

FUZZY QUERYING IN XML DATABASES

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY**

BY

EKİN ÜSTÜNKAYA

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE**

IN

COMPUTER ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Adnan Yazıcı
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Ferda Nur Alpaslan(Chairman) (METU, CENG)_____

Prof. Dr. Adnan Yazıcı (METU, CENG)_____

Assoc. Prof. Dr. Nihan Kesim Çiçekli (METU, CENG)_____

Dr. Ayşenur Birtürk (METU, CENG)_____

Levent Çarkacıoğlu, M.S. (T.C.M.B.) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ekin Üstünkaya

ABSTRACT

FUZZY QUERYING IN XML DATABASES

ÜSTÜNKAYA, Ekin

M.S., Department of Computer Engineering

Supervisor: Prof. Dr. Adnan Yazıcı

December 2004, 91 pages

Real-world information containing subjective opinions and judgments has emerged the need to represent complex and imprecise data in databases. Additionally, the challenge of transferring information between databases whose data storage methods are not compatible has been an important research topic. Extensible Markup Language (XML) has the potential to meet these challenges since it has the ability to represent complex and imprecise data.

In this thesis, an XML based fuzzy data representation and querying system is designed and implemented. The resulting system enables fuzzy querying on XML documents by using XQuery, a language used for querying XML documents. In the system, complex and imprecise data are represented using XML combined with the

fuzzy representation. In addition to fuzzy querying, the system enables restructuring of XML Schemas by merging of elements of the XML documents. By using this feature of the system, one can generate a new XML Schema and new XML documents from the existing documents according to this new XML Schema. XML data used in the system are retrieved from Internet by Web Services, which can make use of XML's capabilities to transfer data and, XML documents are stored in a native XML database management system.

Keywords: Fuzzy Query, XML, XML Databases

ÖZ

XML VERİTABANLARINDA BULANIK SORGULAMA

ÜSTÜNKAYA, Ekin

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Adnan Yazıcı

Aralık 2004, 91 sayfa

Gerçek dünyadaki bilgilerin öznel kanı ve yargılar içermesi, kompleks ve belirsiz verilerin veritabanlarında temsil edilmesi ihtiyacını doğurmuştur. Buna ek olarak, veri depolama teknikleri birbiriyle uyumsuz olan veritabanları arasındaki bilgi transferi de önemli bir araştırma konusu oluşturmaktadır. Extensible Markup Language (XML), kompleks ve belirsiz verileri temsil etme yeteneği ile bu konularda çözüm olma potansiyeline sahiptir.

Bu çalışmada, XML tabanlı bir bulanık veri sunumu ve sorgulama sistemi tasarlanmış ve gerçekleştirilmiştir. Sonuçta geliştirilmiş olan sistem, XML dokümanlarını sorgulama amaçlı kullanılan XQuery dilini kullanarak bulanık

sorgulama yapmayı sağlamaktadır. Sistemde kompleks ve belirsiz veriler bulanık betimlemeyle birleştirilerek, XML kullanılarak temsil edilmiştir. Bulanık sorgulamaya ek olarak, sistem, XML dokümanlarının elemanlarının birleştirilmesiyle, XML şemalarının yeniden yapılandırılmasına olanak verir. Sistemin bu özelliği kullanılarak yeni bir XML şeması ve varolan XML dokümanlarından bu şemaya uygun yeni XML dokümanları yaratılabilir. Sistemde kullanılan XML verileri, XML'in veri taşımadaki yeteneklerini kullanabilen Web Servisleri aracılığıyla, İnternet'den alınmıştır ve XML dokümanları yerel bir XML veritabanı sisteminde saklanmaktadır..

Anahtar Kelimeler: Bulanık Sorgulama, XML, XML Veritabanları

ACKNOWLEDGEMENTS

I would like to thank my supervisor Adnan Yazıcı for all his guidance and support during this study.

I would also like to thank my parents and my brother, for all their support during this study.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF FIGURES	xi
LIST OF TABLES	xiii
CHAPTER	
1. INTRODUCTION	1
2. THEORETICAL BACKGROUND	6
2.1 XML and XML Technologies	6
2.1.1 XML (Extensible Markup Language)	6
2.1.2 XML Querying Languages	8
2.1.3 Stylesheet for XML	13
2.1.4 Document Object Model (DOM)	14
2.2 ExNF ² Model	15
2.2.1 The Extended NF ² Algebra	18
2.2.2 Basic Set Operations	19
2.2.3 Restructuring Operations	22
2.3 XML Databases	26
2.4 Fuzzy Querying	31
2.5 Web Services	35

3. DESIGN AND IMPLEMENTATION OF XML-BASED FUZZY QUERYING SYSTEM	37
3.1 System Objectives	37
3.2 Modeling Complex and Uncertain Data in XML.....	37
3.3 System Architecture	42
3.4 Data Objects & Database Design	44
3.5 Implementation.....	46
3.5.1 Retrieving Data through Web Services	47
3.5.2 Mapping of the Data to the XML Schema	48
3.5.3 Querying the Data.....	48
3.5.3.1 Fuzzy Queries	56
3.5.4 Merging XML Schemas	60
3.6 Programming Environment	64
4. CONCLUSION	65
REFERENCES.....	67
APPENDICES	
A. AN EXAMPLE XML DOCUMENT FOR A PRODUCT	69
B. THE XML SCHEMA CODE FOR PRODUCTS	70
C. XML DOCUMENT FOR COLOUR SIMILARITY TABLE	72
D. THE XML SCHEMA CODE FOR COLOUR SIMILARITY TABLE	74
E. XML DOCUMENT FOR SUBJECT SIMILARITY TABLE	75
F. THE XML SCHEMA CODE FOR SUBJECT SIMILARITY TABLE.....	78

LIST OF FIGURES

FIGURE

Figure 1: An Example XML Document	7
Figure 2: DTD Representation of Product Information	8
Figure 3: Sample XML-QL Query	9
Figure 4: Sample XPath Expressions	10
Figure 5: XQuery Example	12
Figure 6: XSL Document Format	14
Figure 7: Sales Order Document	30
Figure 8: Example XML for Atomic Values	38
Figure 9: Example XML for Empty Values	39
Figure 10: Example XML for Set-Valued Attributes	39
Figure 11: Example XML for Fuzzy-Valued Attributes	40
Figure 12: Example XML for Range-Valued Attributes	41
Figure 13: Example XML for Relation-Valued Attributes	41
Figure 14: System Architecture	42
Figure 15: XML Schema Representation of a Product	45
Figure 16: XML Schema Representation of Colour Similarity Table	46
Figure 17: XML Schema Representation of Subject Similarity Table	46
Figure 18: Call and Request Format of Amazon Web Services	48
Figure 19: Architectural Overview of the Query Processing	49
Figure 20: XQuery Example(1)	50

Figure 21: Query Results for Query Example(1)	50
Figure 22: XQuery Example(2)	51
Figure 23: Query Results for Query Example(2)	52
Figure 24: Details Form	55
Figure 25: XML Representation of Book-“Deception Point”	57
Figure 26: “OR” Predicate Query Example	58
Figure 27: “AND” Predicate Query Example (1)	59
Figure 28: “AND” Predicate Query Example (2)	60
Figure 29: Merge Schema Form	61
Figure 30: XML Schema Representation of a PublisherInfoSchema	62
Figure 31: XML Schema Structure Before Merging	63
Figure 32: New XML Schema Structure	63
Figure 33: XML Schema Structure Regarding Primary Key	64

LIST OF TABLES

TABLE

Table 1: Employee Relation	18
Table 2: Similarity Matrix for Age Attribute	34
Table 3: Explanation of Fields on the Query Form	53
Table 4: Explanation of Fields on the Details Form	56

CHAPTER 1

INTRODUCTION

A database is any organized collection of information. There are many traditional database management systems that can represent crisp data using well-understood structures. However, real-world information containing subjective opinions and judgments may contain complex and imprecise data along with crisp data. The representation of such uncertain and complex data, in a database still, has been a research issue. One of the proposed models for the representation of uncertain and complex data is the Extended Non First Normal Form (ExNF²) data model, an extended relational database model [1].

Another important issue is the sharing of information between databases. Because different databases store data in different, and sometimes incompatible, formats, which makes exchanging information a challenge. In many businesses, data from a large number of heterogeneous databases need to be integrated in connection with data warehousing, or system integration. Many organizations and enterprises establish distributed working environments, where different users need to exchange information based on a common model. The Information Brokerage project is an example which is built around an open systems architecture developed to allow a

collaborating group of legacy systems, based on heterogeneous database and server platforms, to offer an integrated query service over the Internet [25].

Extensible Mark-up Language (XML) is a proposed solution as a data representation and exchange format through the Internet and different database models to meet this challenge. The Lore system is one example of a database model implemented in XML [2]. Unlike HTML, XML allows the separation of content and presentation, that is, XML documents simply define the data representation and do not deal with the presentation. XML can also be used to represent complex and imprecise data formats in addition to crisp data formats. Not only can XML process complex and hierarchical information, it can also be used for commercial transactions. Therefore XML documents can be used to transfer data between the Internet applications and different database models.

In order to make use of advantages of XML, information in XML documents must be made available quickly, reliably and in large amounts when necessary, for transactions. As is true for management of other forms of data, management of persistent XML data requires capabilities for data independence, integration, access rights, versions, views, integrity, redundancy, consistency, and recovery standards. Functionality, consistency, restart capability, data security, and recovery tools of a database management system can be utilized by the XML data.

An approach for integrating XML into conventional database technologies would be to implement XML structures within data models, such as the relational model [3].

To achieve this, XML-formatted objects have to be converted before they fit into relational database structures, so it is necessary to map the XML document schema (DTD [4], XML Schemas [5], etc.) to the database schema. Mappings between document schemas and database schemas are performed on element types, attributes, and text. Object-oriented databases (ODBMS) are another alternative for storing XML information. XML-formatted objects also have to be converted before they fit into object-oriented databases.

It is also possible to store data in XML documents in a native XML database management system. Native XML databases use some storage strategies to increase database retrieval speed. Native XML databases store entire documents physically or use physical pointers between the parts of the document. Choosing the most appropriate database management system depends on your application and data structure.

With a large amount of data represented as XML documents, it becomes necessary to store and query these XML documents. To better query XML data, several XML query languages have been proposed. XML-QL [6], XPath [7] and XQuery [8] are among XML querying languages. One of the great strengths of XML is its flexibility in representing many different kinds of information from diverse sources. To exploit this flexibility, an XML query language must provide features for retrieving and interpreting information from these diverse sources. XQuery [8] language has been endorsed by W3C as a standard for XML querying.

Traditional query languages in database domain allow data selection based on precise data. The evaluation of the query produces a clear-cut partition of the target data those fully satisfy the query condition and those which do not. In general, information may be both complex and imprecise when representing personalities, physical features of individuals, subjective opinions, and judgments concerning medical diagnosis, economic forecasting or personal evaluation and in many other knowledge-intensive applications [1]. A vague predicate represented by a fuzzy set expresses a softer condition. In other words, fuzzy querying allows the user to define goals and constraints, and provide them with relative importance weights on a scale of 1 (least) to 9 (most).

In this thesis, methods to map complex and uncertain data to an XML representation are discussed. The structure of XML so closely resembles that of traditional relational database tuples, therefore the mapping from 1NF tuples to XML elements is almost trivial. However, the ExNF² model [1] introduces extensions for representing complex, uncertain and fuzzy data not so easily represented in XML. In this work, XML documents are adapted to include elements and attributes for representing imprecise and complex data formats.

In this thesis, a system, which enables fuzzy querying in XML documents is designed and implemented. XML documents in the system contain fuzzy attributes. The user can select the fuzzy attribute and threshold value for it and then perform the query. In addition to fuzzy querying, the system enables restructuring of XML Schemas by merging of elements of the XML documents. By using this feature of

the system, one can generate a new XML Schema and new XML documents are generated from the existing documents according to this new XML Schema. XML documents to be queried are retrieved from Internet via Web Services. XML documents are stored in a native XML database management system.

The following summarizes the organization of the thesis. Chapter 2 gives theoretical background information on XML, XML technologies, representation of uncertainty in databases, fuzzy querying and Web Services. Chapter 3 details the design and implementation of the work done in this study. And finally Chapter 4 covers the conclusion.

CHAPTER 2

THEORETICAL BACKGROUND

2.1 XML and XML Technologies

XML is a markup language for documents containing structured information. The XML world is made up of a series of separate “sub-standards” describing various aspects of document representation and reproduction. In this section XML and XML related technologies are explained.

2.1.1 XML (Extensible Markup Language)

XML is a meta-language, which can be used to describe the logical structure of documents and data, using tag names and attribute names. Unlike HTML, XML allows users to define their own tags. As a result, data can be structured not only according to format criteria (such as header, body text, etc.), but also by referring to its content. An example XML document is shown in Figure1.

```
<?xml version="1.0" ?>
<!--data begins here-->
<product>
    ....
    <name> Karce</name>
    <price valid-from = "15/10/2004">14,95 </price>
    .....
</product>
```

Figure 1: An Example XML Document

When XML documents are used in data exchange, there should be a consensus on the usage of element names and their hierarchical relations. This means some constraints need to be declared on XML documents for avoiding confusions on the usage of elements and attributes. DTD (Document Type Definition) [4] documents are used to avoid such confusions. DTDs are schemas for defining document types [3]. If an XML document is declared to be conforming to a DTD, metadata in the XML should satisfy the constraints declared in the DTD. Roughly a DTD defines which tag names and attribute names can be used and in what order they should occur in an XML document. DTDs are laid down at the time the XML applications are developed. XML documents can also be processed without a DTD, but in this case the structure information stored in there is lost. DTDs are normally only used to control XML tools and to verify the structural validity of XML documents – they are not necessary to understand these documents. DTD representation of product information is provided in Figure 2.


```
<!ELEMENT product(name,description?,product-number, availability?,price?)>
  <!ELEMENT name(#PCDATA)>
  <!ELEMENT description(#PCDATA)>
  <!ELEMENT product-number(#PCDATA)>
  <!ELEMENT availability(#PCDATA)>
  <!ELEMENT price(#PCDATA)>
  <!ATTLIST product price valid-from #required>
```

Figure 2: DTD Representation of Product Information

XML Schema [5] is another information modeling method for XML documents developed by W3C. Although XML DTDs (Document Type Definitions) allow the tags and structure of a document class to be specified, the content of document elements and the values of attributes are plain text (strings) [3]. The XML Schema introduces types such as number, date, time, etc. into XML and also permits user defined data types. The XML Schema additionally supports modularization, which makes a schema easier to reuse. XML Schemas are written in XML and can thus be processed using XML tools.

2.1.2 XML Querying Languages

The need for querying XML documents is emerged after XML documents are started to use for data storing. As increasing amounts of information are stored, exchanged, and presented using XML, the ability to intelligently query XML data sources becomes increasingly important. Several query languages for querying XML documents are proposed so far.

XML-QL [6] is one of those query languages, which is proposed for querying XML documents. The significant characteristics in its syntax to other languages are its SELECT-WHERE construct, like SQL. The basic idea is to provide a syntax to locate nodes (elements and text) within an XML document, using a notation inspired by directory path expressions. A sample query in XML-QL is given in Figure 3. The query searches the name the products for book whose “Publication_Date” attribute is equal to 2004.

```
WHERE
<Product >
  <Name >$b</Name>
  <Book Publication_Date="2004">$a</Book>
</Product>
IN "product.xml"
CONSTRUCT
<Result>
  <Product> $b </Product>
</Result>
```

Figure 3: Sample XML-QL Query

Another language, XPath [7] provides a common syntax and semantics for functionality shared between XSL Transformations [9] and XPointer [10]. The main functionality of XPath is to acquire parts of an XML document. It also provides basic facilities for manipulation of strings, numbers and Boolean. XPath uses a non-XML syntax to make it easy to use of XPath within URIs (Uniform Resource Identifiers) and XML attribute values. The primary structure in XPath is the expression. An expression is computed to obtain an object, which can be one of the

following basic types: node-set, Boolean, floating-point number, string. Figure 4 shows examples of XPath expressions used in an XSL document.

```
<!--Match the element named A nested within the B element -->
<xsl:value-of select="A/B"/>
<!--Match the C attribute of the B element -->
<xsl:value-of select ="B/@C"/>
```

Figure 4: Sample XPath Expressions

Another proposed query language is XQuery [8], which involves features taken from several other languages. XQuery is designed by the W3C XML Query Working Group to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. XQuery is a flexible, functional language that allows one to generate query expressions, which are often composed of many other expressions [3].

The W3C Query Working Group has identified requirements [11] for W3C XML Query (XQuery) data model, algebra, and query language. The requirements for XQuery can be summarized as follows:

- at least one XML syntax (at least one human-readable syntax)
- must be declarative
- must be protocol independent
- must respect XML data model

- must be namespace aware
- must coordinate with XML Schema
- must work even if schemas are unavailable
- must support simple and complex data types
- must support universal and existential quantifiers
- must support operations on hierarchy and sequence of document structures
- must combine information from multiple documents
- must support aggregation
- must be able to transform and to create XML structures
- must be able to traverse ID references

XQuery inherits path expression syntax suitable for hierarchical documents from XPath and XQL [12]. It takes the binding variables concept and then using the bound variables to create new structures from XML-QL. It has series of clauses based on keywords that provide a pattern for restructuring data similar to the SELECT-FROM-WHERE pattern in SQL. The basic building block of XQuery is the expression, which is a string of Unicode characters [8]. A FLWR expression (the letters in FLWR stand for the XQuery keywords *for*, *let*, *where*, and *return*) iterates over a sequence of items and binds variables that can be used in the scope of the current expression. If the item sequence is empty, the result of the FLWR expression is an empty sequence. A FLWR expression consists of one or more *for* and *let* clauses in any combination, followed by an optional *where* clause, and a *return* clause. Briefly, these clauses are interpreted as follows:

- A *for* clause binds one or more variables to each value of the result of the following expression.
- A *let* clause binds one or more variables to the complete result of the expression.

- A where clause retains only those intermediate results that satisfy the following condition.
- A return clause evaluates the following expression and returns the result.

XQuery is a functional language, which means that expressions can be nested with full generality. However, unlike a pure functional language, it does not allow variable substitutability if the variable declaration contains construction of new nodes. XQuery is also a strongly-typed language in which the operands of various expressions, operators, and functions must conform to the expected types [8].

Figure 5 shows an XQuery example. This query returns all book/title elements of the collection together with the year of publication provided that the year of publication is 2000 or later.

```
for $b in input()/bib/book
let $y := $b/year where $y > 2000
RETURN
<book>
  { $y }
  { $a/title }
</book>
```

Figure 5: XQuery Example

2.1.3 Stylesheet for XML

The layout of an XML document is not defined in the document itself or in its DTD or XML Schema. It is one of the fundamental principles of XML that content should be absolutely separate from presentation. Extensible Stylesheet Language (XSL) was proposed in order to specify how the elements of an XML document would look like when shown in an output media, such as screen display, printouts, etc.

XSL involves two main functionalities. It is a language for transforming XML documents and it is an XML vocabulary for specifying formatting semantics. It can be used to transform XML documents to other formats like HTML. XSLT [13] (XSL for Transformations) is a subset of XSL designed for transformation of XML documents into other XML documents. The second functionality is similar to CSS (Cascading Style Sheets) [14], but it has more features than CSS. Although CSS provides a mechanism for how the elements are displayed, it has not enough capabilities for controlling what to display and in what order to display. On the other hand XSL provides reordering the elements of the document tree and showing the desired elements. Figure 6 shows the format of an XSL document.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="[XPath Expression]">
    <!-- Define your rules for transformation here -->
</xsl:template>
...
</xsl:stylesheet>
```

Figure 6: XSL Document Format

As shown in Figure 6, an XSL document consists of a set of XSL templates. Each template element defines a set of rules and these rules are applied to XML document when the specified XPath expression is satisfied.

2.1.4 Document Object Model (DOM)

W3C's Document Object Model (DOM) is a standard Application Programming Interface (API) to the structure of documents; it aims to access components and to delete, add, or edit their content, attributes and style [15]. DOM provides writing applications, which work properly on all browsers and servers and on all platforms.

The DOM represents a tree view of the XML document. The *documentElement* is the top-level of the tree. This element has one or many *childNodes* that represent the branches of the tree. A *Node Interface Model* is used to access the individual elements in the node tree. As an example, the *childNodes* property of the *documentElement* can be accessed with a for/each construct to enumerate each individual node.

2.2 ExNF² Model

Normal Forms are introduced by Codd for database design [17]. The First Normal Form (1NF) restricts attribute values to atomic values. The Non-First Normal Form (NF2) relaxes this restriction to allow relation-valued and non-atomic values for attributes.

For representing uncertain data in a NF² database, ExNF² model has been introduced by Yazici et. al [1]. For example, if a person's telephone number is not available, that field has to be left blank or defined as null. If a person doesn't have a phone, then the field is again left blank. If we don't know whether the number is unavailable at the time, or if a number does not exist at all, then, the field is left blank again. The resulting ambiguity as to how to interpret this blank field is where the NF² database schema falls short [1].

Another problem with NF² databases is dealing with multi-valued fields. It is a problem for representation when a person has two or three telephone numbers. One solution is to use multiple tuples, which leads to other concerns. For example, if information for the person changes, then this update has to be reflected in all tuples related to the person, which is inefficient. Another concern is dealing with tuples with more than one multi-valued field. Then many tuples are required to represent this information.

Representing uncertainty within field data and queries is another limitation of NF² database schemas. It is difficult to perform a query to find all the "young" employees of a company based on a numerical value.

Complex and uncertain data can be represented in databases by extending the NF² model. This model, the Extended Non-First Normal Form (ExNF²) [1], provides the extensions necessary to include uncertainty in the model. An extended NF² relation can be defined as follows [1]:

Definition: Let $Sch\ R$ be a relation schema of relation R with attributes (A_1, A_2, \dots, A_n) . Each attribute A_j may be simple or set-valued or fuzzy-valued or range-valued or relation-valued which are all defined below. Again D_1, D_2, \dots, D_n is a finite set of domains. Let r , an instance of R , be composed of a set of ordered k -tuples of the form $\langle a_1, a_2, \dots, a_n \rangle$, which is a subset of $(D_1 \times D_2 \times \dots \times D_n)$. The domains, D_j ($1 \leq j \leq k$), can be one of the following:

1. D_j is the domain of an atomic-valued attribute. Each value a_j is an element of D_j ; that is, it is a typical simple crisp attribute value.
2. D_j' is the domain of a null-valued attribute. Domain D_j consisting of crisp values $\{a_1, a_2, \dots, a_n\}$ is extended to the domain $D_j' = D_j \cup \{\text{unk}, \text{dne}, \text{ni}\}$.
3. D_j is the domain of an incomplete (range)-valued attribute whose values may be atomic or an interval. Interval representation is $[a_{j1} - a_{j2}]$, where a_{j1} is the minimum and a_{j2} is the maximum of the range. Both values are taken from the domain D_j .
4. D_j is the domain of a fuzzy-valued attribute. The domain subtends a set of fuzzy linguistic terms. A fuzzy attribute value is a nonempty subset of D_j and represented as $[a_{j1}, a_{j2}, \dots, a_{jm}]$.

5. D_j is the domain of a set-valued attribute whose values are crisp sets represented as $\{a_{j1}, a_{j2}, \dots, a_{jm}\}$. Any value of this attribute is a subset of the power set of D_j .
6. D_j is the domain of a relation-valued (composite) attribute. Any value of this attribute, a_j , is a tuple of the form $\langle a_{j1}, a_{j2}, \dots, a_{jm} \rangle$ which is an element of $(D_{j1} \times D_{j2} \times \dots \times D_{jm})$, where $1 < m$ and $1 \leq j \leq k$.

When D_j is the domain of a relation-valued (composite) set attribute where each value a_j is a set of tuples, $\{\langle a_{j1}, a_{j2}, \dots, a_{jm} \rangle, \dots, \langle a_{k1}, a_{k2}, \dots, a_{km} \rangle\}$ which are a subset of $(D_{j1} \times D_{j2} \times \dots \times D_{jm})$, one can form such attribute values by using a combination of domains of relation-valued and set-valued attributes.

Examples of various attribute types discussed above are represented in the extended NF² relations, as shown in Table 1. For example, in the *Employee* relation shown in the Table 1, the *Name* and *SSN* are simple crisp atomic-valued attributes, *TelNo* is a null-valued attribute, *Salary* is an incomplete-valued attribute, *Degrees* is a relation-valued (or composite) attribute, *Languages* is a set-valued attribute, *Age* is a fuzzy-valued attribute.

Table 1: Employee Relation

Employee								
Name	SSN	Telno	Salary	Degrees			Languages	Age
				Univ	GPA	Year		
G. Dark	1781	Dne	[2000-3000]	ITU	3.24	1988	{Fr., Eng.}	[MidYoung, Young]
W. Fischer	1834	2214	[3000-3500]	E.U.	3.40	1990	{Ger.}	[MidYoung]
T. Smith	1858	Ni	[3700]	METU	3.54	1991	{Ger.,Fr., Eng.}	[MidYoung, Old]
Y. Tomson	1979	3327	[3500-3750]	CWRU	2.80	1991	{Eng.}	[Young]

2.2.1 The Extended NF² Algebra

Here the relation names R and S , attributes in $Rel\ A$ and in $Att\ A$, and constants in dom are used [1]. The types are more complex than those of the relational data model. Their abstract syntax is given by:

$$\tau = dom \mid fdom \mid ndom \mid idom \mid \langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle \mid \{\tau_s\},$$

where $\tau_s \neq fdom \mid idom \mid \{\tau\}$ and B_1, \dots, B_m are distinct attributes. Intuitively, an element of dom is a constant (traditional atomic crisp attribute), $fdom$ is the sort of a fuzzy-valued attribute (may form a set with OR semantics) in the form of $[v_1, \dots, v_m]$, where v_i is the sort of dom and an element is any subset of this set; $ndom$ is

the sort of a null-valued attribute in the form of $\{unk, dne, ni, \tau\}$, where τ is of sort dom ; $idom$ is the sort of an incomplete-valued attribute in the form of $[v_1 - v_j]$, where v_i 's are of sort dom and an element is v_i , where $v_1 \leq v_i \leq v_j$; an element of sort $\langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle$ is a k -tuple with an element of sort τ_i in entry B_i , $1 \leq i \leq m$; and an element of sort $\{\tau_i\}$ is a finite set of elements of sort τ . Formally, the set of values of sort τ , (i.e. the interpretation of τ), denoted as $t[\tau]$, is defined as follows:

1. $t[dom] = dom$,
2. $t[fdom] = \{[v_1, \dots, v_j] \mid \forall i: 1 \leq i \leq j : v_i \in t[dom]\}$,
3. $t[ndom] = \{v_i \mid \forall i: 1 \leq i \leq j : v_i \in t[\{unk, dne, ni\} \cup dom]\}$,
4. $t[idom] = \{[v_1 - v_j] \mid \forall i: 1 \leq i \leq j : v_1 \leq v_i \leq v_j, v_i \in t[dom]\}$,
5. $t[\{\tau_s\}] = \{\{v_1, \dots, v_j\} \mid \forall i, j: 1 \leq i \leq j : v_i \in t[\tau_s]\}$. If $t[\{\tau_s\}] = \{\}$, then $v_i = dne$,
6. $t[\langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle] = \{\langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle \mid \forall i: 1 \leq i \leq m : v_i \in t[\tau_i]\}$.

R is a relation name and a database schema consists of a finite set of relation names. A relation over relation name R is a finite set of values of sort(R). An instance I of a schema of R is a function of R , where I is a relation instance over R . When relation R has a sort which is $sort(R) = \langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle$, the relation is composed of the set of tuples, where each τ_i may be one of the above given interpretations.

2.2.2 Basic Set Operations

Let I_1, I_2, \dots, I_n be relations of sort $\tau_1, \tau_2, \dots, \tau_n$, respectively. Assume that relation I_j has a set of values of sort $\tau_j = \langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle$. For $j \in [1, 2]$, if $\tau_1 = \tau_2$, then $I_1 \cap I_2, I_1 \cup I_2, I_1 - I_2$ are relations of sort τ_1 (or τ_2 , since they must be union

compatible.) For more formal definition of the fuzzy equality, let us consider the intersection operator, ' \cap '. $I_3 = I_1 \cap I_2$ results in a set of tuples, $I_3 = \{ t_i \mid \forall t_i : t_i \in I_1 \wedge t_i \in I_2 \}$. The redundant tuples are eliminated from the result relation I_3 by using the fuzzy equality of the attribute values of the tuples, which is the generalization of the ordinary equality. That is, a tuple in a nonfuzzy (crisp) database is redundant if it is exactly the same as another tuple. Any two tuples $t_1 \in I_3$ and $t_2 \in I_3$ in database model are redundant, if $\min_{\forall i} \{ \min(s(t_1[B_i], t_2[B_i])) \} \geq \lambda_i$, where $1 \leq i \leq m$. Informally, any two tuples in the model are redundant, if, for pair of corresponding attribute values, the minimum similarity is greater than or equal to the threshold value, λ_i . Note that in a nonfuzzy database, $\forall i$, the cardinality of $t_i[B_i] = 1$ and $s(x, x) = 1$ so $\lambda_i = 1$. The cases for $I_1 \cup I_2$ and $I_1 - I_2$ are similar to $I_1 \cap I_2$.

Selection: If I is a relation of sort $\tau = \langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle$, (where only one B_i may be the attribute for denoting membership degrees), then $\sigma_\gamma(I)$ is a relation of sort τ . The selection condition γ is of the form $B_i = d$, $B_i = B_j$, $B_i \in B_j$ or $B_i = B_j \cdot C$, where d is a constant value, and it is required in the last case that τ_c be a tuple sort with a C field. Then, $\sigma_\gamma(I) = \{ v \mid v \in I, v \Rightarrow \gamma \}$ where ' \Rightarrow ' stands for logical implication and γ is defined by:

1. $\langle \dots, B_i : v_i, \dots \rangle \Rightarrow B_i = d$ if $v_i = d$, depending on τ_i , the definition of equality is as follows:
 - (a) if $\text{sort}(\tau_i) = \text{dom}$, then it is the traditional equality, i.e., $B_i = d$ if $v_i = d$,
 - (b) if $\text{sort}(\tau_i) = \text{fdom}$, then $B_i = d$ with level value $L(B_i) = \lambda_i$ if $\exists v_i \vee s(v_i, d) \geq \lambda_i$, where $L(B_i)$ is the level value specified in the query,
 - (c) if $\text{sort}(\tau_i) = \text{ndom}$, then $B_i = d$ if $v_i = d$ or $v_i = \text{unk}$,

- (d) if $\text{sort}(\tau_i) = \text{idom}$, then $B_i = d$ if $d \in v_i$.
2. $\langle \dots, B_i : v_i, \dots, B_j : v_j, \dots \rangle \Rightarrow B_i = B_j$ if $v_i = v_j$, where depending on τ_i , the definition of equality is as follows:
- (a) if $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{dom}$, then it is the traditional equality, i.e. $B_i = B_j$ if $v_i = v_j$,
 - (b) if $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{fdom}$, then $B_i = B_j$ with level value $L(B_i) = \lambda_i$, if $\forall x \in v_i, \forall y \in v_j : s(x, y) \geq \lambda_i$,
 - (c) if $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{ndom}$, then $B_i = B_j$ if $v_i = v_j$ (if $v_i = \text{unk}$ and/or $v_j = \text{unk}$, then $B_i = B_j$),
 - (d) if $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{idom}$, then $B_i = B_j$ if $v_i \subseteq v_j$ or $v_i \supseteq v_j$.
3. $\langle \dots, B_i : v_i, \dots, B_j : v_j, \dots \rangle \Rightarrow B_i \in B_j$ if $v_i \in v_j$, where we need to examine the case where $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{fdom}$, in which case we have $B_i \in B_j$ with level value, $L(B_i) = \lambda_i$ if $\forall x \in v_i, \exists y \in v_j : s(x, y) \geq \lambda_i$.
4. $\langle \dots, B_i : v_i, \dots, B_j : v_j, \dots \rangle \Rightarrow B_i \in B_j$, for $\text{sort}(\tau_i) = \text{sort}(\tau_j) = \text{idom}, \forall x \in v_i : x \in v_j$.
5. $\langle \dots, B_i : v_i, \dots, B_j : \langle \dots, C : v_c, \dots \rangle, \dots \rangle \Rightarrow B_i = B_j \cdot C$ if $v_i = v_c$, where the definitions are similar to those in above point 2.

Projection: If I is a relation of sort $\tau = \langle B_1 : \tau_1, \dots, B_m : \tau_m \rangle$, then $\prod_{B_i, \dots, B_g} (I)$, where $1 \leq i$ and $g \leq m$ is a relation of sort $\langle B_1 : \tau_1, \dots, B_g : \tau_g \rangle$. The redundant tuples are eliminated from result relation I after the projection operation by using the fuzzy equality of the attribute values of the tuples; that is, any two tuples $t_1 \in I$ and $t_2 \in I$ are redundant, if $1 \leq i \leq m$ and $\min_{\forall i} \{ \min s(t_1[B_i], t_2[B_i]) \} \leq \lambda_i$.

Cartesian Product: Let I_j be a relation of sort $\tau_j = \langle B_1^j : \tau_1^j, \dots, B_k^j : \tau_k^j \rangle$ for $j \in [1, 2]$. Then $\text{Sort } (I_1 \times I_2) = \langle B_1^1 : \tau_1^1, \dots, B_k^1 : \tau_k^1, B_1^2 : \tau_1^2, \dots, B_k^2 : \tau_k^2 \rangle$ and $I_1 \times I_2 = \{ \langle B_1^1 : x_1^1, \dots, B_k^1 : x_k^1, B_1^2 : x_1^2, \dots, B_k^2 : x_k^2 \rangle \mid \langle B_1^j : \tau_1^j, \dots, B_k^j : \tau_k^j \rangle \in I_j \text{ for } j \in [1, 2] \}$

2.2.3 Restructuring Operations

Two restructuring operators, namely, *Merge* and *Unmerge* are defined. The *Merge* operator can be considered as a combination of *Nest* [18,19,20,21] and *Pack* [22] operators. It may also interact with similarity relationships to manipulate imprecise information. That is, the *Merge* operator can not only change the levels of nesting in a relation as the *Nest* operator does, but it can also be used for uncertain querying. The functionality of the *Unmerge* operator defined here is similar to the *Unnest* operator of [19,20,21].

Merge Operator: For a relation schema *Sch R*, the schema of relation *R* having attributes (A_1, A_2, \dots, A_m) , let each A_j be either a simple, set-valued or higher-order (relation-valued) attribute. Let *Rel A* denote the set of higher-order attributes in *R* and *Att A* be the remaining attributes on the topmost level of an NF^2 relation. In the description of the *Merge* operation that follows, an uncertainty level, λ_j , between zero and one $[0,1]$ may be specified. When it is not specified explicitly it is equal to 1 (one) by default. A level value, $L_j(A_j)$ is chosen based on the following rule:

$$L_j(A_j) = \begin{cases} 0 \text{ or } 1 & \text{if } (A_j \in \text{Att } A \wedge A_j : \{\tau_j\}) \vee (A_j \in \text{Rel } A), \\ \lambda_j & \text{otherwise, } 0 \leq \lambda_j \leq 1, \end{cases}$$

where $A_j : \{\tau_j\}$ means that attribute A_j is set-valued.

This rule states that the level value $L_j(A_j)$, is 1 (one) or 0 (zero) if either attribute A_j is a higher order attribute or a set-valued attribute. When the level value is not

specified, it is assumed to be 1 (one) by default. Otherwise, the level value may be equal to value λ_j . The level value of an attribute is 0 (zero) when one wants to merge all pairs of the values of an attribute. $L_j(A_j)$ for attribute A_j is given a priori and determines which tuples may be combined through the set union of the respective attribute values. This value is specified in the query language by the user and never exceeds the threshold value. The threshold value is the minimum similarity over all values in the domain. The result is obtained by merging as many tuples as possible without violating the constraint, which states that the threshold value for domain D_j (of attribute A_j) is always greater than or equal to $L_j(A_j)$.

Definition: Suppose we have relation S with sort $\text{Sort}(S) = \langle A_1 : \tau_1, \dots, A_i : \tau_i, \dots, A_k : \tau_k, \dots, A_n : \tau_n \rangle$, where the scheme $\text{Sch } S : (A_1, \dots, A_i, \dots, A_k, \dots, A_n)$ and $1 \leq j, k \leq n$. Then for instances s of S , we have the following:

$$\begin{aligned}
R &= \text{MERGE } (S) [A_i, \dots, A_k] \rightarrow B \text{ WITH } L(A_1) = \lambda_j, \dots, L(A_n) = \lambda_n \\
&= \{ \langle A_1 : x_1, \dots, A_{i-1} : x_{i-1}, B : y, \dots, A_{k+1} : x_{k+1}, \dots, A_n : x_n \rangle \mid y \\
&= \langle A_i : x_i, \dots, A_k : x_k \rangle \wedge \langle A_1 : x_1, \dots, A_n : x_n \rangle \in s(S) \}, \\
t'[R] &= t'[\text{Sch}R - (A_i, \dots, A_k) \cup B] \mid t'[B] = \\
&\cup \{ t_j[B] \mid \forall t_i \in s(S) \wedge \forall t_j \in s(S) \wedge t_j[\text{Sch } R - B] \cong t_i[\text{Sch } R - B] \}
\end{aligned}$$

where $1 \leq k \leq n$, $1 \leq i, j \leq k$ and produces a relation r with scheme $\text{Sch}R = \text{Sch}S - (A_i, \dots, A_k) \cup B$, where B is a sub-relation with attributes (A_i, \dots, A_k) and does not occur in the scheme of S . Observe that in the process of merging, the attributes $\langle A_i : x_i, \dots, A_n : x_n \rangle$ are merged for m -similar tuples, i.e., t_i and t_j , depending on the

sort of the respective τ_i 's. The definition of m-similarity that we used in the above definition is as follows:

Definition: (m-similarity, \cong)

$$t_i \cong t_j, \text{ if } (\forall t_i[A_j], \forall t_j[A_j], \min(s(t_i[A_j], t_j[A_j])) \geq L_j(A_j)), \text{ where } t_i \in r \wedge t_j \in r.$$

The level value $L_j(A_j)$ should be specified in the query for the attributes of the relations specified in the query (if it is not specified, it is assumed to be 1 (one) by default). Depending on sort of τ , there are some different cases for m-similarity:

Case 1: If τ is the domain of an atomic crisp attribute or a null-valued attribute, then m-similarity becomes the classical equality. That is,

$$t \text{ and } v \text{ are m-similar iff } t = v; \text{ since } \min(s(t[A_j], v[A_j])) = 1.0.$$

Case 2: If τ is the domain of an incomplete (range)-valued attribute, where $t = [x_1 - x_2]$ and $v = [y_1 - y_2]$, where $x_1 \leq x_2$ and $y_1 \leq y_2$, then there may be two approaches:

- (a) We may require that $x_1 = y_1$ and $x_2 = y_2$ for t and v to be m-similar, which is the case for strict equality.
- (b) t and v are m-similar, when the resulting range-value (needed in the merge), say z , is defined as $z = t \cup v = [z_1 - z_2]$, where $z_1 = \min(x_1, y_1)$ and $z_2 = \min(x_2, y_2)$. It should be noted that in this case, the result of the *Merge* operator may not be reversible by the *Unmerge* operation.

In the definition of the *Merge* operator, if $L_j(A_j) = 1$, then tuples are merged only when the values of all attributes except the one being merged are equivalent. If the values that are being merged are already merged, that is, in the form of sets or a subrelation, then the level value is assigned as 1 (one) by definition, while

restructuring the relation. However, for fuzzy queries, $L_j(A_j) = 0$ is assigned to all of the prime attributes (attributes that participate in a key attribute) and $L_i(A_i) = \lambda_i$ is assigned to uncertain attributes, where λ_i is between 0 (zero) and 1(one). Thus, any pair of the values of the prime attributes can be merged if the similarity relations of the values of uncertain attributes are greater than or equal to the level value (assigned to those attributes) specified in the fuzzy query. This additional capability of the *Merge* operator allows us to handle fuzzy queries that may operate on uncertain information as well as precise information.

If each $L_j(A_j) = 1$ and the schema redefinition clause, $[B = (A_{k+1}, \dots, A_n)]$, is not null, then the *Merge* operator is equivalent to the *Nest* operator [19,20,21]. If the schema redefinition clause is null and each $L_j(A_j) = 1$, tuples in R are combined but the schema remains unchanged (the second rule above is not applied). In this case the *Merge* operator becomes equivalent to the *Pack* operator [22].

Unmerge Operator: There exists another restructuring operator, called *Unmerge*, that under certain conditions, is an inverse of *Merge*. This condition is that the NF^2 relation be in Partitioned Normal Form (PNF) [21]. A relation is in PNF if and only if (1) all or a subset of the zero-order attributes forms a relation key and (2) every subrelation is in PNF. This operator takes a relation structure nested on a set of attributes and desegregates the structure to make it "flatter". The definition of the *Unmerge* operation is as follows.

Definition: Suppose that we have relation R with the sort; $\text{Sort}(R) = \langle A_1 : \tau_1, \dots, B : \{ \langle A_i : \tau_i, \dots, A_k : \tau_k \rangle \}, \dots, A_n : \tau_n \rangle$, where the schema of R is $\text{Sch } R : (A_1, \dots, B, \dots, A_n)$. Then for instances r of R , we have the following:

$$S = \text{UNMERGE}(R)[B] = \{ \langle A_1 : x_1, \dots, A_i : x_i, \dots, A_k : x_k, \dots, A_n : x_n \rangle \mid \langle A_1 : x_1, \dots, B : y, \dots, A_n : x_n \rangle \in r(R) \wedge \langle A_i : x_i, \dots, A_k : x_k \rangle \in y(B) \wedge B \in \text{Rel}A \wedge B : (A_i, \dots, A_k) \}, \text{ where } 1 \leq i, k \leq n.$$

$$t_m[S] = t_m[\{ \text{Sch}R - B \} \cup (A_i, \dots, A_k)] \mid t_v \in r \wedge t_m \in s \wedge (t_m[A_i, \dots, A_k] \in \{ t_v[B] \}).$$

The *Unmerge* operator produces a relation s with scheme $S = \text{Sch}R - B \cup (A_i, \dots, A_k)$. It takes a relation structure nested on a set of attributes and desegregates the structure to make it flatter.

2.3 XML Databases

XML's great flexibility opens up a range of applications for this standard, far broader than that available to HTML. Not only can XML process complex, hierarchical information, it can also be used for commercial transactions. To get maximum gain from these advantages there are two requirements that must be satisfied by the XML infrastructure:

- XML information must be made available quickly and reliably
- XML information must be integrated with existing corporate data.

Whereas HTML pages can still be managed in a file system, more powerful concepts are needed for complex XML documents. The present state of the art is to use

databases whose functionality, consistency, restart capability, data security and recovery tools can be utilized by the XML data.

The simplest way to accommodate XML objects in a database would be to save them as “character large objects“. The tags in these objects would be interpreted as straightforward running text, which could be handled using the retrieval methods for full text. A database offers further possibilities for indexing the information objects and thus facilitates more flexible access paths. This would permit these objects to be accessed both via their structures (structure-based retrieval) and via their content (content-based retrieval). In this case all relevant information would have to be managed in separate fields or tables. On the other hand, it would not be possible to use this method to map the hierarchical structures of XML documents. Although a solution of this kind could be implemented (relatively) easily with the means currently available, it would certainly not be adequate to cope efficiently with large volumes of data or, above all, XML transaction data.

The most obvious approach for integrating XML into conventional database technologies would be to implement XML structures within data models that are in common use, such as the relational model. To achieve this, XML-formatted objects have to be converted before they fit into relational database structures. Since, with relational data, complex data objects are composed of “flat” tables in the application logic, either additional definition is required in the form of data maps or different data records must be joined together with a JOIN operator integrated in an XQuery statement. On the other hand, implementing every possible tree structure that is

supported by XML in a relational data model is a complex but solvable problem. XML allows information to be graded hierarchically to any required depth. In order to reproduce such hierarchies in a relational database, complex table links would have to be not simply created but also maintained. This kind of model would only be suitable for managing very simple XML documents.

The locking mechanisms of RDBMSs represent a further serious obstacle. Locks at the document level are not supported, since documents are irrelevant to the RDBMS methodology. In order to modify an XML document mapped in an RDBMS, a large number of locks would have to be checked in several different tables – yet another cause of deteriorations in performance which would also entail an enormous amount of internal administration.

At first glance, the world of hierarchically structured XML objects suggests a direct relationship with object-oriented databases (ODBMS). The latter do indeed offer the almost generic option for XML of implementing the structures of persistent objects in a server. Persistent objects could thus be made available with XML at a transparent interface and accessed by a large number of users either locally or via the Web. On the other hand, the familiar drawbacks of ODBMSs cannot be avoided if the data passes via an XML interface. In practice, ODBMSs are inadequate for high throughput or large volumes of data, as well as being particularly unsuitable for transaction tasks of the kind essential for electronic business.

It is also possible to store data in XML documents in a native XML database. There are several reasons to do this. The first of these is when your data is semi-structured. That is, it has a regular structure, but that structure varies enough that mapping it to a relational database results in either a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient). Although semi-structured data can be stored in object-oriented and hierarchical databases, you can also choose to store it in a native XML database in the form of an XML document.

The second reason to store data in a native XML database is retrieval speed. Depending on how the native XML database physically stores data, it might be able to retrieve data much faster than a relational database. The reason for this is that some storage strategies used by native XML databases store entire documents together physically or use physical (rather than logical) pointers between the parts of the document. This allows the documents to be retrieved either without joins or with physical joins, both of which are faster than the logical joins used by relational databases.

For example, consider the sales order document shown in Figure 7. In a relational database, this information would be stored in four tables: a table for sales orders, a table for items in the sale order, a table for customers and a table for the parts of the sale order and retrieving the document would require joins across these tables.

```

<SalesOrder SONumber="12345">
  <Customer CustNumber="543">
    <CustName>ABC Industries</CustName>
    <Street>123 Main St.</Street>
    <City>Chicago</City>
    <State>IL</State>
    <PostCode>60609</PostCode>
  </Customer>
  <OrderDate>981215</OrderDate>
  <Item ItemNumber="1">
    <Part PartNumber="123">
      <Description>
        <p><b>Turkey wrench:</b><br />
          Stainless steel, one-piece construction, lifetime guarantee.</p>
      </Description>
      <Price>9.95</Price>
    </Part>
    <Quantity>10</Quantity>
  </Item>
</SalesOrder>

```

Figure 7: Sales Order Document

In a native XML database, the entire document might be stored in a single place on the disk, so retrieving it or a fragment of it requires a single index lookup and a single read to retrieve the data. A relational database requires four index lookups and at least four reads to retrieve the data.

The drawback in this case is that the increased speed applies only when retrieving data in the order it is stored on disk. If you want to retrieve a different view of the data, such as a list of customers and the sales orders that apply to them, performance will probably be worse than in a relational database. Thus, storing data in a native XML database for performance reasons applies only when one view of the data predominates in your application.

An XML server provides an ideal basis for storing and exchanging XML objects of different types. Database queries and instructions are not formulated as a string of SQL queries, but are sent to the server as URL. This means that a single query can be applied not just to the documents actually on the XML server, but also to other data sources such as remote XML servers, relational databases and multimedia elements installed on special multimedia servers. As far as the user is concerned, the data appears to come from one server only – XML is the glue that holds the different elements together.

One problem with storing data in a native XML database is that most native XML databases can only return the data as XML. If the application needs the data in another format it must parse the XML before it can use the data. This is clearly a disadvantage for local applications that use a native XML database instead of a relational database, as it incurs overhead not found in for example an ODBC application. It is not a problem with distributed applications that use XML as a data transport, since they must incur this overhead regardless of what type of database is used.

2.4 Fuzzy Querying

Most of the traditional tools for formal modeling, reasoning and computing are crisp, deterministic and precise in nature. However, real situations are very often not crisp and deterministic and cannot be described precisely, i.e., real situations are very often uncertain or vague in a number of ways and a complete description of a real

system would require far more detailed data than a human being could recognize and process simultaneously.

Fuzziness can be defined as the vagueness concerning the semantic meaning of events, phenomenon or statements themselves. It is particularly frequent in all areas in which human judgment, evaluation and decisions are important. Fuzzy Set Theory whose objects *fuzzy sets* are sets with boundaries that are not precise and the membership in this fuzzy set is not a matter of true or false, but rather a matter of *degree* was first introduced by Lofti A Zadeh [23].

A querying system is a kind of information retrieval system that may be used to retrieve relevant objects from a database. The database stores a collection of objects, some of which are of interest to the current user. Each object in database contains the index component that can be used to help identify and select the objects that may be relevant to a user. The essential problem in data retrieval processes is to find the subset of objects in the database that is relevant to a given user. Data retrieval operations are defined by particular user's queries, which identify search criteria in terms of features (attributes) of interest used to describe the objects through own index component.

A search criterion with respect to current relational database being tabular representation of information where objects or records (tuples) are represented on rows consisted of Boolean expression involving attribute names and their values, which cover the index component. One characteristic of these queries, being crisp

queries, is that their search criteria involve precisely defined certain attributes (features) presented through their numerical values. There, atomic query is used with respect to each certain attribute, each of which is expressed quite clear by employing ordinary (crisp) set theory. The result of search executed according to this criterion, which is supported by standard structured query language (SQL), is simply a subset with a crisply-defined collection of objects in the database that satisfy all correspondent atomic queries.

Fuzzy querying allows usage of fuzzy features to describe objects such that a database querying system may then select a subset of the database objects (records), which conform to vague or imprecise description of the objects. The selections of a subset of relevant objects, features of, which are approximately similar, have been provided by fuzzification of numerical attributes of the objects in entire database based on fuzzy set theory [23]. A vague predicate represented by a fuzzy set expresses a softer condition. In other words, fuzzy querying allows the user to define goals and constraints, and provide them with relative importance weights on a scale of 1 (least) to 9 (most).

Two kinds of fuzziness can be found in a relational model, one is to associate membership degrees with individual tuples and another is to represent attribute values with possibility distributions. A membership degree associated with a tuple is interpreted to mean the possibility of the tuple being a member of the corresponding relation. A possibility distribution represented an attribute value means that a crisp

value is not known but the range of values that the attribute may take and the possibility of each value being true are known [24].

In order to make uncertain queries on uncertain data, a similarity matrix is used. An example is given in Table 2.

Table 2: Similarity Matrix for Age Attribute

	young	midyoung	midage	old
young	1	0.7	0.3	0
midyoung	0.7	1	0.5	0.2
midage	0.3	0.5	1	0.5
Old	0	0.2	0.5	1

The similarity between two values is a number from zero and one, inclusive. A similarity value of one indicates the two values are identical, and a value of zero indicates that the two values are not similar. A value in between indicates a degree of similarity. For example, "young" is fairly similar to "midyoung", but not very similar to "midage". "young" is not at all similar to "old". Queries are made with a similarity threshold, and matches are returned based on the matrix. To support a crisp query on fuzzy data, or a fuzzy query on crisp data, a membership function mapping crisp ages to membership values within the possible fuzzy categories are used.

2.5 Web Services

Web services are applications whose logic and functions are accessible using standard Internet protocols and data formats such as Extensible Markup Language (XML) over Hypertext Transfer Protocol (HTTP), and SOAP (Simple Object Access Protocol). Like component-based development, Web services represent black-box functionality that can be reused without knowledge about how the service is implemented.

A Web service interface is defined strictly in terms of the messages that the service accepts and generates. Applications using a Web Service can be implemented on any platform in any programming language, as long as they can create and consume messages defined for the service interface. A Web service can also aggregate other services to provide a higher-level set of features.

Most Web services make use of either SOAP or REST (also known as XML over HTTP) to make requests and deliver responses via the Internet. *REST* (or XML over HTTP or XML/HTTP) uses URLs with specific name/value pairs to invoke methods. The URL is the primary method used for message passing. Once the URL is processed, a well-formatted XML document is returned as a response.

SOAP (Simple Object Access Protocol), a more complex method of sharing messages between client and server, was developed to deal with the limitations of REST. SOAP is a lightweight protocol intended for exchanging structured

information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation-specific semantics.

CHAPTER 3

DESIGN AND IMPLEMENTATION OF XML-BASED FUZZY QUERYING SYSTEM

In this chapter, design and implementation details of our study will be explained.

3.1 System Objectives

In this study, main objective is to develop a system, which can model the data in XML, including the extensions introduced by the Extended Non First Normal Form (ExNF²) [1], perform fuzzy queries on XML data and restructure the XML Schemas. The system provides the user a graphical user interface for performing fuzzy queries on XML data stored in a native XML database management system and restructuring the XML Schemas.

3.2 Modeling Complex and Uncertain Data in XML

In this section XML representation of data including the attribute types introduced in ExNF² model, are explained. The attribute types introduced in ExNF² model are

atomic, null-valued, set-valued, relation-valued, fuzzy-valued, and range-valued attributes.

The data is crisp in an atomic attribute, and it resides directly within the field. Atomic attributes are put in a string as the value of the element in XML document. An example is given in Figure 8; title of a book is an atomic attribute.

```
<ProductInfo>
<book>
  <title> Angels </title>
  .....
  .....
</book>
</ProductInfo>
```

Figure 8: Example XML for Atomic Values

For representing empty values in XML, new values are introduced as parsed character data, such as ni (no information), unk (unknown), and dne (does not exist). For instance, image description of the cover of a book can have one of these values. An example is shown in Figure 9.

```

<ProductInfo>
<book>
  <image>
    <image_description> dne </image_description>
  </image>
</book>
<book>
  <image>
    <image_description> ni </image_description>
  </image>
</book>
</ProductInfo>

```

Figure 9: Example XML for Empty Values

A set-valued attribute is one in which an attribute has several values. For the set-valued attributes, putting the values in a string is unnecessary. Each value in the set can be added as an element to the XML. For instance a book may have several authors; then each author is added in an “author” tag inside “authors” tag. An example is shown in Figure 10.

```

<ProductInfo >
<book>
  <authors>
    <author> Thomas H. Cormen </author>
    <author> Charles E. Leiserson </author>
    <author> Ronald L. Rivest </author>
    <author> Clifford Stein </author>
  </ authors >
  .....
</book>
</ProductInfo>

```

Figure 10: Example XML for Set-Valued Attributes

Fuzzy-valued attributes are used to represent imprecise or vague data. For representing fuzzy-valued attributes, linguistic terms exist such as "red" or "green", for representing colour of a book; the fuzzy value is stored as the value of the element. The semantics of the fuzzy data is represented by "FuzzyPredicate" attribute in "colour" element as a fuzzy term. The fuzzy-valued attributes may have various semantics, such as "OR", "XOR", "AND" for relating the fuzzy values. An example is given in Figure 11.

```
<ProductInfo>
<book>
  <image>
    <colors>
      <colour FuzzyPredicate="OR">red</colour>
      <colour FuzzyPredicate="OR">green</colour>
    </colors>
  </image>
  .....
</book>
</ProductInfo>
```

Figure 11: Example XML for Fuzzy-Valued Attributes

As for range-valued attributes, a string format is used to designate range values. We store ranged-valued attributes for price information of a book with "minPrice" and "maxPrice" tags. An example is shown in Figure 12.

```

<ProductInfo >
<book>
  <minPrice>$12.95</minPrice>
  <maxPrice>$80.00</maxPrice>
  .....
</book>
</ProductInfo>

```

Figure 12: Example XML for Range-Valued Attributes

In Relation-valued attributes, contents of the fields are pointers to other tuples. Relation-valued attributes are represented like set-valued attributes. That is, for each value, one complete relation appears as an element in the tuple. For instance, the relation-valued attribute is represented in the “image” element with “image_url”, “colors”, “image_description” elements. An example is shown in Figure 13.

```

<ProductInfo>
<image>
  <image_url> http://images.amazon.com/images </image_url>
  <colors>
    <colour FuzzyPredicate=”AND”>Green</colour>
  </colors>
  < image_description >circles, numbers</ image_description >
</image>
  .....
</book>
</ProductInfo>

```

Figure 13: Example XML for Relation-Valued Attributes

3.3 System Architecture

The major modules associated with the system and their functions are as follows:

- Module 1: Retrieving data through a web service.
- Module 2: Mapping of retrieved data according to the appropriate XML Schema.
- Module 3: Querying and restructuring XML data that involve complex and uncertain information.

The system architecture is depicted in Figure 14.

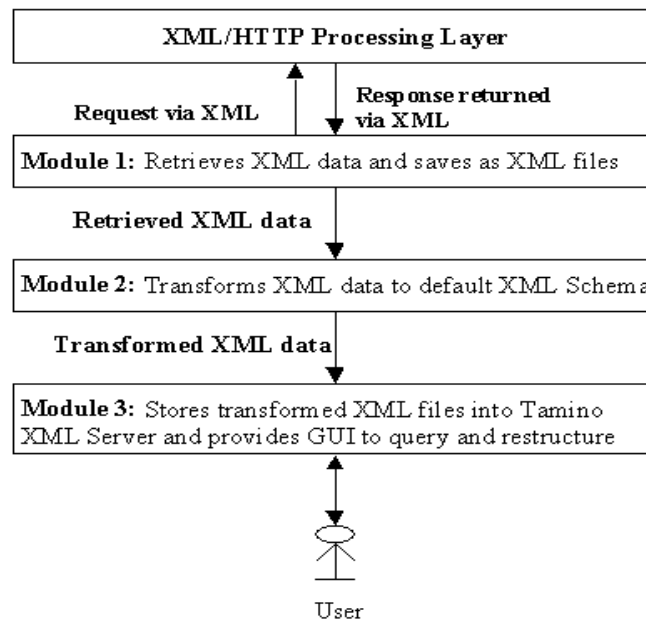


Figure 14: System Architecture

In Module 1, Web Services are used, which is Amazon Web Services in our study, to retrieve the data. We use REST interface in our system. REST (or XML over HTTP

or XML/HTTP) uses URLs with specific name/value pairs to invoke methods and processes within the web services framework. The URL is the primary method used for message passing. Once the URL is processed, a well-formatted XML document is returned as a response. In Module 1, a series of URL's are processed and the well-formatted XML documents returned as response are saved as XML files.

In Module 2, transformation of data retrieved from Amazon's database to our XML schema is implemented. Although the data retrieved is in XML format, we have to map the elements and attributes of the XML document to elements and attributes in our XML Schema because some elements are unnecessary or need reformatting.

In Module 3, the mapped XML documents are loaded into Tamino XML Server. The system provides the user a graphical use interface to query the data stored as XML documents in Tamino native XML database management system. The most important property of this query is its ability to query fuzzy attributes. After performing the query, the user can see the details of the search results. Module 3 also provides the user to restructure the XML Schemas. The user can select the elements of the XML Schema to be merged in a new element tag, and then a new XML Schema is generated and loaded in the Tamino Server. The system also generates new XML documents from the existing XML data and stores them in the database.

3.4 Data Objects & Database Design

In this study, all data are designed to be in XML, stored in a native XML database management system. Other than data, similarity tables for fuzzy attributes are also stored in the database in XML.

In our system books are selected as the data objects. We get the product information from the XML documents retrieved by Web Services. We transform retrieved data to our XML Schema to make it suitable for our application and then insert these documents to our database. An example XML document stored in the database is included in Appendix A. The XML Schema representation of an XML document for storing product information in our system is shown in Figure 15. The code for this XML Schema is included in Appendix B.

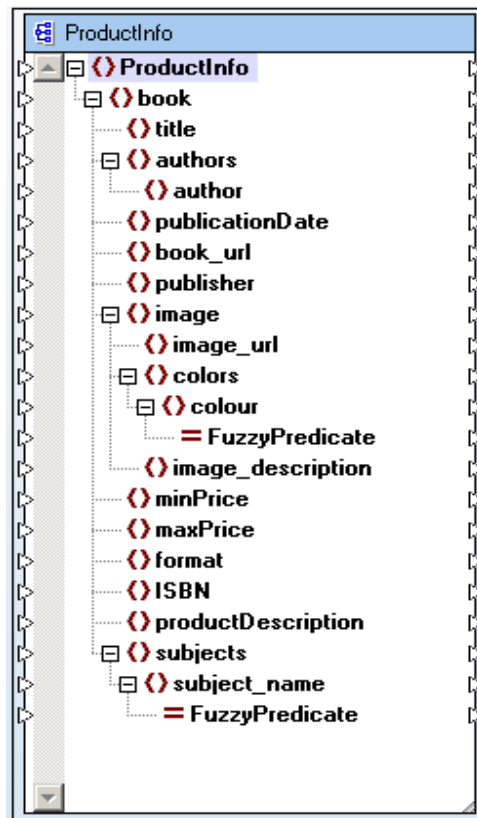


Figure 15: XML Schema Representation of a Product

We use fuzzy “colour” attribute for the cover of a book. The cover of a book may have many colours. Since “colour” is fuzzy attribute, similarity values of the colours are stored in an XML document. This XML document is included in Appendix C. The XML Schema representation of an XML document for storing similarity values of fuzzy “colour” attribute in our system is shown in Figure 16. The code for this XML Schema is included in Appendix D.

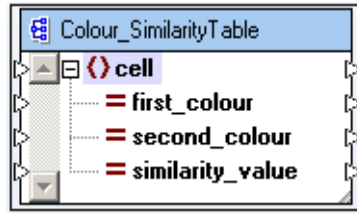


Figure 16: XML Schema Representation of Colour Similarity Table

We also use fuzzy “subject” attribute for books. A book may have several subjects. Since “subject” is fuzzy attribute, similarity values of the subjects are stored in an XML document. This XML document is included in Appendix E. The XML Schema representation of an XML document for storing similarity values of fuzzy “subject” attribute in our system is shown in Figure 17. The code for this XML Schema is included in Appendix F.

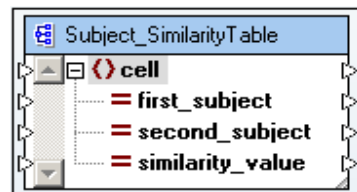


Figure 17: XML Schema Representation of Subject Similarity Table

3.5 Implementation

In this section, the implementation details including user interfaces and modules described previously will be explained in detail.

3.5.1 Retrieving Data through Web Services

In this system, Web Services, which is Amazon Web Services in our study, are used to retrieve the data of the books. We use REST (also known as XML over HTTP) interface to access information in Amazon.com's catalog and databases. REST (representational state transfer) is an approach for getting information content from a Web site by reading a designated Web page that contains an XML file that describes and includes the desired content [16]. We have retrieved data of book with the *Author Search* request. Each request returns an XML file and these files are saved to our directory system. Figure 18 shows call and response formats for the *Author Search*. In the request, "Associates ID" can be obtained from Amazon.com and when a product is sold directed with this ID, you get a commission; "Developer Token" can be obtained by registering to Amazon Web Services. The resulting XML can come in two forms - a "lite" document that contains essential catalog information such as a product's name or price, and a "heavy" document that contains more complete product information such as sales ranking and customer reviews. We have selected "heavy" form in our application.

Request Format	http://xml.amazon.com/onca/xml3?t=[Associates ID]&dev-t=[Developer Token]&AuthorSearch=[author/artist name]&mode=[product line]&type=[lite or heavy]&page=[page #]&f=xml
Response	The AuthorSearch request returns a ProductInfo node. The ProductInfo node contains an array of Detail nodes.

Figure 18: Call and Request Format of Amazon Web Services

3.5.2 Mapping of the Data to the XML Schema

The XML documents including data of the books should be transformed to our “ProductInfo” XML Schema because the data retrieved need to be reformatted accordingly. It may contain unnecessary data or data types should be changed. We reformat data using Altova MAPFORCE; in this tool XML Schemas/documents can be mapped to each other on a graphical user interface and then tool can generate the necessary code for the transformation. After implementing the necessary transformations we have generated the Java code and applied it to our XML documents retrieved by web services. After this transformation XML documents are suitable for inserting them into Tamino XML Server.

3.5.3 Querying the Data

In this system, users can query the books whose data are stored as XML documents in Tamino XML Server with a graphical user interface. The user selects or enters

his/her search criteria on the user interface and an XQuery expression is formed according to this search criteria. This XQuery expression is sent to Tamino XML Server and query results are displayed on the user interface. Tamino API for Java is used for accessing, querying data stored in the database. Figure 19 shows the architectural overview of the query processing in our system.

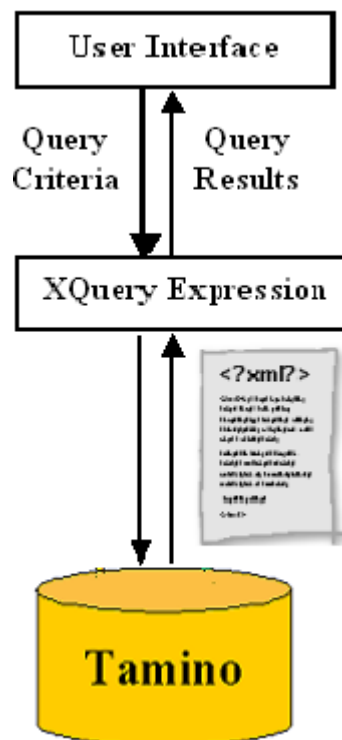


Figure 19: Architectural Overview of the Query Processing

Figure 20 shows the XQuery expression for the query “*Show all books, whose publisher is ‘Thomas Dunne Books’*”. Figure 21 shows the results of this query.

```

for $b in input()/ProductInfo/book
where
  $b/publisher ="Thomas Dunne Books"
return $b"

```

Figure 20: XQuery Example(1)

Fuzzy Book Search

Book

Author:

Title:

Subject:

ISBN:

Publisher:

Publication Date: (e.g. 2002)

Format:

Price:

Cover Colour:

Author	Title	Subject	ISBN	Publisher	Publication Date	Format	Price
Dan Brown	Deception Point	{ History,Classics }	0671027387	Thomas Dunne Books	2002	Paperback	[\$4.00 , \$7 ...
Dan Brown	Digital Fortress : A Thriller	{ History,Science }	0312335164	Thomas Dunne Books	2004	Hardcover	[\$11.95 , \$1 ...
C. J. Date	Foundation for Future Dat...	{ Computers,General }	0201709287	Thomas Dunne Books	2000	Paperback	[\$120.14 , \$...
Sunzi	The Art of War	{ General,History }	0486425576	Thomas Dunne Books	2002	Paperback	[\$3.50 , \$4...
Sun Tzu	Sun Tzu's The Art of War ...	{ History }	1929194196	Thomas Dunne Books	2003	Hardcover	[\$7.41 , \$10...
Sunzi	The Art of Strategy: A Ne...	{ History }	0385237847	Thomas Dunne Books	1988	Paperback	[\$3.35 , \$16...
Sun-Tzu	The Complete Art of War (...)	{ History,Science }	0813330858	Thomas Dunne Books	1996	Hardcover	[\$9.00 , \$35...

7 records

Figure 21: Query Results for Query Example(1)

If the query includes search criteria for fuzzy-valued attributes, the query is performed as follows: At first the values of the attributes that are not fuzzy are

retrieved. Then this result set from the first step is “AND”ed with the result of the fuzzy-valued attributes. For fuzzy-valued attributes we need to use similarity tables for fuzzy values. For instance, if the query is “*Show all books, whose publisher is ‘Thomas Dunne Books and subject is ‘Science’ with threshold value 0,7*”, first the books satisfying publisher criteria are retrieved, and then the books satisfying subject criteria are retrieved using the similarity matrix for subject attribute. If the similarity between “Science” the other subjects are greater or equal to the specified threshold value in query, then those tuples are also included in the resulting answer set. Figure 22 shows an example XQuery expression for the finding the similarity value of “Science” and “History” subjects. The result of this expression is “0,5” as shown in Appendix E. Figure 23 shows the results of the query “*Show all books, whose publisher is ‘Thomas Dunne Books and subject is ‘Science’ with threshold value 0,7*”. As seen in Figure 21, there are 7 records that satisfy search criteria “*books with ‘Thomas Dunne Books’ as publisher*”, but the number of records is 3 in Figure 23. Four books do not satisfy the criteria “*subject science with threshold 0,7*”. Since the similarity between science and computers are 0.9 (which is greater than the specified threshold 0.7), the second tuple in Figure 23 (the author is “C.J. Date”) is among the retrieved tuples.

```

for $b in input()/Subject_SimilarityTable/cell
where
  $b/@first_subject = "Science" and
  $b/@second_subject = "History"
return $b/@similarity_value";

```

Figure 22: XQuery Example(2)

Fuzzy Book Search

Book

Author:

Title:

Subject:

☒ AND ☐ OR

ISBN:

Publisher:

Publication Date: (e.g. 2002)

Format:

Price:

Cover Colour:

☒ AND ☐ OR

Author	Title	Subject	ISBN	Publisher	Publication Date	Format	Price
Dan Brown	Digital Fortress : A Thriller	{ History, Science }	0312335164	Thomas Dunne Books	2004	Hardcover	[\$11.95 , \$...
C. J. Date	Foundation for Future Dat...	{ Computers, General }	0201709287	Thomas Dunne Books	2000	Paperback	[\$120.14 , ...
Sun-Tzu	The Complete Art of War (...)	{ History, Science }	0813330858	Thomas Dunne Books	1996	Hardcover	[\$9.00 , \$3...

3 records

Figure 23: Query Results for Query Example(2)

The details of each field on the “Fuzzy Book Search” form are explained in Table 3.

Table 3: Explanation of Fields on the Query Form

Name	Explanation	Values
Author	The author of the book is entered in this field.	String
Title	The title of the book is entered in this field.	String
Subject (*fuzzy attribute)	The subject of the book and the threshold value for this subject is selected from the combo boxes. If no threshold value is selected, “1” is default value.	Multiple Choice Subject: Art, Business, Classics, Computers, Cooking, History, Medicine, Science, Travel Threshold Value: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Subject Table	Each subject and associated threshold value is listed as a row in this table.	
Add Button for Subject	Each subject and associated threshold value can be added to the subject table with this button.	
“AND”, “OR” radio buttons for subject.	Each entry in the subject table can be searched with the option of logical “AND” or “OR” operator.	
ISBN	The ISBN of the book is entered in this field.	String
Publisher	The publisher of the book is entered in this field.	String
Publication Date	The publication date of the book is entered in this field.	Multiple Choice All dates, Before the year, During the year, After the year
Format	The format of the searched book is entered/selected from this field.	Multiple Choice Hardcover, Paperback, Mass Market Paperback

Table 3: cont.

Name	Explanation	Values
Price	The price range for the searched book can be selected from this field.	Multiple Choice Min Price: 5\$, 10\$, 15\$, 20\$, 25\$, 30\$, 35\$, 40\$, 45\$ Max Price: 10\$, 20\$, 30\$, 40\$, 50\$, 60\$, 80\$, 100\$, 150\$, 200\$
Cover Colour (*fuzzy attribute)	The colour of the cover of the book and the threshold value for this colour is selected from the combo boxes. If no threshold value is selected, "1" is default value.	Multiple Choice Colour: black, blue, brown, green, red, yellow, white Threshold Value: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Cover Colour Table	Each colour and associated threshold value is listed as a row in this table.	
Add Button for Cover Colour	Each colour and associated threshold value can be added to the colour table with this button.	
"AND", "OR" radio buttons for cover colour.	Each entry in the colour table can be searched with the option of logical "AND" or "OR" operator.	
Books Table	Query results are listed in this table. Each row of the table corresponds to a book satisfying the search criteria.	
Search Button	This button is used to perform the query.	
Clean Button	This button is used to clean search criteria.	

After performing the query, user can see the details of the products satisfying the search criteria by double-clicking the selected list item. Figure 24 shows the form designed for showing the details of a book. The details of each field on the “Book” form are explained in Table 4.

The screenshot shows a window titled "Book" with a yellow background. On the left is a book cover for "Introduction to Algorithms, Second Edition" by Thomas H. Cormen. To the right of the cover are labels for various fields, each followed by a text input box or a list box. The fields are: Author (Thomas H. Cormen), Title (Introduction to Algorithms, Second Edition), Subjects (a list box with "General" and "Computers" selected), ISBN (0262032937), Publisher (MIT Press), Publication Date (2001), Format (Hardcover), Price (\$12.95 - \$80.00), and Description (a text area with a scroll bar). At the bottom left, there is a blue link "Click to Buy>>" and a brown "Buy" button. At the bottom center is a brown "OK" button.

Author:	Thomas H. Cormen
Title:	Introduction to Algorithms, Second Edition
Subjects:	<div>Subject</div> <div>General</div> <div>Computers</div>
ISBN:	0262032937
Publisher:	MIT Press
Publication Date:	2001
Format:	Hardcover
Price:	\$12.95 - \$80.00
Description:	<p>Aimed at any serious programmer or computer science student, the new second edition of Introduction to Algorithms builds on the tradition of the original with a truly magisterial guide to the world of algorithms. Clearly presented, mathematical algorithms and not amenable to</p>

[Click to Buy>>](#)

Figure 24: Details Form

Table 4: Explanation of Fields on the Details Form

Name	Explanation
Title	The title of book is displayed in this field.
Subjects	The subject(s) of the book is listed in this table.
ISBN	The ISBN of the book is displayed in this field.
Publisher	The publisher of the book is displayed in this field.
Publication Date	The publication date of the book is displayed in this field.
Format	The format of the book is displayed from in this field.
Price	The price range of the book is displayed in this field.
Description	The description of the book is displayed in this field.
Buy Button	While viewing the details of a book, user can click “Buy” button to go the Amazon’s page to buy the product.
OK Button	This button is used to close the form.

3.5.3.1 Fuzzy Queries

As explained above fuzzy attributes “subject” and “colour” can be queried with threshold values and “AND”, “OR” predicates. Example queries are performed on fuzzy “colour” attribute in the following part. The queries are performed for the book named “Deception Point”. XML representation of the data of this book is given in Figure 25.

```

<ProductInfo
xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<book>
  <title>Deception Point</title>
  <image>
    <colors>
      <colour FuzzyPredicate = "OR">black</colour>
      <colour FuzzyPredicate = "OR">blue</colour>
    </colors>
  </image>
  .....
  .....
</book>
</ProductInfo>

```

Figure 25: XML Representation of Book-“Deception Point”

Figure 26 shows the query for “black”, “blue” colours and “OR” predicate. This query returns the record for “Deception Point” because the book has colours “black” and “blue” with “OR” fuzzy predicate as shown in Figure 25.

Fuzzy Book Search

Book

Author:

Title:

Subject: >>add

☐ AND
☐ OR

ISBN:

Publisher:

Publication Date: (e.g. 2002)

Format:

Price:

Cover Colour: >>add

☐ AND
☐ OR

Author	Title	Subject	ISBN	Publisher	Publication Date	Format	Price
Dan Brown	Deception Point	{ History,Classi...	0671027387	Pocket	2002	Paperback	[\$4.00 , \$7.99]

1 records

Figure 26: “OR” Predicate Query Example

When the query is performed for “black”, “blue” colours and “AND” predicate, no record is returned as the result of the query because the book does not have the “black”, “blue” colours with “AND” predicate. This query is shown in Figure 27.

Fuzzy Book Search

Book

Author: _____

Title: Deception Point

Subject: _____ **>>add**

Subject

☒ AND
☐ OR

ISBN: _____

Publisher: _____

Publication Date: _____ (e.g. 2002)

Format: _____

Price: Min Price >> << Max Price

Cover Colour: blue **>>add**

Colour

black, 1
blue, 1

☒ AND
☐ OR

Author	Title	Subject	ISBN	Publisher	Publication Date	Format	Price

0 records

Search **Clean**

Figure 27: “AND” Predicate Query Example (1)

When we change the threshold values for colours in Figure 27 to “0.3”, the query returns the record for “Deception Point”. Although the book has these colours with “OR” predicate, “black” and “blue” has the similarity value “0.6” as given in Appendix C. Since the threshold value in the query is smaller than the similarity value, the query criteria is satisfied. This query is shown in Figure 28.

Author:

Title:

Subject:

☒ AND
☐ OR

ISBN:

Publisher:

Publication Date: (e.g. 2002)

Format:

Price:

Cover Colour:

☒ AND
☐ OR

Author	Title	Subject	ISBN	Publisher	Publication Date	Format	Price
Dan Brown	Deception Point	{ History,Classi...	0671027387	Pocket	2002	Paperback	[\$4.00 , \$7.99]

1 records

Figure 28: “AND” Predicate Query Example (2)

3.5.4 Merging XML Schemas

In this system, users can merge the elements of the XML Schemas stored in Tamino XML Server. Users can merge the elements of an XML Schema from the graphical user interface and new XML Schema is generated and loaded into the database. In addition to new XML Schema, new XML documents are generated according to new XML Schema and loaded into the database.

In order to restructure an XML Schema user first chooses the XML Schema file to be changed. The chosen XML Schema's elements are shown on the form opened and user can select the elements to be merged by clicking check boxes. User can also disregard primary key, which is "ISBN" in our case, in merge by checking "Ignore Primary Key" check box. By this way, primary key is disregarded when it is not chosen to be merged. The form for choosing XML Schema elements is shown in Figure 29.

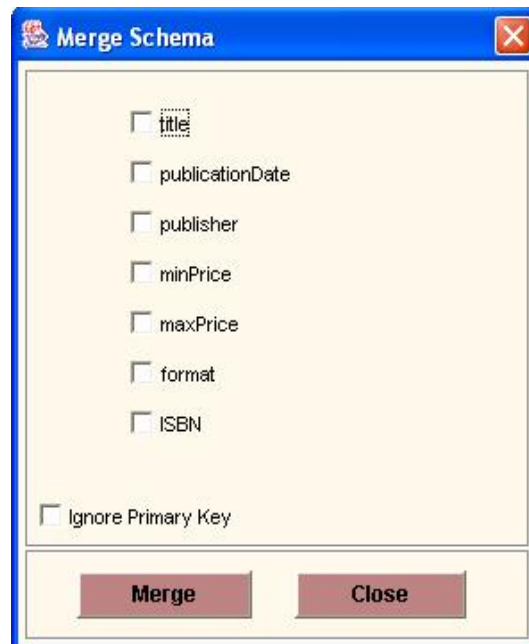
A screenshot of a Windows-style dialog box titled "Merge Schema". The dialog has a blue title bar with a standard close button (X) in the top right corner. The main area has a light yellow background and contains a list of XML elements, each preceded by an unchecked checkbox: "title", "publicationDate", "publisher", "minPrice", "maxPrice", "format", and "ISBN". Below this list is another checkbox labeled "Ignore Primary Key", which is also unchecked. At the bottom of the dialog, there are two buttons: "Merge" and "Close", both with a reddish-brown gradient and black text.

Figure 29: Merge Schema Form

If the user clicks the "Merge" button after selecting the elements to be merged, a dialog is opened asking for the name of the new element, which is formed with the merging of selected elements. Then new XML schema, and XML documents generated according to this new XML Schema are stored into the database.

For example, if the user selects all elements except “publisher” element to be merged with the name “PublisherInfo”, a new XML schema named PublisherInfoSchema is generated. Figure 30 shows the XML Schema structure of this new schema:

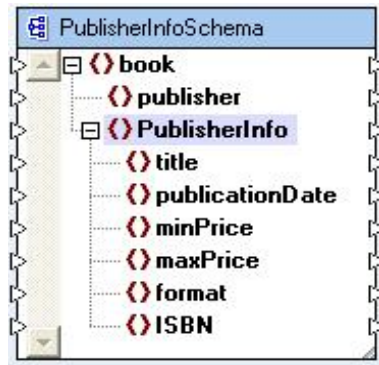


Figure 30: XML Schema Representation of a PublisherInfoSchema

Then the user can view the new XML Schema structure with the XML documents formed according to new XML Schema in a new form. XML documents stored according to original XML Schema and new XML Schema are shown in Figure 31 and Figure 32 respectively.

If the user deselects “ISBN” element and checks “Ignore Primary Key” checkbox in the former example, the views of XML documents are same as in Figure 32. But if the user does not check “Ignore Primary Key” checkbox, primary key is considered during merge, and the view of XML documents are shown in Figure 33.

View Schema						
title	publicationDate	publisher	minPrice	maxPrice	format	ISBN
Angels	2001	Pocket Star	\$3.25	\$7.99	Mass Market Paperback	0671027360
Deception Point	2002	Thomas Dunne Books	\$4.00	\$7.99	Paperback	0671027387
Digital Fortress : A Thriller	2004	Thomas Dunne Books	\$11.95	\$19.95	Hardcover	0312335164
Temporal Data	2002	Morgan Kaufmann	\$49.90	\$49.95	Paperback	1558608559
A Guide to SQL Standard (4th Edition)	1996	Addison-Wesley Pub Co	\$20.79	\$47.95	Paperback	0201964260
Foundation for Object / Relational Databases: The Third Manifesto	1998	Addison-Wesley Professional	\$9.82	\$44.95	Hardcover	0201309785
SQL Standard: A User's Guide to the Standard Relational Database	1993	Addison-Wesley Pub Co	\$4.93	\$36.53	Paperback	020155822X
Foundation for Future Database Systems: The Third Manifesto (2nd Edition)	2000	Thomas Dunne Books	\$120.14	\$39.95	Paperback	0201709287
The Art of War	2002	Thomas Dunne Books	\$3.50	\$4.95	Paperback	0486425576
Sun Tzu's The Art of War Plus The Original Chinese Revealed	2003	Thomas Dunne Books	\$7.41	\$10.95	Hardcover	1929194196
Strategy: A New Translation of Sun Tzu's Classic The Art of War	1988	Thomas Dunne Books	\$3.35	\$16.95	Paperback	0385237847
The Complete Art of War (History and Warfare)	1996	Thomas Dunne Books	\$9.00	\$35.00	Hardcover	0813330858
Wives and Men (Penguin Great Books of the 20th Century)	1993	Addison-Wesley Pub Co	\$1.99	\$8.00	Paperback	0140177396
The Pearl	2000	Addison-Wesley Pub Co	\$2.80	\$8.00	Paperback	014017737X
Of Mice and Men (Penguin Great Books of the 20th Century)	2002	Addison-Wesley Pub Co	\$5.50	\$15.00	Paperback	0142000663
East of Eden (Oprah's Book Club)	2003	Addison-Wesley Pub Co	\$4.95	\$16.00	Paperback	0142004235
Travels With Charley: In Search of America	1980	Addison-Wesley Pub Co	\$1.00	\$9.00	Mass Market Paperback	0140053204

Figure 31: XML Schema Structure Before Merging

View Schema						
publisher	PublisherInfo					
	title	publicationDate	minPrice	maxPrice	format	ISBN
Pocket Star	Angels	2001	\$3.25	\$7.99	Mass Market Paperback	0671027360
Thomas Dunne Books	Deception Point, Digital Fortress	2002, 2004, 2000, 2002	\$4.00, \$11.95, \$120.14	\$7.99, \$19.95, \$39.95	Paperback, Hardcover	0671027387, 0312335164
Morgan Kaufmann	Temporal Data	2002	\$49.90	\$49.95	Paperback	1558608559
Addison-Wesley Pub Co	A Guide to SQL Standard	1996, 1993, 1993, 2000	\$20.79, \$4.93, \$1.99	\$47.95, \$36.53, \$8.00	Paperback, Paperback	0201964260, 020155822X
Addison-Wesley Professional	Foundation for Object / Relational Databases	1998	\$9.82	\$44.95	Hardcover	0201309785

Figure 32: New XML Schema Structure

View Schema						
publisher	ISBN	PublisherISBNInfo				
		title	publicationDate	minPrice	maxPrice	format
Pocket Star	0671027360	Angels	2001	\$3.25	\$7.99	Mass Market Paperback
Thomas Dunne Books	0671027387	Deception Point	2002	\$4.00	\$7.99	Paperback
Thomas Dunne Books	0312335164	Digital Fortress : A Thriller	2004	\$11.95	\$19.95	Hardcover
Morgan Kaufmann	1558608559	Temporal Data	2002	\$49.90	\$49.95	Paperback
Addison-Wesley Pub Co	0201964260	A Guide to SQL Standa...	1996	\$20.79	\$47.95	Paperback
Addison-Wesley Professional	0201309785	Foundation for Object / ...	1998	\$9.82	\$44.95	Hardcover
Addison-Wesley Pub Co	020155822X	A Guide to the SQL Sta...	1993	\$4.93	\$36.53	Paperback
Thomas Dunne Books	0201709287	Foundation for Future D...	2000	\$120.14	\$39.95	Paperback
Thomas Dunne Books	0486425576	The Art of War	2002	\$3.50	\$4.95	Paperback
Thomas Dunne Books	1929194196	Sun Tzu's The Art of W...	2003	\$7.41	\$10.95	Hardcover
Thomas Dunne Books	0385237847	The Art of Strategy: A ...	1988	\$3.35	\$16.95	Paperback
Thomas Dunne Books	0613330858	The Complete Art of Wa...	1996	\$9.00	\$35.00	Hardcover
Addison-Wesley Pub Co	0140177396	Of Mice and Men (Peng...	1993	\$1.99	\$8.00	Paperback
Addison-Wesley Pub Co	014017737X	The Pearl	2000	\$2.80	\$8.00	Paperback
Addison-Wesley Pub Co	0142000663	The Grapes of Wrath: J...	2002	\$5.50	\$15.00	Paperback
Addison-Wesley Pub Co	0142004235	East of Eden (Oprah's ...	2003	\$4.95	\$16.00	Paperback
Addison-Wesley Pub Co	0140053204	Travels With Charley: In...	1980	\$1.00	\$9.00	Mass Market Paperback

Figure 33: XML Schema Structure Regarding Primary Key

3.6 Programming Environment

Java programming language is chosen for the development of system in order to achieve the platform independency. The system is implemented as a standalone application on Borland JBuilder 9 Enterprise Edition. Tamino API for Java is used to access and manipulate the data stored in Tamino database. The jar files included in this API are used in the project libraries. The Tamino implementation of XQuery, called Tamino XQuery 4 is used for performing the queries, which can also be accessed in Tamino API for Java. Software AG's Tamino XML Server 4.1.4 is used as the XML database system. Software AG's Tamino Schema Editor 4.1.4.2 is used for designing the XML schemas. Altova MAPFORCE Enterprise Edition 2004 is used for transforming the XML Schemas.

CHAPTER 4

CONCLUSION

There are many traditional database management systems that store crisp data. However, real-world information containing subjective opinions and judgments contain complex and imprecise data. One of the proposed models for the representation of uncertain and complex data is the Extended Non First Normal Form (ExNF²) data model, an extended relational database model [1].

Sharing of information between databases is also an important issue. Different databases store data in different, and sometimes incompatible, formats, which makes exchanging information a challenge. Many organizations and enterprises establish distributed working environments, where different users need to exchange information based on a common model.

Extensible Mark-up Language (XML) is a proposed solution as a data representation and exchange format through the Internet and different database models to meet this challenge. XML documents simply define the data representation and do not deal with the presentation. XML can also be used to represent complex and imprecise data formats in addition to crisp data formats. XML can process complex,

hierarchical information, and also be used for commercial transactions. Therefore, XML documents can be used to transfer data between the Internet applications and different database models.

In this thesis, complex and uncertain data are represented in XML. In this system, users can perform fuzzy queries on XML documents. XML documents in the system contain fuzzy attributes. The user can select a fuzzy attribute and a threshold value for it and then perform a query. In addition to fuzzy querying, the system enables restructuring of XML Schemas by merging of elements of the XML documents. Using this feature of the system can generate a new XML Schema and new XML documents are generated from the existing documents according to this new XML Schema. XML documents in the system are retrieved from Internet via Web Services. XML documents are stored in a native XML database management system.

REFERENCES

- [1] Yazici, Adnan, Alper Soysal, Bill P. Buckles, and Fred E. Petry. "Uncertainty in nested relational database model." *Data & Knowledge Engineering* 30, pp. 275-301, 1999.
- [2] Widom, Jennifer. "Data Management for XML: Research Directions." Stanford University, <http://www-db.stanford.edu/~widom/xml-whitepaper.html>, 1999.
- [3] Frank Jung, XML Backgrounder, Software AG, http://www1.softwareag.com/Corporate/Images/e-XML_Backgrounder_XML-WP05E0803_tcm16-7780.pdf, August 2003.
- [4] Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1, <http://www.w3.org/XML/1998/06/xmlspec-report-v21.htm>, 1998.
- [5] Charter of the XML Schema Working Group, <http://www.w3.org/2003/09/xmlap/xml-schema-wg-charter.html>, June 2004.
- [6] XML-QL: A Query Language for XML, <http://www.w3.org/TR/1998/NOTE-xml-ql-19980819>, August 1998.
- [7] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation, <http://www.w3.org/TR/xpath>, November 1999.
- [8] XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>, July 2004.
- [9] Extensible Stylesheet Language Transformations, <http://www.w3.org/TR/1999/REC-xslt-19991116>, November 1999.
- [10] XML Pointer Language, <http://www.w3.org/TR/2001/WD-xptr-20010108>, January 2001.
- [11] XML Query (XQuery) Requirements W3C Working Draft, <http://www.w3.org/TR/xquery-requirements/>, November 2003.

- [12] J. Robie, J. Lapp, D. Schach, XML Query Language (XQL), <http://www.w3.org/TandS/QL/QL98/pp/xql.html>, 1998.
- [13] XSL Transformations (XSLT) Version 1.0, <http://www.w3.org/TR/xslt>, November 1999.
- [14] Cascading Style Sheets, <http://www.w3.org/Style/CSS>, June 2001.
- [15] Document Object Model (DOM) Activity Statement, <http://www.w3.org/DOM/Activity>, March 2004.
- [16] REST, http://searchwebservices.techtarget.com/sDefinition/0,sid26_gci823682,00.html, May 2002.
- [17] Elmasri, Ramez and Shamkant B. Navathe. Fundamentals of Database Systems, Second Edition. CA: Addison-Wesley, 1994.
- [18] L.S. Colby, A recursive algebra for nested relations, Information Systems 15 (5) 567-662, 1990.
- [19] M.A. Roth, H.F. Korth, D.S. Batory, SQL/NF: a query language for non-1NF relational databases, Information Systems 12 99-114, 1987.
- [20] H.J. Schek, M.H. Scholl, The relational model with relational-valued attributes, Information Systems 11 (2) 137-147, 1986.
- [21] S.J. Thomas, P.C. Fischer, Nested relational structures, Advances in Computing Research, vol. 3, JAI Press, pp. 269-307, 1986.
- [22] G. Ozsoyoglu, Z.M. Ozsoyoglu, V. Matos, Extending relational algebra and relational calculus with set-valued attributes and aggregate functions, ACM Transactions on Database Systems 12 (4) 566-592, 1987.
- [23] Zadeh, L.A. "Fuzzy Sets" Information and Control, 8(3), pp. 338-353, 1965.
- [24] Z. M. Ma, Weiyin Ma, Wenjun Zhang, An Extended Conceptual Model for Fuzzy Data Modeling, WISE (2) 75-80, 2000.
- [25] G.M. Bryan, J.M. Curry, C. McGregor, D. Holdsworth, R. Sharply, "Modeling Fuzzy Data in XML", <http://csdl.computer.org/comp/proceedings/hicss/2002/1435/04/14350119.pdf>, 2004.

APPENDIX A

AN EXAMPLE XML DOCUMENT FOR A PRODUCT

```
<? xml version="1.0" encoding="UTF-8"?>
<ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C://ProductInfo.TSD"
xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <book>
    <title>Introduction to Algorithms, Second Edition</title>
    <authors>
      <author>Thomas H. Cormen</author>
      <author>Charles E. Leiserson</author>
    </authors>
    <publicationDate>2001</publicationDate>
    <book_url>http://www.amazon.com/exec/obidos </book_url>
    <publisher>MIT Press</publisher>
    <image>
      <image_url>http://images.amazon.com/images/P/0262032937
.01.MZZZZZZZ.jpg</image_url>
      <colors>
        <colour FuzzyPredicate="AND">red</colour>
        <colour FuzzyPredicate="AND">green</colour>
      </colors>
    </image>
    <minPrice>$12.95</minPrice>
    <maxPrice>$80.00</maxPrice>
    <format>Hardcover</format>
    <ISBN>0262032937</ISBN>
    <productDescription>Aimed at any serious programmer or computer
science student, the new second edition of Introduction to Algorithms
builds on the tradition of the original with a truly magisterial guide to the
world of algorithms.</productDescription>
    <subjects>
      <subject_name>Computers</subject_name>
    </subjects>
  </book>
</ProductInfo>
```

APPENDIX B

THE XML SCHEMA CODE FOR PRODUCTS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="ProductInfo">
        <tsd:collection name="deneme"/>
        <tsd:doctype name="ProductInfo">
          <tsd:logical> <tsd:content>closed</tsd:content> </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="ProductInfo"> <xs:complexType>
    <xs:sequence>
      <xs:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence></xs:complexType> </xs:element>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="authors"><xs:complexType><xs:sequence>
        <xs:element ref="author" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence></xs:complexType>
    </xs:element>
    <xs:element name="publicationDate" type="xs:gYear"/>
    <xs:element name="book_url" type="xs:string"/>
    <xs:element name="publisher" type="xs:string"/>
    <xs:element name="image"><xs:complexType><xs:sequence>
      <xs:element ref="image_url" minOccurs="0"/>
      <xs:element ref="colors" maxOccurs="unbounded"/>
      <xs:element ref="image_description" minOccurs="0"/>
    </xs:sequence></xs:complexType></xs:element>
    <xs:element name="minPrice" type="xs:string"/>
    <xs:element name="maxPrice" type="xs:string"/>
    <xs:element name="format" type="xs:string"/>
  </xs:element>
</xs:schema>
```

```

<xs:element name="ISBN" type="xs:string"/>
<xs:element name="productDescription" type="xs:string"/>
<xs:element name="subjects"><xs:complexType><xs:sequence>
  <xs:element ref="subject_name" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence></xs:complexType></xs:element>
  <xs:element name="book"><xs:complexType><xs:sequence>
    <xs:element ref="title" minOccurs="0"/>
    <xs:element ref="authors"/>
    <xs:element ref="publicationDate" minOccurs="0"/>
    <xs:element ref="book_url" minOccurs="0"/>
    <xs:element ref="publisher" minOccurs="0"/>
    <xs:element ref="image" minOccurs="0"/>
    <xs:element ref="minPrice" minOccurs="0"/>
    <xs:element ref="maxPrice" minOccurs="0"/>
    <xs:element ref="format" minOccurs="0"/>
    <xs:element ref="ISBN" minOccurs="0"/>
    <xs:element ref="productDescription" minOccurs="0"/>
    <xs:element ref="subjects" minOccurs="0"/>
  </xs:sequence></xs:complexType>
</xs:element>
  <xs:element name="image_url" type="xs:string"/>
  <xs:element name="colors"><xs:complexType><xs:sequence>
    <xs:element ref="colour" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence></xs:complexType></xs:element>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="image_description" type="xs:string"/>
  <xs:element name="subject_name"><xs:complexType>
    <xs:simpleContent><xs:extension base="xs:string">
      <xs:attribute name="FuzzyPredicate" type="xs:string"/>
    </xs:extension></xs:simpleContent></xs:complexType>
  </xs:element>
  <xs:element name="colour"><xs:complexType><xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="FuzzyPredicate" type="xs:string"/>
    </xs:extension></xs:simpleContent> </xs:complexType>
  </xs:element>
</xs:schema>

```


APPENDIX C

XML DOCUMENT FOR COLOUR SIMILARITY TABLE

```
<?xml version="1.0" encoding="UTF-8"?>
<Colour_SimilarityTable>
  <cell first_colour = "black" second_colour = "black" similarity_value = "1"/>
  <cell first_colour = "black" second_colour = "blue" similarity_value = "0.6"/>
  <cell first_colour = "black" second_colour = "brown" similarity_value = "0.9"/>
  <cell first_colour = "black" second_colour = "green" similarity_value = "0.6"/>
  <cell first_colour = "black" second_colour = "red" similarity_value = "0.4"/>
  <cell first_colour = "black" second_colour = "yellow" similarity_value = "0.1"/>
  <cell first_colour = "black" second_colour = "white" similarity_value = "0"/>
  <cell first_colour = "blue" second_colour = "black" similarity_value = "0.6"/>
  <cell first_colour = "blue" second_colour = "blue" similarity_value = "1"/>
  <cell first_colour = "blue" second_colour = "brown" similarity_value = "0.4"/>
  <cell first_colour = "blue" second_colour = "green" similarity_value = "0.7"/>
  <cell first_colour = "blue" second_colour = "red" similarity_value = "0.4"/>
  <cell first_colour = "blue" second_colour = "yellow" similarity_value = "0.4"/>
  <cell first_colour = "blue" second_colour = "white" similarity_value = "0.2"/>
  <cell first_colour = "brown" second_colour = "black" similarity_value = "0.9"/>
  <cell first_colour = "brown" second_colour = "blue" similarity_value = "0.4"/>
  <cell first_colour = "brown" second_colour = "brown" similarity_value = "1"/>
  <cell first_colour = "brown" second_colour = "green" similarity_value = "0.6"/>
  <cell first_colour = "brown" second_colour = "red" similarity_value = "0.5"/>
  <cell first_colour = "brown" second_colour = "yellow" similarity_value = "0.2"/>
  <cell first_colour = "brown" second_colour = "white" similarity_value = "0"/>
  <cell first_colour = "green" second_colour = "black" similarity_value = "0.6"/>
  <cell first_colour = "green" second_colour = "blue" similarity_value = "0.7"/>
  <cell first_colour = "green" second_colour = "brown" similarity_value = "0.6"/>
  <cell first_colour = "green" second_colour = "green" similarity_value = "1"/>
  <cell first_colour = "green" second_colour = "red" similarity_value = "0.4"/>
  <cell first_colour = "green" second_colour = "yellow" similarity_value = "0.2"/>
  <cell first_colour = "green" second_colour = "white" similarity_value = "0.1"/>
  <cell first_colour = "red" second_colour = "black" similarity_value = "0.4"/>
  <cell first_colour = "red" second_colour = "blue" similarity_value = "0.4"/>
  <cell first_colour = "red" second_colour = "brown" similarity_value = "0.5"/>
  <cell first_colour = "red" second_colour = "green" similarity_value = "0.4"/>
```

```

<cell first_colour = "red" second_colour = "red" similarity_value = "1"/>
<cell first_colour = "red" second_colour = "yellow" similarity_value = "0.7"/>
<cell first_colour = "red" second_colour = "white" similarity_value = "0.3"/>
<cell first_colour = "yellow" second_colour = "black" similarity_value = "0.1"/>
<cell first_colour = "yellow" second_colour = "blue" similarity_value = "0.4"/>
<cell first_colour = "yellow" second_colour = "brown" similarity_value = "0.2"/>
<cell first_colour = "yellow" second_colour = "green" similarity_value = "0.2"/>
<cell first_colour = "yellow" second_colour = "red" similarity_value = "0.7"/>
<cell first_colour = "yellow" second_colour = "yellow" similarity_value = "1"/>
<cell first_colour = "yellow" second_colour = "white" similarity_value = "0.7"/>
<cell first_colour = "white" second_colour = "black" similarity_value = "0"/>
<cell first_colour = "white" second_colour = "blue" similarity_value = "0.2"/>
<cell first_colour = "white" second_colour = "brown" similarity_value = "0"/>
<cell first_colour = "white" second_colour = "green" similarity_value = "0.1"/>
<cell first_colour = "white" second_colour = "red" similarity_value = "0.3"/>
<cell first_colour = "white" second_colour = "yellow" similarity_value = "0.7"/>
<cell first_colour = "white" second_colour = "white" similarity_value = "1"/>
</Colour_SimilarityTable>

```

APPENDIX D

THE XML SCHEMA CODE FOR COLOUR SIMILARITY TABLE

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:tsd =
"http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition" xmlns:xs =
"http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "Colour_SimilarityTable">
        <tsd:collection name = "deneme"></tsd:collection>
        <tsd:doctype name = "Colour_SimilarityTable">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name = "Colour_SimilarityTable">
    <xs:complexType> <xs:sequence>
      <xs:element ref = "cell" maxOccurs = "unbounded"></xs:element>
    </xs:sequence> </xs:complexType>
  </xs:element>
  <xs:element name = "cell">
    <xs:complexType> <xs:simpleContent>
      <xs:extension base = "xs:string">
        <xs:attribute name = "first_colour" type = "xs:string" use =
"required"></xs:attribute>
        <xs:attribute name = "second_colour" type = "xs:string" use =
"required"></xs:attribute>
        <xs:attribute name = "similarity_value" type = "xs:string" use =
"required"></xs:attribute>
      </xs:extension> </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

APPENDIX E

XML DOCUMENT FOR SUBJECT SIMILARITY TABLE

```
<?xml version="1.0" encoding="UTF-8"?>
<Subject_SimilarityTable>
  <cell first_subject = "Art" second_subject = "Art" similarity_value = "1"/>
  <cell first_subject = "Art" second_subject = "Business" similarity_value = "0.2"/>
  <cell first_subject = "Art" second_subject = "Computers" similarity_value = "0.5"/>
  <cell first_subject = "Art" second_subject = "Cooking" similarity_value = "0.6"/>
  <cell first_subject = "Art" second_subject = "History" similarity_value = "0.9"/>
  <cell first_subject = "Art" second_subject = "Medicine" similarity_value = "0.4"/>
  <cell first_subject = "Art" second_subject = "Science" similarity_value = "0.1"/>
  <cell first_subject = "Art" second_subject = "Travel" similarity_value = "0.7"/>
  <cell first_subject = "Art" second_subject = "Classics" similarity_value = "0.8"/>
  <cell first_subject = "Business" second_subject = "Art" similarity_value = "0.2"/>
  <cell first_subject = "Business" second_subject = "Business" similarity_value = "1"/>
  <cell first_subject = "Business" second_subject = "Computers"
similarity_value = "0.7"/>
  <cell first_subject = "Business" second_subject = "Cooking" similarity_value = "0.4"/>
  <cell first_subject = "Business" second_subject = "History" similarity_value = "0.2"/>
  <cell first_subject = "Business" second_subject = "Medicine"
similarity_value = "0.7"/>
  <cell first_subject = "Business" second_subject = "Science" similarity_value = "0.4"/>
  <cell first_subject = "Business" second_subject = "Travel" similarity_value = "0.5"/>
  <cell first_subject = "Business" second_subject = "Classics" similarity_value = "0.2"/>
  <cell first_subject = "Computers" second_subject = "Art" similarity_value = "0.5"/>
  <cell first_subject = "Computers" second_subject = "Business"
similarity_value = "0.7"/>
  <cell first_subject = "Computers" second_subject = "Cooking"
similarity_value = "0.1"/>
  <cell first_subject = "Computers" second_subject = "History"
similarity_value = "0.2"/>
  <cell first_subject = "Computers" second_subject = "Medicine"
similarity_value = "0.8"/>
  <cell first_subject = "Computers" second_subject = "Science"
similarity_value = "0.9"/>
  <cell first_subject = "Computers" second_subject = "Travel" similarity_value = "0.1"/>
```

```

    <cell first_subject="Computers" second_subject="Classics"
similarity_value="0.1"/>
    <cell first_subject="Cooking" second_subject="Art" similarity_value="0.6"/>
    <cell first_subject="Cooking" second_subject="Business" similarity_value="0.4"/>
    <cell first_subject="Cooking" second_subject="Computers"
similarity_value="0.1"/>
    <cell first_subject="Cooking" second_subject="Cooking" similarity_value="1"/>
    <cell first_subject="Cooking" second_subject="History" similarity_value="0.3"/>
    <cell first_subject="Cooking" second_subject="Medicine"
similarity_value="0.5"/>
    <cell first_subject="Cooking" second_subject="Science" similarity_value="0.2"/>
    <cell first_subject="Cooking" second_subject="Travel" similarity_value="0.1"/>
    <cell first_subject="Cooking" second_subject="Classics" similarity_value="0.1"/>
    <cell first_subject="History" second_subject="Art" similarity_value="0.9"/>
    <cell first_subject="History" second_subject="Business" similarity_value="0.2"/>
    <cell first_subject="History" second_subject="Computers"
similarity_value="0.2"/>
    <cell first_subject="History" second_subject="Cooking" similarity_value="0.3"/>
    <cell first_subject="History" second_subject="History" similarity_value="1"/>
    <cell first_subject="History" second_subject="Medicine" similarity_value="0.5"/>
    <cell first_subject="History" second_subject="Travel" similarity_value="0.5"/>
    <cell first_subject="History" second_subject="Classics" similarity_value="0.8"/>
    <cell first_subject="History" second_subject="Science" similarity_value="0.5"/>
    <cell first_subject="Medicine" second_subject="Art" similarity_value="0.4"/>
    <cell first_subject="Medicine" second_subject="Business"
similarity_value="0.7"/>
    <cell first_subject="Medicine" second_subject="Computers"
similarity_value="0.8"/>
    <cell first_subject="Medicine" second_subject="Cooking"
similarity_value="0.5"/>
    <cell first_subject="Medicine" second_subject="History" similarity_value="0.5"/>
    <cell first_subject="Medicine" second_subject="Medicine" similarity_value="1"/>
    <cell first_subject="Medicine" second_subject="Science" similarity_value="0.8"/>
    <cell first_subject="Medicine" second_subject="Travel" similarity_value="0.1"/>
    <cell first_subject="Medicine" second_subject="Classics" similarity_value="0.1"/>
    <cell first_subject="Science" second_subject="Art" similarity_value="0.1"/>
    <cell first_subject="Science" second_subject="Business" similarity_value="0.4"/>
    <cell first_subject="Science" second_subject="Computers"
similarity_value="0.9"/>
    <cell first_subject="Science" second_subject="Cooking" similarity_value="0.2"/>
    <cell first_subject="Science" second_subject="History" similarity_value="0.5"/>
    <cell first_subject="Science" second_subject="Medicine" similarity_value="0.8"/>
    <cell first_subject="Science" second_subject="Science" similarity_value="1"/>
    <cell first_subject="Science" second_subject="Travel" similarity_value="0.1"/>
    <cell first_subject="Science" second_subject="Classics" similarity_value="0.6"/>
    <cell first_subject="Travel" second_subject="Art" similarity_value="0.7"/>
    <cell first_subject="Travel" second_subject="Business" similarity_value="0.5"/>

```

```

<cell first_subject="Travel" second_subject="Computers" similarity_value="0.1"/>
<cell first_subject="Travel" second_subject="Cooking" similarity_value = "0.1"/>
<cell first_subject = "Travel" second_subject ="History" similarity_value= "0.5"/>
<cell first_subject="Travel" second_subject="Medicine" similarity_value="0.1"/>
<cell first_subject="Travel" second_subject= "Science" similarity_value = "0.1"/>
<cell first_subject = "Travel" second_subject = "Travel" similarity_value = "1"/>
<cell first_subject ="Travel" second_subject ="Classics" similarity_value="0.7"/>
<cell first_subject = "Classics" second_subject = "Art" similarity_value = "0.8"/>
<cell first_subject="Classics" second_subject="Business" similarity_value="0.2"/>
<cell first_subject="Classics" second_subject="Computers"
similarity_value="0.1"/>
<cell first_subject="Classics" second_subject="Cooking" similarity_value="0.1"/>
<cell first_subject="Classics" second_subject="History" similarity_value= "0.8"/>
<cell first_subject="Classics" second_subject="Medicine" similarity_value="0.1"/>
<cell first_subject="Classics" second_subject="Science" similarity_value="0.6"/>
<cell first_subject = "Classics" second_subject="Travel" similarity_value="0.7"/>
<cell first_subject = "Classics" second_subject= "Classics" similarity_value="1"/>
<Subject_SimilarityTable>

```

APPENDIX F

THE XML SCHEMA CODE FOR SUBJECT SIMILARITY TABLE

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:tsd = http:// namespaces.softwareag.com/ tamino/
TaminoSchemaDefinition xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "Subject_SimilarityTable">
        <tsd:collection name = "deneme"></tsd:collection>
        <tsd:doctype name = "Subject_SimilarityTable">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name = "Subject_SimilarityTable">
    <xs:complexType> <xs:sequence>
      <xs:element ref = "cell" maxOccurs = "unbounded"></xs:element>
    </xs:sequence></xs:complexType>
  </xs:element>
  <xs:element name = "cell">
    <xs:complexType><xs:simpleContent>
      <xs:extension base = "xs:string">
        <xs:attribute name="first_subject" type="xs:string" use =
"required"></xs:attribute>
        <xs:attribute name = "second_subject" type = "xs:string" use =
"required"></xs:attribute>
        <xs:attribute name = "similarity_value" type = "xs:decimal" use =
"required"></xs:attribute>
      </xs:extension>
    </xs:simpleContent></xs:complexType>
  </xs:element>
</xs:schema>
```