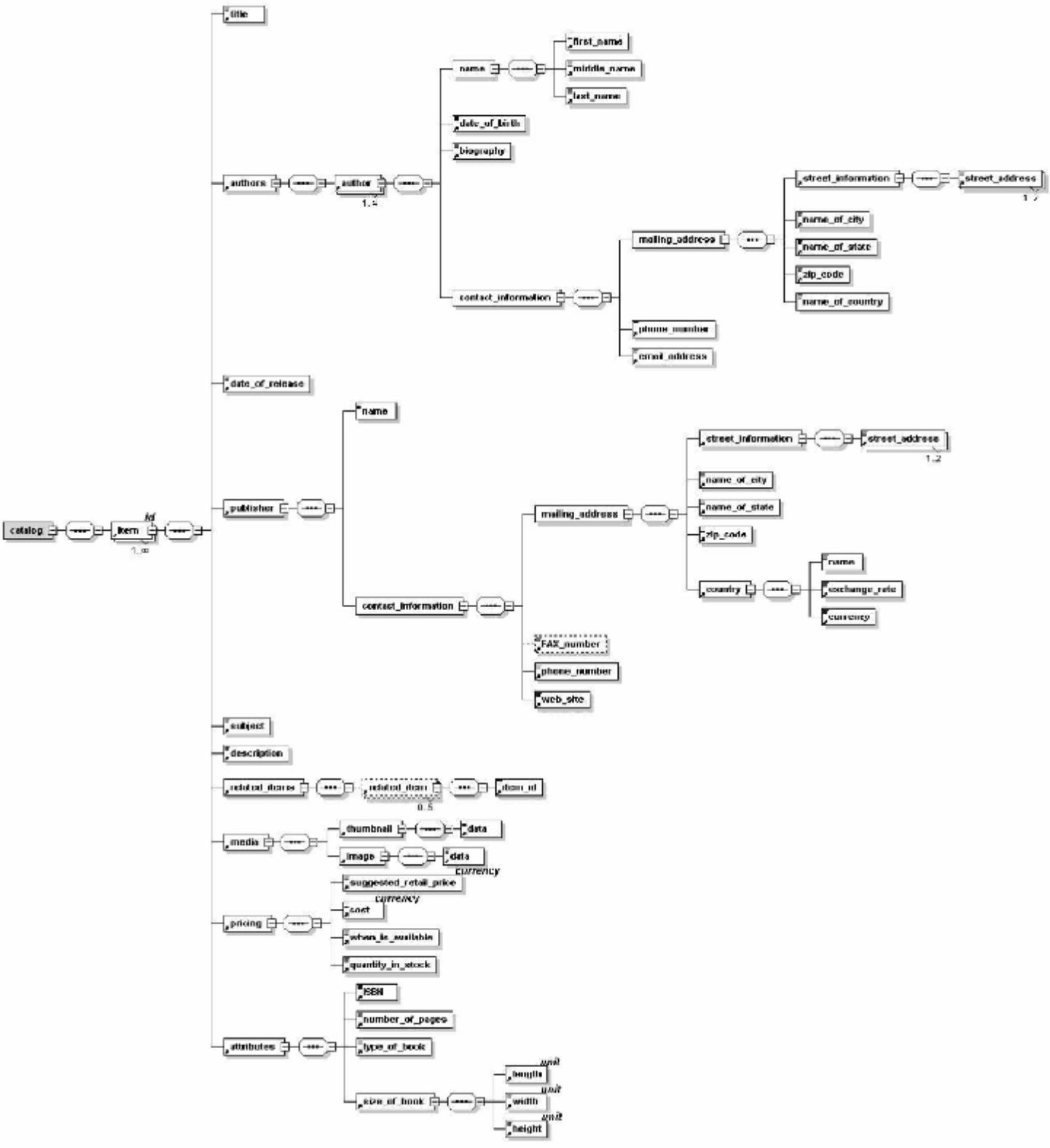
```
<Nodes>
 <Node name= "movie-list" S="0" E="15" T="E" CID="0" CNO="0">
  <Node name= "movie" S="1" E="14" T="E" CID="1" CNO="0">
   <Node name= "title" S="2" E="3" T="E" CID="1" CNO="1"/>
   <Node name= "rating" S="4" E="5" T="E" CID="1" CNO="2"/>
   <Node name= "actor" S="6" E="13" T="E" CID="2" CNO="0">
    <Node name= "a_name" S="7" E="8" T="A" CID="2" CNO="1"/>
    <Node name= "award" S="9" E="12" T="E" CID="3" CNO="1">
     <Node name= "year" S="10" E="11" T="A" CID="3" CNO="2"/>
    </Node>
   </Node>
  </Node>
 </Node>
</Nodes>
```

3.3 Data Content

```
<Contents>
<Cluster ID= "0">
  | |0 |L |1 1 |L |1 2 |L |1 3 |E
</Cluster>
<Cluster ID= "1">
  |1 |0 |1 |movie1 |E |2 |5 |E |L |2 1.1 |L |2 1.2 |E
  |2 |0 |1 |movie2 |E |2 |4 |E |L |2 2.1 |E
  |3 |0 |1 |movie3 |E |2 |3 |E |L |2 3.1 |L |2 3.2 |E
</Cluster>
<Cluster ID= "2">
  |1.1 |0 |1 |a1 |L |3 1.1.1 |E
  |1.2 |0 |1 |a2 |L |3 1.2.1 |E
  |2.1 |0 |1 |a3 |L |3 2.1.1 |L |3 2.1.2 |E
  |3.1 |0 |1 |a4 |L |3 3.1.1 |E
  |3.2 |0 |1 |a5 |E
</Cluster>
<Cluster ID= "3">
  |1.1.1 |0 |award1 |1 |1998 |E
  |1.2.1 |0 |award2 |1 |1998 |E
  |2.1.1 |0 |award1 |1 |1996 |E
  |2.1.2 |0 |award3 |1 |1997 |E
  |3.1.1 |0 |award2 |1 |2000 |E
</Cluster>
</Contents>
```

APPENDIX B
TEST DATA

1. The Products Catalog Schema Diagram



2. Cluster-Table Configuration

```
Cluster 0 (S:0, E:133, F:1)
  Member:  catalog(S:0, E:133, F:1, C:0)
Cluster 1 (S:1, E:132, F:25)
  Member:  item(S:1, E:132, F:25, C:0), id(S:2, E:3, F:1, C:1),
  title(S:4, E:5, F:1, C:2), authors(S:6, E:41, F:1, C:0),
  date_of_release(S:42, E:43, F:1, C:3), publisher(S:44, E:75,
  F:1, C:0), name(S:45, E:46, F:1, C:4), contact_information(S:47,
  E:74, F:1, C:0), mailing_address(S:48, E:67, F:1, C:0),
  street_information(S:49, E:52, F:1, C:0), name_of_city(S:53,
  E:54, F:1, C:5), name_of_state(S:55, E:56, F:1, C:6),
  zip_code(S:57, E:58, F:1, C:7), country(S:59, E:66, F:1, C:0),
  name(S:60, E:61, F:1, C:8), exchange_rate(S:62, E:63, F:1, C:9),
  currency(S:64, E:65, F:1, C:10), FAX_number(S:68, E:69, F:1,
  C:11), phone_number(S:70, E:71, F:1, C:12), web_site(S:72, E:73,
  F:1, C:13), subject(S:76, E:77, F:1, C:14), description(S:78,
  E:79, F:1, C:15), related_items(S:80, E:85, F:1, C:0),
  media(S:86, E:95, F:1, C:0), thumbnail(S:87, E:90, F:1, C:0),
  data(S:88, E:89, F:1, C:0), image(S:91, E:94, F:1, C:0),
  data(S:92, E:93, F:1, C:0), pricing(S:96, E:109, F:1, C:0),
  suggested_retail_price(S:97, E:100, F:1, C:16), currency(S:98,
  E:99, F:1, C:17), cost(S:101, E:104, F:1, C:18), currency(S:102,
  E:103, F:1, C:19), when_is_available(S:105, E:106, F:1, C:20),
  quantity_in_stock(S:107, E:108, F:1, C:21), attributes(S:110,
  E:131, F:1, C:0), ISBN(S:111, E:112, F:1, C:22),
  number_of_pages(S:113, E:114, F:1, C:23), type_of_book(S:115,
  E:116, F:1, C:24), size_of_book(S:117, E:130, F:1, C:0),
  length(S:118, E:121, F:1, C:25), unit(S:119, E:120, F:1, C:26),
  width(S:122, E:125, F:1, C:27), unit(S:123, E:124, F:1, C:28),
  height(S:126, E:129, F:1, C:29), unit(S:127, E:128, F:1, C:30)
Cluster 2 (S:7, E:40, F:4)
  Member:  author(S:7, E:40, F:4, C:0), name(S:8, E:15, F:1, C:0),
  first_name(S:9, E:10, F:1, C:1), middle_name(S:11, E:12, F:1,
  C:2), last_name(S:13, E:14, F:1, C:3), date_of_birth(S:16, E:17,
  F:1, C:4), biography(S:18, E:19, F:1, C:5),
  contact_information(S:20, E:39, F:1, C:0), mailing_address(S:21,
  E:34, F:1, C:0), street_information(S:22, E:25, F:1, C:0),
  name_of_city(S:26, E:27, F:1, C:6), name_of_state(S:28, E:29,
  F:1, C:7), zip_code(S:30, E:31, F:1, C:8), name_of_country(S:32,
  E:33, F:1, C:9), phone_number(S:35, E:36, F:1, C:10),
  email_address(S:37, E:38, F:1, C:11)
Cluster 3 (S:23, E:24, F:2)
  Member:  street_address(S:23, E:24, F:2, C:1)
Cluster 4 (S:50, E:51, F:2)
  Member:  street_address(S:50, E:51, F:2, C:1)
Cluster 5 (S:81, E:84, F:5)
  Member:  related_item(S:81, E:84, F:5, C:0), item_id(S:82, E:83,
  F:1, C:1)
```

3. Node Configuration

```
<Nodes>
  <Node  name= "catalog" S="0" E="133" T="E" CID="0" CNO="0">
    <Node  name= "item" S="1" E="132" T="E" CID="1" CNO="0">
      <Node  name= "id" S="2" E="3" T="A" CID="1" CNO="1"/>
      <Node  name= "title" S="4" E="5" T="E" CID="1" CNO="2"/>
      <Node  name= "authors" S="6" E="41" T="E" CID="1" CNO="0">
        <Node  name= "author" S="7" E="40" T="E" CID="2" CNO="0">
          <Node  name= "name" S="8" E="15" T="E" CID="2" CNO="0">
            <Node  name= "first_name" S="9" E="10" T="E" CID="2" CNO="1"/>
            <Node  name= "middle_name" S="11" E="12" T="E" CID="2" CNO="2"/>
            <Node  name= "last_name" S="13" E="14" T="E" CID="2" CNO="3"/>
          </Node>
          <Node  name= "date_of_birth" S="16" E="17" T="E" CID="2" CNO="4"/>
          <Node  name= "biography" S="18" E="19" T="E" CID="2" CNO="5"/>
          <Node  name= "contact_information" S="20" E="39" T="E" CID="2" CNO="0">
            <Node  name= "mailing_address" S="21" E="34" T="E" CID="2" CNO="0">
              <Node  name= "street_information" S="22" E="25" T="E" CID="2" CNO="0">
                <Node  name= "street_address" S="23" E="24" T="E" CID="3" CNO="1"/>
              </Node>
              <Node  name= "name_of_city" S="26" E="27" T="E" CID="2" CNO="6"/>
              <Node  name= "name_of_state" S="28" E="29" T="E" CID="2" CNO="7"/>
              <Node  name= "zip_code" S="30" E="31" T="E" CID="2" CNO="8"/>
              <Node  name= "name_of_country" S="32" E="33" T="E" CID="2" CNO="9"/>
            </Node>
            <Node  name= "phone_number" S="35" E="36" T="E" CID="2" CNO="10"/>
            <Node  name= "email_address" S="37" E="38" T="E" CID="2" CNO="11"/>
          </Node>
        </Node>
      </Node>
      <Node  name= "date_of_release" S="42" E="43" T="E" CID="1" CNO="3"/>
      <Node  name= "publisher" S="44" E="75" T="E" CID="1" CNO="0">
        <Node  name= "name" S="45" E="46" T="E" CID="1" CNO="4"/>
        <Node  name= "contact_information" S="47" E="74" T="E" CID="1" CNO="0">
          <Node  name= "mailing_address" S="48" E="67" T="E" CID="1" CNO="0">
            <Node  name= "street_information" S="49" E="52" T="E" CID="1" CNO="0">
              <Node  name= "street_address" S="50" E="51" T="E" CID="4" CNO="1"/>
            </Node>
            <Node  name= "name_of_city" S="53" E="54" T="E" CID="1" CNO="5"/>
            <Node  name= "name_of_state" S="55" E="56" T="E" CID="1" CNO="6"/>
            <Node  name= "zip_code" S="57" E="58" T="E" CID="1" CNO="7"/>
            <Node  name= "country" S="59" E="66" T="E" CID="1" CNO="0">
              <Node  name= "name" S="60" E="61" T="E" CID="1" CNO="8"/>
              <Node  name= "exchange_rate" S="62" E="63" T="E" CID="1" CNO="9"/>
              <Node  name= "currency" S="64" E="65" T="E" CID="1" CNO="10"/>
            </Node>
          </Node>
          <Node  name= "FAX_number" S="68" E="69" T="E" CID="1" CNO="11"/>
          <Node  name= "phone_number" S="70" E="71" T="E" CID="1" CNO="12"/>
          <Node  name= "web_site" S="72" E="73" T="E" CID="1" CNO="13"/>
        </Node>
      </Node>
      <Node  name= "subject" S="76" E="77" T="E" CID="1" CNO="14"/>
      <Node  name= "description" S="78" E="79" T="E" CID="1" CNO="15"/>
      <Node  name= "related_items" S="80" E="85" T="E" CID="1" CNO="0">
        <Node  name= "related_item" S="81" E="84" T="E" CID="5" CNO="0">
          <Node  name= "item_id" S="82" E="83" T="E" CID="5" CNO="1"/>
        </Node>
      </Node>
      <Node  name= "media" S="86" E="95" T="E" CID="1" CNO="0">
        <Node  name= "thumbnail" S="87" E="90" T="E" CID="1" CNO="0">
          <Node  name= "data" S="88" E="89" T="E" CID="1" CNO="0"/>
        </Node>
        <Node  name= "image" S="91" E="94" T="E" CID="1" CNO="0">
          <Node  name= "data" S="92" E="93" T="E" CID="1" CNO="0"/>
        </Node>
      </Node>
      <Node  name= "pricing" S="96" E="109" T="E" CID="1" CNO="0">
        <Node  name= "suggested_retail_price" S="97" E="100" T="E" CID="1" CNO="16">
```

```
          <Node  name= "currency" S="98" E="99" T="A" CID="1" CNO="17"/>
        </Node>
        <Node  name= "cost" S="101" E="104" T="E" CID="1" CNO="18">
          <Node  name= "currency" S="102" E="103" T="A" CID="1" CNO="19"/>
        </Node>
        <Node  name= "when_is_available" S="105" E="106" T="E" CID="1" CNO="20"/>
        <Node  name= "quantity_in_stock" S="107" E="108" T="E" CID="1" CNO="21"/>
      </Node>
      <Node  name= "attributes" S="110" E="131" T="E" CID="1" CNO="0">
        <Node  name= "ISBN" S="111" E="112" T="E" CID="1" CNO="22"/>
        <Node  name= "number_of_pages" S="113" E="114" T="E" CID="1" CNO="23"/>
        <Node  name= "type_of_book" S="115" E="116" T="E" CID="1" CNO="24"/>
        <Node  name= "size_of_book" S="117" E="130" T="E" CID="1" CNO="0">
          <Node  name= "length" S="118" E="121" T="E" CID="1" CNO="25">
            <Node  name= "unit" S="119" E="120" T="A" CID="1" CNO="26"/>
          </Node>
          <Node  name= "width" S="122" E="125" T="E" CID="1" CNO="27">
            <Node  name= "unit" S="123" E="124" T="A" CID="1" CNO="28"/>
          </Node>
          <Node  name= "height" S="126" E="129" T="E" CID="1" CNO="29">
            <Node  name= "unit" S="127" E="128" T="A" CID="1" CNO="30"/>
          </Node>
        </Node>
      </Node>
    </Node>
  </Node>
</Nodes>
```

APPENDIX C
QUERY DETAIL

1. Node ID

ID	NODE
0	<Node name= "catalog" S="0" E="133"/>
1	<Node name= "item" S="1" E="132”/>
2	<Node name= "id" S="2" E="3”/>
3	<Node name= "title" S="4" E="5”/>
4	<Node name= "authors" S="6" E="41”>
5	<Node name= "author" S="7" E="40">
6	<Node name= "name" S="8" E="15">
7	<Node name= "first_name" S="9" E="10”/>
8	<Node name= "middle_name" S="11" E="12"/>
9	<Node name= "last_name" S="13" E="14"/>
10	<Node name= "date_of_birth" S="16" E="17"/>
11	<Node name= "biography" S="18" E="19"/>
12	<Node name= "contact_information" S="20" E="39">
13	<Node name= "mailing_address" S="21" E="34">
14	<Node name= "street_information" S="22" E="25">
15	<Node name= "street_address" S="23" E="24"/>
16	<Node name= "name_of_city" S="26" E="27"/>
17	<Node name= "name_of_state" S="28" E="29"/>
18	<Node name= "zip_code" S="30" E="31"/>
19	<Node name= "name_of_country" S="32" E="33"/>
20	<Node name= "phone_number" S="35" E="36"/>
21	<Node name= "email_address" S="37" E="38"/>
22	<Node name= "date_of_release" S="42" E="43"/>
23	<Node name= "publisher" S="44" E="75">
24	<Node name= "name" S="45" E="46"/>
25	<Node name= "contact_information" S="47" E="74">
26	<Node name= "mailing_address" S="48" E="67">
27	<Node name= "street_information" S="49" E="52">

ID	NODE
28	<Node name= "street_address" S="50" E="51"/>
29	<Node name= "name_of_city" S="53" E="54"/>
30	<Node name= "name_of_state" S="55" E="56"/>
31	<Node name= "zip_code" S="57" E="58"/>
32	<Node name= "country" S="59" E="66">
33	<Node name= "name" S="60" E="61"/>
34	<Node name= "exchange_rate" S="62" E="63"/>
35	<Node name= "currency" S="64" E="65"/>
36	<Node name= "FAX_number" S="68" E="69"/>
37	<Node name= "phone_number" S="70" E="71"/>
38	<Node name= "web_site" S="72" E="73"/>
39	<Node name= "subject" S="76" E="77"/>
40	<Node name= "description" S="78" E="79"/>
41	<Node name= "related_items" S="80" E="85">
42	<Node name= "related_item" S="81" E="84">
43	<Node name= "item_id" S="82" E="83"/>
44	<Node name= "media" S="86" E="95">
45	<Node name= "thumbnail" S="87" E="90">
46	<Node name= "data" S="88" E="89"/>
47	<Node name= "image" S="91" E="94">
48	<Node name= "data" S="92" E="93"/>
49	<Node name= "pricing" S="96" E="109”>
50	<Node name= "suggested_retail_price" S="97" E="100">
51	<Node name= "currency" S="98" E="99"/>
52	<Node name= "cost" S="101" E="104">
53	<Node name= "currency" S="102" E="103"/>
54	<Node name= "when_is_available" S="105" E="106"/>
55	<Node name= "quantity_in_stock" S="107" E="108"/>
56	<Node name= "attributes" S="110" E="131">
57	<Node name= "ISBN" S="111" E="112"/>
58	<Node name= "number_of_pages" S="113" E="114"/>
59	<Node name= "type_of_book" S="115" E="116"/>
60	<Node name= "size_of_book" S="117" E="130">
61	<Node name= "length" S="118" E="121">
62	<Node name= "unit" S="119" E="120"/>
63	<Node name= "width" S="122" E="125">
64	<Node name= "unit" S="123" E="124"/>

ID	NODE
65	<Node name= "height" S="126" E="129">
66	<Node name= "unit" S="127" E="128"/>

2. Detail of The Query

Workload group	Query	Related Node
G1 (1 node)	Q1.1	1
	Q1.2	5
	Q1.3	15
	Q1.4	28
	Q1.5	43
G2 (10 nodes)	Q2.1	1,2,3,4,22,23,24,39,49,52
	Q2.2	1,3,4,5,6,7,10,11,22,39
G3 (20 nodes)	Q3.1	1,4,5,6,9,10,11,12,20,21,23,25,26,27,29,32,33,36,37,41
	Q3.2	1,2,3,4,22,23,24,25,26,36,37,38,32,33,34,39,56,57,58,59
	Q3.3	1,3,4,5,12,13,14,15,23,25,26,27,28,41,42,43,22,24,10,11
G4 (30 nodes)	Q4.1	1,3,4,22,23,24,25,26,27,29,30,31,32,33,34,35,36,37,38,39,40, 41,44,45,46,47,48,49,50,52
	Q4.2	1,3,4,5,6,7,8,9,10,11,12,13,16,17,18,19,22,44,45,46,49,50,52, 54,55,56,57,58,60,61
	Q4.3	1,3,4,5,6,7,8,9,10,11,12,13,14,15,20,21,22,23,24,25,26,27,28, 41,42,43,49,50,51,52
G5 (40 nodes)	Q5.1	1,3,22,23,24,25,26,27,29,30,31,32,33,34,35,36,37,38,39,40, 44,45,46,47,48,49,50,51,52,53,54,55,56,60,61,62,63,64,65,66
	Q5.2	1,3,4,5,6,7,8,9,10,11,12,13,16,17,18,19,22,23,25,26,29,30,44, 45,46,49,50,52,54,55,56,57,58,60,61,62,63,64,65,66
	Q5.3	1,3,4,5,12,13,14,15,16,17,18,19,23,25,26,27,28,41,42,43,22, 24,10,11,6,7,8,9,20,21,44,45,46,49,50,51,52,53,56,57
G6 (50 nodes)	Q6.1	1,3,4,5,6,7,8,9,10,11,12,13,16,17,18,19,22,23,25,26,29,30,31, 32,33,34,35,36,37,38,44,45,46,47,48,49,50,52,54,55,56,57,58, 60,61,62,63,64,65,66
	Q6.2	1,3,4,5,6,7,8,9,10,11,12,13,14,15,18,19,22,23,25,26,27,28,31, 32,33,34,35,36,37,38,41,42,43,44,45,46,47,48,49,50,52,54,56, 60,61,62,63,64,65,66

Workload group	Query	Related Node
G7 (60 nodes)	Q7.1	1,2,3,4,5,6,7,8,9,10,11,12,13,16,17,18,19,20,21,22,23,25,26, 27,29,30,31,32,33,34,35,36,37,38,39,40,41,44,45,46,47,48,49, 50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66
	Q7.2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,19,20,21,22,23,25,26, 27,28,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,47,48,49, 50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66

3. Querying Time Comparison

Query Number	Querying Time Comparison (secs)		
	Table-based	Element-based	Subtree-based
Q1.1	0.2533	0.0533	0.7100
Q1.2	0.4467	0.1033	0.7067
Q1.3	0.2933	0.2767	0.3300
Q1.4	0.1233	0.1067	0.1100
Q1.5	0.1400	0.1433	0.1833
Q2.1	0.3967	0.4500	0.7933
Q2.2	1.1633	1.1833	1.8933
Q3.1	1.5700	1.8133	2.1633
Q3.2	0.5633	0.9100	0.9567
Q3.3	1.7300	2.0533	2.6967
Q4.1	0.8167	1.6100	1.1767
Q4.2	1.8300	2.6100	2.4267
Q4.3	2.0933	2.9900	3.1033
Q5.1	2.0933	2.9900	3.1033
Q5.2	0.9500	1.9067	1.3267
Q5.3	1.8833	3.1767	2.5833
Q6.1	2.4500	3.7100	3.3100
Q6.2	2.0800	3.7267	2.8133
Q7.1	2.3867	3.9900	3.2733
Q7.2	2.3633	4.4800	3.1367

BIOGRAPHY

NAME Mr. Nattapong Thawornkool

DATE June 28, 1976

PLACE OF BIRTH Bangkok, Thailand

INSTITUTIONS ATTENDED

Kasetsart University, 1994-1997:
Bachelor of Science (Computer Science)

Mahidol University, 2000-2005:
Master of Science (Computer Science)

POSITION & OFFICE

Metropolitan Waterworks Authority
Bangkok, Thailand
Position: 5th Grade Computer Officer