MEU جامعة الشرق الأوسط
MIDDLE EAST UNIVERSITY

XML Database Schema Refinement through Functional Dependencies and Normalization

Master Thesis

By

Zina Zuhair Al Shamaa

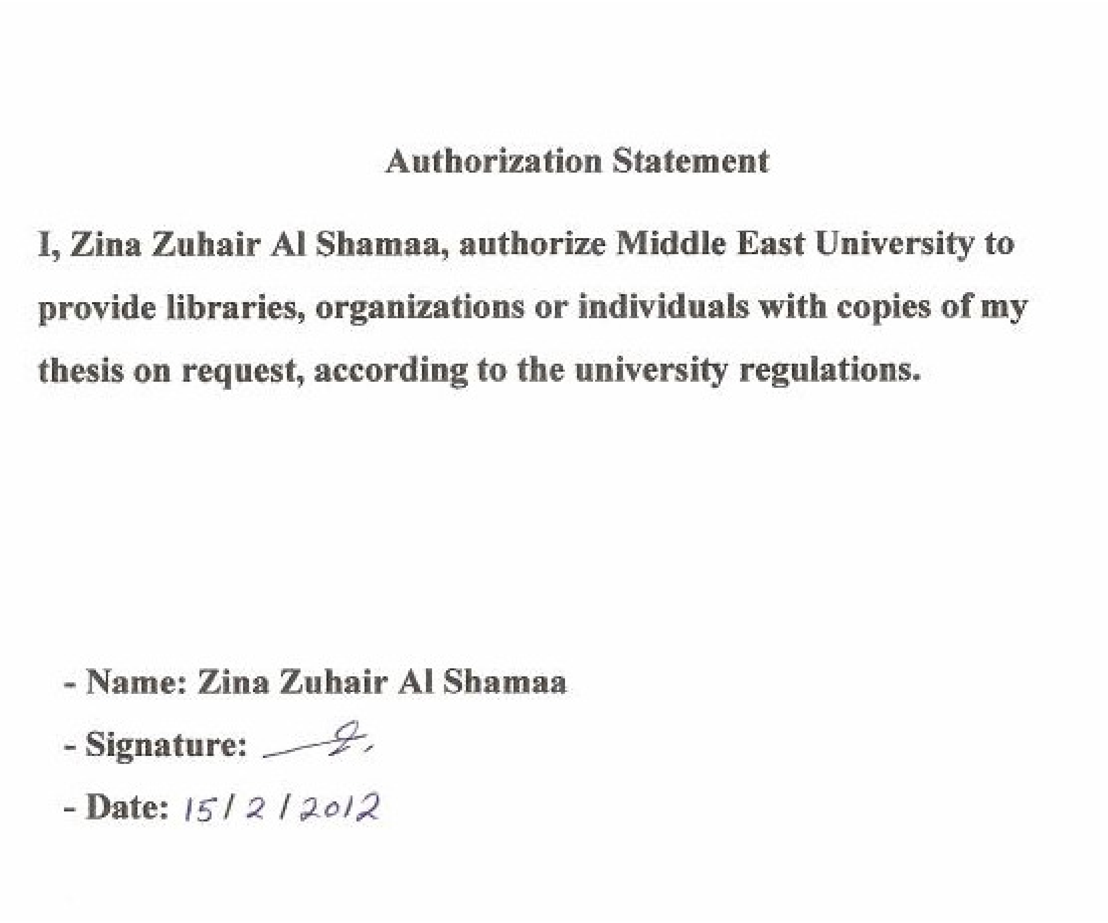
Supervised by

Prof. Musbah M. Aqel

Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Information Systems

Department of Computer Information Systems
Faculty of Information Technology
Middle East University
Amman, Jordan

January, 2012



Authorization Statement

I, Zina Zuhair Al Shamaa, authorize Middle East University to provide libraries, organizations or individuals with copies of my thesis on request, according to the university regulations.

- Name: Zina Zuhair Al Shamaa

- Signature:

- Date: 15/2/2012

Examination Committee Decision

This is to certify that the thesis entitled "XML Database Schema Refinement through Functional Dependencies and Normalization" was successfully defended and approved in February 2012.

Examination Committee Members **Signature**

R. Naoum

Chairman

Prof. Reyadh S. Naoum

Dean of Faculty of information Technology

Middle East University

Musb A

Supervisor

Prof. Musbah M. Aqel

Faculty of Science & Information Technology

Zarqa University

External Examiner

Dr. Ghassan F. Issa

Associate Professor & Dean

Faculty of Information Technology

Petra University

DECLARATION

I do declare hereby that the present research work has been carried out by me under the supervision of Prof. Musbah M. Aqel, and this work has not been submitted elsewhere for any other degree, fellowship or any other similar title.

-Name: Zina Zuhair Al Shamaa

Department of Computer Information Systems.

Faculty of Information Technology.

Middle East University.

Amman, Jordan.

-Signature:

-Date: 15/2/2012

DEDICATION

This thesis is dedicated to my parents, my husband Fakhry and my children, Mahmood, Hassan and Mariam.

Acknowledgments

First and foremost, I thank and pray to God who gave me the health and the strength to do this research.

My deepest gratitude is conveyed to my supervisor Prof. Musbah M. Aqel for his advice, guidance, and constant supervision during the work of this desertation.

Thanks are extended to Dr. Zeyad M. Al Fawaer for his support in the proposal of this research.

I am deeply indebted to thank my parent for their immense patience, emotional support and encouragement in all aspects of my life, and for being my first and greatest teachers.

Furthermore, I would like to thank my dear husband, Fakhry, for his patience and understanding during hard times over the studying period.

My sister and brothers deserve my wholehearted thanks as well.

I could have never finished my degree without the exceptional support of my loving family, friends and supervisors.

Table of Contents

Contents **page#**

List of Figures

List of Appendices

List of Abbreviations

XML	eXtensible Markup Language
XNF	XML Normal Form
DTD	Document Type Definition
XSD	XML Schema Definition
XK	XML Key
XFK	XML Foreign Key
XFD	XML Functional Dependency
LHS	Left Hand Side
RHS	Right Hand Side
PXFD/F	Partial XFD for Flat XML Schema
TXFD/F	Transitive XFD for Flat Schema
AXFD/H	Absolute XML Functional Dependency for Hierarchical Schema
RXFD	Relative XML Functional Dependency
RTXFD	Relative Transitive XML Functional Dependency
RFXFD	Relative Full XML functional Dependency
RPXFD	Relative Partial XML Functional Dependency
EAXFD	Eliminate Absolute XFD
ERTXFD	Eliminate Relative Transitive XFD
ERPXFD	Eliminate Relative Partial XFD

ABSTRACT

XML Database Schema Refinement through Functional Dependencies and Normalization

By

Zina Zuhair Al Shamaa

As eXtensible Markup Language (XML) has become the standard means for storing and exchanging data over the Web, the methods for designing XML database schemas are becoming more and more important. XML documents may contain redundant information due to the bad design of XML schemas. Redundantly stored information can lead to take up unnecessary storage space, inflates data storage and transfer cost; furthermore, it leads to operation anomalies. One approach to remove data redundancies in XML documents is based on normalization theory. This approach proposed XML normal forms (XNFs) to determine whether an XML schema is properly designed or not. Then the XML schema is redesigned or refined to satisfy some XNFs based on the supposed known XML functional dependencies (XFDs). The research about XFD and XML document normalization on the basis of XFD is still an open problem.

In this thesis, we first present a formal definition of the XML Schema Definition (XSD), and then we improve the definition of XFD according to the hierarchical structure of XSD. Since the main goal of identifying XFDs is to detect the possible redundancy they may cause, we defined an XNF that generalizes BCNF. We also define a set of normalization rules for converting any XSD into one in XNF. Finally, we design and implement the process of XML normalization through a semi-automated XML Normalizer tool. The XML Normalizer is very useful for designer to facilitate, accelerate and accurate the process of normalization. We evaluate our approach through examples. The results demonstrate that the XML schema generated by XML normalizer contribute to a normal form schema.

الملخص

تحسين قواعد بيانات (الاكس ام ال) بواسطة دالات التبعية وعملية التبسيط

تعتبر (الاكس ام ال) وسيلة قياسية لتخزين وتبادل البيانات عبر شبكة الويب العالمية. وكاي قاعدة بيانات اخرى فان التصميم السئ لقاعدة البيانات هذه يؤدي الى احتوائها على معلومات فائضة والتي تسبب زيادة مساحة تخزين هذه البيانات وبالتالي تزداد كلفة تخزين ونقل البيانات وكذلك تظهر مشاكل في عمليات تحديث البيانات.

في هذه الدراسة قمنا بتعريف مخطط لقاعدة بيانات الاكس ام ال وهو (الاكس اس دي)، ثم قمنا بتحسين تعريف دالات التبعية (الاكس اف دي) والذي اعتمدنا عليه لتعريف صيغة نموذجية لقواعد بيانات الاكس ام ال الخالية من الفائضية.

كما قمنا بتعريف مجموعة من قوانين التبسيط ثم صممنا المخطط الانسيابي لعملية التبسيط والتي تحول المخطط الاولي لقاعدة بيانات الاكس ام ال الى مخطط بصيغة نموذجية.خالي من الفائضية. واخيرا قمنا بتنفيذ هذه العملية برمجيا واختبرنا النتائج من خلال الامثلة.

Chapter 1

Introduction

With the widespread use of the Web application and the accessibility of a huge amount of electronic data, XML has been used as the standard data model for storing and exchanging data over the Web. Currently, XML is used for many different types of applications which can be classified into two main types. The first application type is called document centric XML and the other is called data centric XML. The document centric XML is used as a markup language for text documents with mixed-content elements and comments. The data centric XML consist of regular structure data for automated processing (Zainol & Wang, 2010).

In data centric applications, a huge amount of data has been managed and stored in XML database which may contain redundant information due to the bad design. The redundantly stored information means the same information stored in more than one place and at different sub trees, which can lead to operation anomalies and waste of storage space that lead to increase the cost of storage and an overmuch costs for transferring and manipulating data . In fact, once a huge XML document are created, its very difficult to change their structure; therefore there is an adventure of having a huge amounts of widely accessible, but poorly structured data (Arenas, 2006).

One strategy to avoid data redundancies is to design schema without redundancies. Thus, a good XML schema design has become an important task. In relational data model, it is clear that the process of designing database is a non trivial and time consuming task, it has two main approaches applied to design a good relational database: The conceptual approach and normalization approach (Connolly

& Begg, 2010). XML database researchers extend these two approaches with some modification to apply in designing a good XML schema.

The first design approach is the conceptual XML data approach (Zainol & Wang, 2010), which is first displayed XML data in terms of a conceptual model, then the model is restructured to eliminate redundancy by using normalization rules, and finally mapping the model into an XML normalized schema.

The other design approach is XML normalization theory (Arenas & Libkin, 2004). It is directly choosing an appropriate schema such as Document Type definition (DTD) or XML Schema Definition (XSD) which describe the constraints on the structure of an XML document. Then a set of data dependencies such as XML functional dependency (XFD) are defined. The data dependencies are used to detect data redundancies in the XML document. Finally, a lossless algorithm is applied to convert an initial schema into one in normal form (Arenas, 2005) which eliminate redundant information and update anomalies.

Just like relational database, the concept of functional dependency (Lee et al, 2002) plays an important role in providing richer data semantic information and normalizing XML data, which has been widely investigated over the past few years. (Chen & Liao, 2010) clarify that a good definition for XFD should have some properties such as: extend the concept of relational model, consider the shape of hierarchical structure, have a powerful to capture a list of nodes as the involved information items, and facilitate the investigation of Normalization for XML.

Although the theory of functional dependencies and normalization in relational database has matured, there is no such mature and systematic theory for XML world and it is still an open problem (Zhao et al, 2009). Some researchers ((Provost, 2002), (Arena, 2006) and (Pankowski & Pilka, 2009)) proposed the idea of applying the theories of relational database on XML database schema design. However, extending functional dependency and normalization theory from relational database can not be applied directly in the XML schema design due to the substantial differences in structure between the two models, relational model are flat and structured while XML schemas are nested, and have hierarchical structure that makes XML functional dependencies items appear at different levels of XML tree (Wu, 2004), (Lv & Yan, 2007).

Research on normalization of XML data was reported in a number of papers ((Arenas & Libkin, 2004) ;(Vincent et al, 2004); (Wu, 2004); (ALibkin, 2007); (LV & Yan, 2007); (Yu & Jagadish, 2008)). Since there is still no standard way in defining XML functional dependencies, a lot of attempts have been made ((Lee et al, 2002); (Vincent et al, 2004); (Yan & Lv, 2006); (Ahmad & Ibrahim, 2008); (Zhao et al., 2009)), the previous definitions of XFD differ in how to choose nodes of sub trees or how to specify equality between nodes. However, they are not powerful enough to specify constrains for every structure of XML. Some literatures researched on XML keys ((Buneman et al, 2001), (Shahriar & Liu, 2008)). Whatever, the above research has not solved the problems of XML functional dependency and XML Normalization perfectly, and obviously, the task of designing XML schema is becoming more complex than designing relational database due to the irregular hierarchical structure of XML schema.

1.1 Problem Statement

As XML has increasingly used by Web application, a huge amount of data has been managed and stored in XML database. Just like any other database model, XML database may contain redundant information due to the bad design of schema.

Redundantly stored information take up unnecessary storage, inflates data transfer cost, and can lead to the problem of update anomalies such as insertion anomaly, rewriting anomaly and deletion anomaly. Furthermore, once massive Web database are created, it is challenging and hard to change their organization; hence, there is a risk of having huge amounts of widely accessible, but poorly organized data. One strategy to avoid data redundancies is to design redundancy-free schema. Starting from an intuitively correct XML schema, then specify a set of functional dependencies which reflect semantic constraint existing in application domain. Then the schema is normalized, and restructured according to some roles to obtain new schema that has no redundancy. However, the hierarchical structure of XML documents makes the normalization process quite challenging. Figure 1.1 illustrates the normalization process steps for XML document. This process takes an XSD as input, and then a set of constraints is defined such as keys and functional dependencies. Finally, Normalization is carried out according to set of rules to convert initial schema into one in normal form.

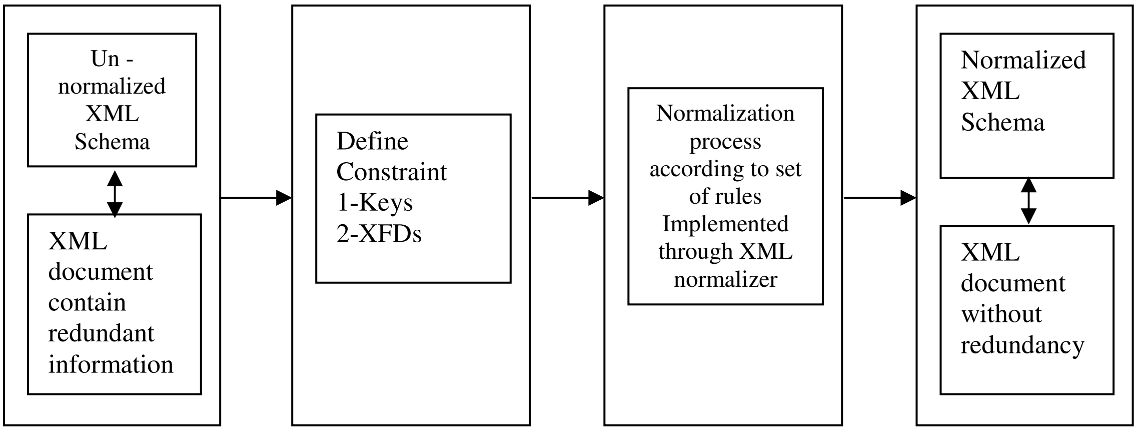

Un - normalized XML Schema
XML document contain redundant information
Define Constraint
1-Keys
2-XFDs
Normalization process according to set of rules Implemented through XML normalizer
Normalized XML Schema
XML document without redundancy

Figure 1.1: XML document normalization process steps.

1.2 Thesis Contribution

The main contribution of our thesis is as follows:

1-As XSD is an improvement over DTD and touted to overcome some shortfalls of DTD. In our research we use XSD and introduce the formal definition of this schema.

2-The notion of functional dependency plays an important role in normalization theory, so we improve the definition of XML functional dependency by using the XPath language and takes into consideration the hierarchical structure of XSD schema by adding the level of last elements of paths to definition.

3-We introduce a set of dependencies depending on our improved functional dependencies definition and integrated the definition of relative dependency for (Zhao et al., 2009) with the definition of (Wu et al., 2002).

4- We define an XML normal form that generalizes BCNF.

5- We designed and implemented a case tool, called XML normalizer that automates the XML schema normalization process.

1.3 Related Work

Normalization theory for XML was proposed by (Provost, 2002) to perform in similar manner to relational normalization. Even though there were many differences between relational schema and XML schema, similar techniques were used. Arena & Libkin, (2004) proposed a formal definition of XML functional dependency which is considered as basis of other related research such as normalizing XML document and schema design, then proposed the most accepted XML normal form based on XFD.

There are two major approaches of XML functional dependency definitions. The first approach is based on tree tuple (Arenas & Libkin, 2004) and the second approach is based on path (Vincent et al, 2004; Yan & Lv, 2006)). Yu & Jagadish, (2007) showed that the previous definitions of XFD is not sufficient and propose a Generalized Tree Tuple XML FD. Many researches defined the XFD without considering the scope of XFD. The scope is an important characteristic in XML documents according to their nested structure, some researchers classified XFDs into two categories according to the scope of XFD: local and global (Yu & Jagadish, 2007; Ahmad & Ibrahim, 2009), while (Zhao et al., 2009) proposed a new kind of XFD that can be classified into: absolute and relative XFD which has stronger expression ability to XML documents.

Several XML normal forms were proposed by ((Wu et al., 2002); (Arenas & Libkin , 2004); (Lv & Yan, 2007); (Yu & Jagadish, 2008); (Pankowski & Pilka, 2009); (Zhao et al., 2009); (Zainol & Wang, 2010)) depending on definition of XFDs ((Lee et al, 2002); (Vincent et al, 2004); (Lv & Yan, 2006); (Zhao et al., 2009)) and keys ((Buneman et al., 2001); (Shahriar & Liu, 2008)) which are studied in the context of XML. They differ in Terms of schema and how to describe constraint, but are dependent on the same set of transformations.

In the following section we introduce some of the most important related studies in the field of XML functional dependencies and XML normal form which provide us a good guidance to our work.

(Wu et al., 2002) They presented the notion of a semi-structured data model which is richer, more complex than the flat relational data model, and plays an important role in the prevalent Web applications. They incorporated the definition of semi-structured schema with integrity constraint (dependency and key constraints). They clarified that just like in relational database, semi structured schema may contain data redundancy and inconsistency if it is not designed well which causes the occurrences of different anomalies such as insertion anomaly, deletion anomaly, rewriting anomaly, and path anomaly. They proposed a Normal Form for Semi Structured Schema (NF-SS) which guarantee minimal redundancy, no undesirable updating anomalies and a more reasonable representation of real world semantics. They introduced restructuring rules. Finally they developed an algorithm used to restructuring a semi structured schema into a normal form based on defined set of rules.

(Arenas & Libkin, 2004) took the first step towards a good XML schema design and normalization theory. Firstly, they introduced an XFD by considering a relational representation of documents based on the approach of tree tuple, and defined XFD for a DTD as an expression of the form $S_1 \to S_2$ where S_1,S_2 are finite non-empty subsets of paths(X). Secondly, they defined an XML normal form that avoids redundancy caused by XFDs and disallows update anomalies. The definition of normal form they defined generalizes BCNF in relational database. There definition is as follows:

Given a DTD schema and a set of functional dependencies, then the DTD schema is in normal form if and only if for every non-trivial functional dependency of the form $S \to p.@l$ or $S \to p.S$, it is the case that $S \to p$ must be implied by the schema. Where the LHS is path end with element and the RHS is path end with string or attribute. The intuition is for every set of values of the element in *S*, there is exactly one value of the path $p.@l$. Finally, they introduced a decomposition algorithm for two kinds of commonly design problem that combines two basic ideas: creating a new element type, and moving an attributes. This algorithm is converting an arbitrary DTD schema into one in normal form depending on a given set of XFD.

(Lv & Yan, 2007) defined XFD based on the tree model and introduced the definition of keys and three types of functional dependencies such as full Functional dependency, partial functional dependency and transitive functional dependency. Then according to these definitions they proposed three XML normal forms such as first XML normal form, second XML normal form and third XML normal form. They supposed that their normal form can eliminate data redundancies and operation anomalies.

(Yu & Jagadish, 2008) showed that the XFD was defined by (Arenas & Libkin, 2004) and the one defined by (Vincent et al., 2004) are insufficient for capturing certain XML data redundancies, therefore they proposed a new XFD based on generalized tree tuple that extends and improves the notion introduced by (Arenas & Libkin, 2004) and showed that there XFD can capture more data redundancies. Yu & Jagadish defined XFD as a triple $< C_p, LHS, RHS >$, where C_p denotes a tuple class, LHS is a set of paths relative to p, and RHS is a single path relative to p. They classified functional dependency into two categories: local and global functional dependency.

They present the design and implement of the first detection system, Discover XFD, for efficiently discover XFDs and showed that discovered XFD can capture more data redundancies. Moreover, they introduced a new normal form for schema based on new XFD, called generalized tree tuple normal form which satisfied if and only if for each XFD of the form (C_p, *LHS, RHS*) the (C_p, *LHS*) is an XML key. Finally, they introduced a normalization algorithm for reconstruction the initial XML schema into a new normal form by eliminating the redundancy in global functional dependency, and eliminating the redundancy of local functional dependency.

(Pankowski & Pilka, 2009) presented a language, which is a class of XPath expressions, to express XML functional dependencies. They defined XML normal form based on the approach of (Arena, 2006). They showed how to develop a method for normalizing XML data by firstly building a conceptual model using ER schema and specifying all dependencies for its attributes, then following some condition to create XML schema in normal form according to its functional dependencies. Finally, they showed that generated schema can be further normalized by using the decomposition algorithm proposed by (Arenas & Libkin, 2004).

(Zhao et al., 2009) proposed a novel expression of functional dependency depending on path language of XML model, which is used to better express the XML data constraints that can result in redundancies. Their definition is of the form $(O,(P,(Q_1,Q_2,.....Q_n \rightarrow Q_{n+1}(S_1,.....S_m))))$ where O is context path, P is target path, $Q_1,Q_2,....,Q_n$ is called left path, Q_{n+1} is called right target path, and $S_1,...,S_m$ are called right paths. Furthermore, they classify the functional dependencies into absolute functional dependency and relative functional dependency. Then depending on the definition of XFD, they proposed a kind of XML normal form and a lossless conversion algorithm of DTD that convert abnormal XML document into normal form.

(Zainol & Wang, 2010) They introduced a method to improve XML structural design by transforming the DTD into a proposed conceptual model called Graphical Notation for Document Type Definition (GN-DTD). The GN-DTD is a graphical model approach that present the DTD schema and XML documents. They defined data dependencies between object of schema which is categorized into key dependencies and functional dependencies (Global functional dependency, transitive functional dependency, and partial functional dependency). Furthermore, they presented normalization rule to switch the model into proposed first normal form, second normal form, third normal form, and normal form for GN-DTD; based on the defined dependencies. Finally they present mapping rules to transform from normalized GN-DTD back to a new DTD schema.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2: We present the definition of basic knowledge and terminology for XML structure upon which the thesis rests. We introduce the formal model for XML documents and XSDs as well as the notion of paths in XML documents and in XSDs. We also introduce the notion of tree tuple and the concept of keys over XML schema. Finally we introduce the types of XML schema representation.

Chapter 3: we present the definition of Functional Dependencies for XML (XFDs), and then we introduce our improvement definition of XFD**.** We also introduce the types of dependencies according to the types of representation. Finally, we introduce the notion of XML normal form (XNF).

Chapter 4: we present normalization rules that we used to transform the un-normal form of XSD into a normal one. We also, present the flowchart of normalization process and the case tool we developed to automate the process of XML database normalization. Finally, we present examples to illustrate how the XSD is restructured to XNF.

Chapter 5: Conclusion and future work.

Chapter 2

XML Databases

This chapter presents the basic knowledge and terminology upon which the thesis rests. The content presented here will be the frame of reference for the remaining chapters.

2.1 Introduction

Extensible Markup Language was mainly appeared to disseminate electronic data, but recently has become standard format for storing and exchanging data over the Web (Arenas, 2006). The data in XML document are represented in hierarchical model and XML schema describes the structure of those data. The easiest way to create an XML Schema is to follow the structure of the document and define each element in the document (Connolly & Begg, 2010).

An example of XML document is shown in Figure 2.1. The document contains two different types of tags: *start tag*, such as `<Dname>` and *end tag*, such as `</Dname>`. XML element tags are case sensitive. These tags must be balanced and they are used to delimit elements. For example, `<cname>Database</cname>` is an element bounded by matching tags `<cname>` and `</cname>`. Every element can contain raw text, other elements, or a mixture of them. For instance, the element we mentioned above contains raw text while the element delimited by `<school >` contains three elements. The first element `<school >` must be a root element.

The XML document shown in Figure 2.1 is part of a database for storing information about school management activity. The school has many departments. Every department includes department name (Dname), a set of courses, and offices.

Every course has course number (cno), course name (cname), and list of students which chose these courses. The student list contains a set of student's number (sno), student name (sname), student age (age), and grade (grade). Each office has room number (room-no), office name (Oname) and address (address) of the building it located.

```
<school xmlns="http://tempuri.org/XSDSchema1.xsd">
 <department >
  <Dname>CIS</Dname>
  <course>
    <cno>10</cno>
    <cname>Database</cname>
    <student>
      <sno>1</sno>
      <sname>Ahmad</sname>
      <age>35</age>
      <grade>B</grade>
    </student>
  </course>
  <office>
      <room-no>100</room-no>
      <Oname>Secrtary</Oname>
      <address>Buld-1</address>
  </office>
 </department>
</school>
```

Figure 2.1: An XML document for school management database.

XML documents have a nested structure. This gives a lot of flexibility when storing information. To specify the structure of a class of XML documents, we have to specify a schema. Schema languages for XML have been heavily researched with the DTD (Document Type Definition) (Shipman, 2009) and XSD (XML Schema Definition) (Thompson et al., 2004) being the most popular currently. DTD has been the de-facto schema language for XML for the past couple of years and has widely used in many theoretical researches. It defines the key and foreign key in the form of ID and IDREF, however it is not clear that ID and IDREF attributes are used as database key rather than internal pointers. The XSD is an effort to overcome some shortfalls of the DTD, In general, it is richer than DTD and has additional property such as specifying type constraint and complex cardinality constraints that make XSD schema more powerful and expressive than DTD (Lee et al., 2002). Obviously, from a theoretical point of view, DTDs can be characterized in terms of unranked tree automata, which have been widely studied in automata theory and more recently in database theory (Arenas, 2005), but XSDs have increasingly used by Web applications to manage their data since they have many characteristics over DTD that are desired in applications. In this research, we consider only XSD.

The XML schema is a node labeled tree. It is defined as a method to express the content model of an XML document structure. It consists of a finite set of elements with two distinguished types (complex type, simple type). The element that contains other elements is defined of complex type while the elements that have no sub elements or attributes are defined of simple type. Each element has some relationship with other elements. These relationships define the structure of the schema which can ensure that the data is well organized and can be maintained and exchanged by applications robustly. To express the structure, some rules and constraints have to define for the data. The rules that state the hierarchy of element in an XML schema are defined in XSD (Connolly & Begg, 2010). Example 2.1.1 shows an element of complex type and Example 2.1.2 shows an element of simple type.

Example 2.1.1 (Connolly & Begg, 2010)

```
 <xs: element name "       ">
         <xs: complex type>
         <xs: sequence>
            <!--- children defined here..>
         </xs: sequence>
         </xs:complex>
</xs:element>
```

Example 2.1.2 (Connolly & Begg, 2010)

```
 <xs:element name="STAFNO" type="xs:string"/>
```

An XSD schema for the school management database is shown in Figure 2.2. This schema specifies the structure of elements allowed in XML document. The first element tag must be a root (school) which declared as complex element followed by an arbitrary number of complex elements (department) which declared by the maxoccurs constraint as unbounded. Each department contains simple element

(Dname), complex element (course) with unbounded constraint, and complex element (office) also unbounded constraint. Each complex element course contains two simple elements (Cno, Cname) and one complex element (student) with unbounded constraint while each student contains four simple elements (Sno, Sname, age, grade). The second complex element in department is (office) contains three simple elements (room-no, Oname, address).

```
<xs:element name="school">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="department" minOccurs="0" maxOccurs="unbounded" >
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Dname" type="xs:string" />
            <xs:element name="course" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="cno" type="xs:string" />
                  <xs:element name="cname" type="xs:string" />
                  <xs:element name="student" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="sno" type="xs:string" />
                        <xs:element name="sname" type="xs:string" />
                        <xs:element name="age" type="xs:string" />
                        <xs:element name="grade" type="xs:string" />
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="office" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="room-no" type="xs:string" />
                  <xs:element name="Oname" type="xs:string" />
                  <xs:element name="address" type="xs:string" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        <xs:key name="departmentKey1">
          <xs:selector xpath="." />
          <xs:field xpath="mstns:Dname" />
        </xs:key>
        <xs:key name="departmentKey2">
          <xs:selector xpath=".//mstns:course" />
          <xs:field xpath="mstns:cno" />
        </xs:key>
        <xs:key name="departmentKey3">
          <xs:selector xpath=".//mstns:student" />
          <xs:field xpath="mstns:sno" />
        </xs:key>
        <xs:key name="departmentKey4">
          <xs:selector xpath=".//mstns:office" />
          <xs:field xpath="mstns:room-no" />
        </xs:key>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 2.2: An XSD schema for school management database.

In the next section, we formalize the notion of XML document and XSD.

2.2 XML Documents and XSDs

In this section, we present the formal model for XML documents and XSDs. Also in this section, we introduce the notion of paths in XML documents and in XSDs.

Assume that we have the following disjoint sets:

El is any set of complex element names and simple element names,

A is any set of attribute names (to refer to attribute, all attribute names start with the symbol @ to distinguish them from labels),

L is any set of labels,

S String values of attributes

N is any set of Nodes, represented as $\{n_0, n_1, ..., n_i\}$

In Chen and Liao's model (Chen & Liao, 2010), XML documents are represented as trees.

Definition 2.2.1 (XML Tree)

An XML tree T is defined to be $T = (N, root, label, comp, val)$, where

- *N* is a finite set of nodes in the tree that represent $CE \cup SE \cup A$
- *root* is the first complex node in tree
- *Label* is a function that assigns a label to each node in tree, such that for each node $n \in N$, if $lab(n) \in CE$ then *n* is called complex element, if $lab(n) \in SE$ then *n* is called simple element, and if $lab(n) \in A$ then *n* is called attribute.
- *comp* is a function from complex element node to a sequence node of $CE \cup SE \cup A$ such as when $n \in N$,if $u \in comp(n)$ then we call *u* a child of *n* and *n* the parent of *u*, so the parent-child relationships represent the structure of tree.

- *val* is a function that assigns a value to each *SE* or *A*, the *CE* is null.

Figure 2.3 is an example of a tree representation of the XML document of the school management database in Figure 2.2.

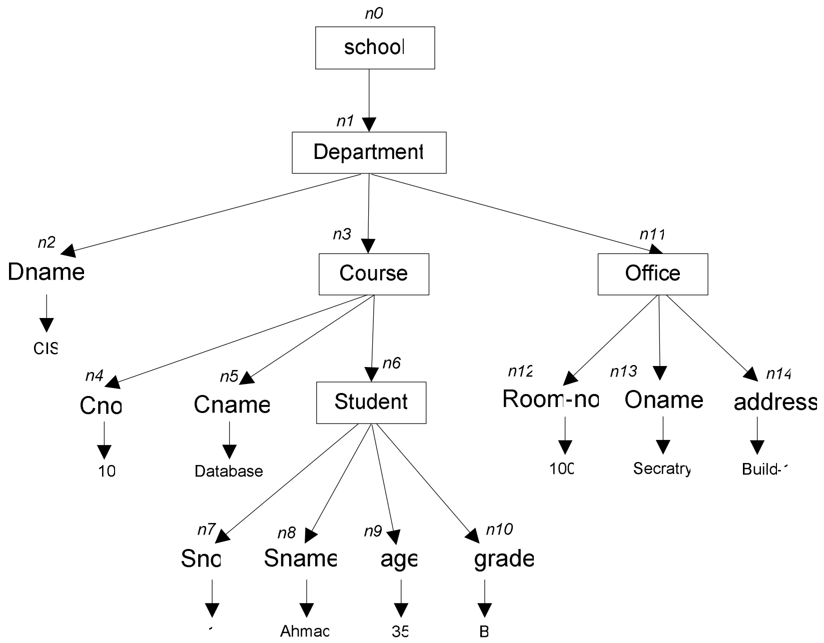


Figure 2.3: Tree representation of an XML document.

Example 2.2.1: Figure 2.3 shows the tree representation of the XML document shown in Figure 2.1. This tree contains a set of nodes, which are labeled as follows:

label(n0)= school *label(n1)*=Department *label(n2)*=Dname

label(n3)=Course *label(n4)*=Cno *label(n5)*=Cname

label(n6)=Student *label*(n7)=Sno *label(n8)*=Sname

label(n9)=age *label(n10)*=grade *label(n11)*=Office

label(n12)=room-no *label(n13)*=Oname *label(n14)*=address

comp(school)=[department] *comp*(department)=[Dname, course, office]

comp(course)=[Cno, Cname, student] *comp*(student)=[Sno, Sname, age, grade]

comp(office)=[room-no, Oname, address] *val*(Dname)=CIS *val*(Cno)=10 *val*(Cname)=Database

val(Sno)=1 *val*(Sname)=Ahmad *val*(age)=35

val(grade)=B *val*(room-no)=100 *val*(Oname)=secrtary

val(address)= Build-1

Definition 2.2.2 of XSD

An XSD X is defined to be $X = (CE, SE, A, F, root, C)$, where:

- CE is a set of complex elements which has another complex element, simple elements and attribute. The children of complex element can be described by three types of model groups (all, choice and sequence). For simplicity we consider on sequence model group whose defines the appearance of sub-elements items in specified order.

SE is a set of simple element nodes that have no sub-elements or attributes, they associated with a data types such as string, date, and decimal.

- A is a set of attribute names that used to identify the properties of a complex element node.
- F is a function from each CE to the children of the element set and attribute ($CE \rightarrow El \cup A \cup S$), and from each SE to its element type definitions.
- $root$ is the root element of the schema, it is of complex type and it is in level zero.
- C is any set of identity constraint such as Keys, and key references.

Example 2.2.2: The XSD shown in Figure 2.2 is represented as follows:
root= school,
CE= {school, department, course, student, office}
SE={Dname, Cno, Cname, Sno, Sname, age, grade, room-no, Oname, address}
F(CE)
F(school)=department* F(department)= Dname,course*, office*

F(course)=Cno, Cname, student* F(student)= Sno, Sname, age, grade
F(office)= room-no, Oname, address
F(SE)
F(Dname)= S F(Cno)= S F(Cname)= S F(Sno)=S
F(Sname)= S F(age)= S F(grade)= S F(room-no)= S
F(Oname)= S F(address)= S
C = Dname, Cno, Sno, room-no (are keys)

2.2.3 Paths in XSDs and Instance in XML Tree Documents

An important concept in XSD is the path expression which is used for navigating and specifying the sequence of elements in XML document. This sequence represents a route that starts from the element type of the root. Parses X according to the rules defined in the XSD, and ends at any specific location in X. The formal definition of paths in XSDs is given bellow (Vincent et al., 2004).

Definition 2.2.3.1 Path in XSD (Vincent et al., 2004)

Given the XSD $X = (CE, SE, A, F, root, C)$ then the path is defined as a sequence of elements $P(x) = e_1 /...../ e_k$ where:

- $e_i \in CE \cup SE \cup A \cup \{S\}$
- $1 \leq i \leq k$
- e_1 = *root*, *Length (P)* = *k* and *last(P)*= e_k
- e_i is in the alphabet of $F(e_{i-1})$ for $2 \leq i \leq k-1$, which mean the element whose name is e_i, is sub- element of the element whose name is e_{i-1}
- e_k is in the alphabet of $F(e_{k-1})$, or e_k =*S*, or $e_k = @a$, which mean the element whose name is e_k is sub-element of the element whose name is e_{k-1}

Definition 2.2.3.2: Path instance in an XML Tree document (Vincent et al., 2004)

A path instance in an XML tree $T = (N, root, lab, comp, val)$ is defined as a sequence of nodes $P(T) = n_1 / ...... / n_k$, where:

$n_1 = root$

n_i is in the alphabet of $comp(n_{i-1})$

n_k is in the alphabet of $comp(n_{k-1})$, or $n_k = SE$, or $n_k = A$

Generally there may be many instances that refer to as target set of paths. For example the set of paths in the XML document shown in Figure 2.1 and Figure 2.3 are:

School,
school/department,
school/department/Dname,
school/department/Dname/S,
school/department/course
school/department/course/Cno
school/department/course/Cno/S,
school/department/course/Cname
school/department/course/Cname/S,
school/department/course/student
school/department/course/student/Sno
school/department/course/student/Sno/S
school/department/course/student/Sname
school/department/course/student/Sname/S
school/department/course/student/Age
school/department/course/student/Age/S
school/department/course/student/Grade
school/department/course/student/Grade/S
school/department/office
school/department/office/room-no
school/department/office/room-no/S
school/department/office/Oname

school/department/office/Oname/S
school/department/office/address
school/department/office/address/S

We can define the following functions for both XSD path and Tree path:

Paths(X) = {*P*|*P* is a path in X},

$Paths(T)$ = {n|n is a path in T}

$Epaths(X) = \{P \in paths(X) \mid last(P) \in CE \bigcup SE\}$,

$Epaths(T) = \{n \in Paths(T) \mid last(n) \in label(n)\}$

$Xpaths(X) = Paths(X) - Epaths(X)$,
$Xpaths(T) = Paths(T) - Epaths(T)$

Let $p = p_1 / .... / p_n$ and $q = q_1 / ..... / q_m$ two paths in X schema then we can have the following functions:

child(p_{i-1} *)=* p_i

$Parent(p_n) = p_1 / ..... / p_{n-1}$

(p)Prefix (q) if $p_1 = q_1$ and $n \leq m$

(p)Equal (q) if $p_1 = q_1$ *and* $p_n = q_m$ *also its true if (p) Prefix (q) and (q) Prefix (p)*

The previous functions are applied to a Tree, if we assumed, we have two instance paths of nodes. (Vincent et al., 2004)

To express about the path logically the XPath language is used. The XPath is a standardized path description language, which is familiar to users and has powerful to express rich cases. It treats an XML document as logical tree with nodes to represent elements, attributes, text, namespace, and root. The basis for XPath technique is the context node and location path, which describe a starting point and path direction respectively. So, XPath is used for navigating and specifying sets of nodes and sets of paths in an XML document tree (Connolly & Begg, 2010). It is used to move in all directions in the document tree such as down to children and descendents, or upwards to parents and ancestors, or may be sideways to siblings (Buneman et al., 2001). The path that begins with a slash (/) is an absolute path, it starting from the top of the document, while the path that start with double slash (//) means that the node start from anywhere in the document. Thus, it is reasonable to use XPath as path description language.

2.3 Tuples and Tree Tuples for XML

In relational database, tuples are used to assign to each attribute a value from the corresponding domain. In order to extend the concept of relational database to the XML data model, the XML data tree is considered as tree tuple (Arenas & Libkin, 2004). The tree tuples are used to assign to each path in X schema a value of stored data. It is defined as a finite XML tree constructed with at most one occurrence of each path in a schema X. The following definition regarding tree tuples are adopted from (Zhang, 2004).

Definition 2.3.1: Tuples for XML (Zhang, 2004)

Given a schema $X = (CE, SE, A, F, root, C)$, a tuple t in X is a function from paths(X) to $N \cup S \cup \{\perp\}$ such that:

- $t(r) \neq \perp$
- If $p \in Epaths(X)$ then $t(p) \in N \cup \{\perp\}$
- If $p \in paths(X) - Epaths(X)$, then $t(p) \in S \cup \{\perp\}$
- If $t(p1) = t(p2)$ and $t(p1), t(p2) \in N$ then $p1=p2$

- If $t(p1) = \perp$ and *p1* is a prefix of *p2*, then $t(p2) = \perp$
- $\{p \in paths(X) \mid t(p) \neq \perp\}$ is finite.

The set of all tuples in X is defined as *T(X)*

Example 2.3.1: Suppose that X is the XSD shown in Figure 2.2 then a tuple in X assigns values to each path in paths(X):

t(school)=n0
t(school/department)=*n1*
t(school/department/Dname)=CIS
t(school/department/course)=*n3*
t(school/department/course/Cno)=10
t(school/department/course/Cname)=Database
t(school/department/course/student)=*n6*
t(school/department/course/student/Sno)=1
t(school/department/course/student/Sname)=Ahmad
t(school/department/course/student/age)=35
t(school/department/course/student/grade)=B
t(school/department/office)=*n34*
t(school/department/office/room-no)=100
t(school/department/office/Oname)=secrtary
t(school/department/office/address)=buld-1

So we can present tuple t1, by collecting all values together as follows:

t1(X)= (n0,n1,CIS,n3,10,Database, n6, 1, Ahmad, 35, B, n34,100, Secratry, Build-1), in the same way we can present tuple t2 for the same schema by assigning another values from the XML document representation in Figure 2.5, as follows:

t2(X)= (n0,n1,CIS,n3,10,Database, n11, 2, Faris, 40, A, n34, 100, Secratry, Build-1).

Definition 2.3.2: $tree_x(t)$ (Arenas & Libkin, 2004)

Given an XSD $X = (CE, SE, A, F, root, C)$ and a tuple $t \in T(X)$, $tree_x(t)$ is defined to be an XML tree $T = (N, root, lab, comp, val)$ with root = t(r) such that

- $N = \{n \in Node \mid \exists p \in Epaths(X) suchthat, n = t(p)\}$

- if $n = t(p)$ and $n \in N$ then $label(n) = last(p)$

- $comp(n) = \{t(p') \mid t(p') \neq \perp \& p' = p.e, e \in EL, or, p' = p.s\}$ for $n \in N$ and $n = t(p)$

Generally, an XML tree can be described as a set of tree tuples. In such a representation each tree tuple is with the maximal information.

Example 2.3.2: Let X be the XSD from Figure 2.2 and t be the tuple from Example 2.3.1 that gives rise to the following XML tree tuple in Figure 2.4:

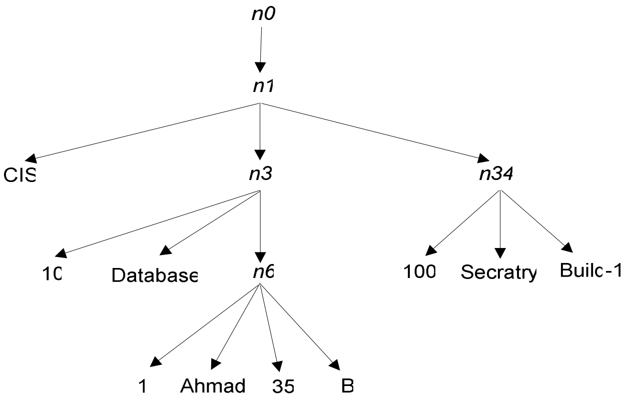


Figure 2.4: Tree tuple $tree_x(t)$

Definition 2.3.3: Node equal and value equal

When considering data redundancy by normalization process, it is important to compare the nodes in the tree and detect value equality between them (Yan & Lv, 2006; Zhao et al., 2009).

Two nodes n_1, n_2 are called value equal denoted as $n_1 =_v n_2$ so that

If n_1, n_2 is an simple element or attribute then

1) $label(n_1) = label(n_2)$

2) $val(n_1) = val(n_2)$

Otherwise, if n_1, n_2 is complex elements called node equal if

1) $label(n_1) = label(n_2)$, or

2) If the attribute $a \in n_1$ there is an attribute $b \in n_2$ such that $a = b$ and vise versa.

3) The sequence of their children elements is equal in pairs, which is mean $Comp(n_1) = comp(n_2)$

Let's consider the XML document of XSD in Figure 2.2 with data as in Figure 2.5. The two data elements (e.g., node 8 and node 26) are value equal which have the same value name Ahmad. While two complex elements are node equal if and only if the sub-trees rooted at those two elements are identical when the order among sibling elements is ignored.

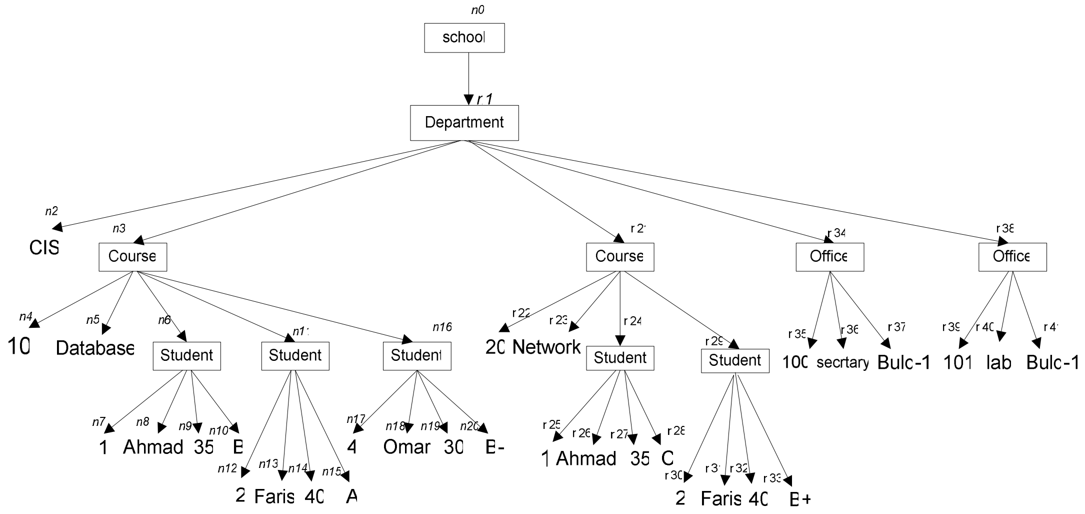


Figure 2.5: XML document tree with data redundancy

2.4 Keys and Foreign Keys for XML Documents

In recent years, the increasingly use of data centric approach of XML has necessitated to enrich the semantics of XML data which can be done through using keys**.** Keys are an important part of any data model, as well as in XML database. It is considered as one of those integrity constraints which specify the way that the elements are associated to each other (Ahmad & Ibrahim, 2008), and identifies the scope of uniqueness over XSD level (Buneman et al., 2001). Just like in relational database model, the key establish the connection between a real word object and its representation in the database thus enabling information about an object to be located in the database (Vincent et al, 2004). The XML needs for specification of keys to help in locating data in an XML document and in enforcing semantic integrity constrains, in order to prevent incorrect tuple insertion in the XML schema, furthermore they are using for indexing and querying optimization. Shahriar & Liu, (2008) proposed that XML key is preserving transformation of XML which can be used in XML-to-XML data transformation and integration.

Keys of DTD are defined in terms of ID and IDREF attribute which can identify uniqueness of element within an XML document, but with limited scope in the entire document; however it is not clear that ID attributes are used as keys rather than internal pointers (Buneman et al., 2001). While the XSD supports the definition of key and foreign key concepts and has precise way for specifying them through the use of XPath language (Clark & DeRose, 1999). The key function confirms uniqueness and asserts that all selected content actually has such tuples, furthermore it confirm that the value of key has to be not null. The location of the key element in the schema provides the context node in which the constraint holds. The constraint place under the selector XPath element is a key that refers to the field XPath element (Connolly & Begg, 2010).

(Wu et al., 2002; Zhao et al., 2009) classified keys into two types, absolute key and relative key, the absolute key indicating that the simple element key of significant complex element has uniquely identified it, otherwise the relative key indicating that more than one simple element key in different level have uniquely identified significant complex element.

In this section we define the key and foreign key based on XSD schema. Our definition adopted from ((Necasky & Pokorny, 2007) by using the XPath expression as follows:

Definition 2.4.1: XML Key (XK)

To define a key constraint, we specify a unique element that can determine uniquely other simple elements or attributes in the whole XML document (Zainol & Wang, 2010). Like in relational database key, it can be defined to include one or more fields which are called a composite key (Provost, 2002). The formal definition of key in XSD is as follows:

Given an XSD $X = (CE, SE, A, F, root, C)$, the XK is a key of X schema that defined as $K(P_S, \{P_{f,1}, ..., P_{f,n}\})$ represented primary keys (which are to be unique and cannot be null) (Necasky & Pokorny, 2007), where

n>0,

P_S is a selector path on XSD that specifies the complex element that hold fields with uniqueness constraint.

$\{P_{f,1}, ..., P_{f,i}, ..., P_{f,n}\}$ are a set of field paths that represent the nodes to be checked for their value equality or uniqueness.

The following expression corresponds to the key in XSD:

```
<xs:key>
        <xs:selector xpath=" $P_s$ "/>
        <xs:field xpath=" $P_{f,1}$ "/>
            ....
        <xs:field xpath=" $P_{f,n}$ "/>
</xs:key>
```

Example 2.4.1 defines a key for the XSD in Figure 2.2. The key named `departmentKey2` define a unique constraint on the simple element cno which is under the complex element course. The location of the unique element in the schema provides the context node in which the constraint holds. So, by placing this constraint under the course element, we specify that this constraint has to be unique within the context of a course element only. This constraint is analogous to specifying a constraint on a relation in relational database.

Example 2.4.1

```
<xs:key name="departmentKey2">
        <xs:selector xpath=".//mstns:course" />
        <xs:field xpath="mstns:cno" />
</xs:key>
```

Definition 2.4.2: XML Foreign Key (XFK)

The foreign key in XSD is defined by the use of keyref function. It specifies association between nodes of XSD and asserts similar constraints on the value of referencing nodes (Provost, 2002).

The formal definition of the XFK is as follows:

$XFK = (XK, P_{SR}, \{P_{fR,1}, ..., P_{fR,m}\})$, (Connolly & Begg, 2010), where:

XFK is defined as foreign key referred to be constrained to *XK* key.

P_{SR} is the selector reference path that specifies the complex element that hold reference fields.

$P_{fR,1}$,........., $P_{fR,m}$ is the set of fields reference paths that consider as a foreign key referred to the key in field of *XK*. The following expression corresponds to the foreign key in XSD:

```
<xs:keyref name=" XFK " refer "XK ">
    <xs:selector xpath=" $P_{SR}$ "/>
    <xs:field xpath=" $P_{fR,1}$ "/>
     ….
    <xs:field xpath=" $P_{f,n}$ "/>
</xs:keyref>
```

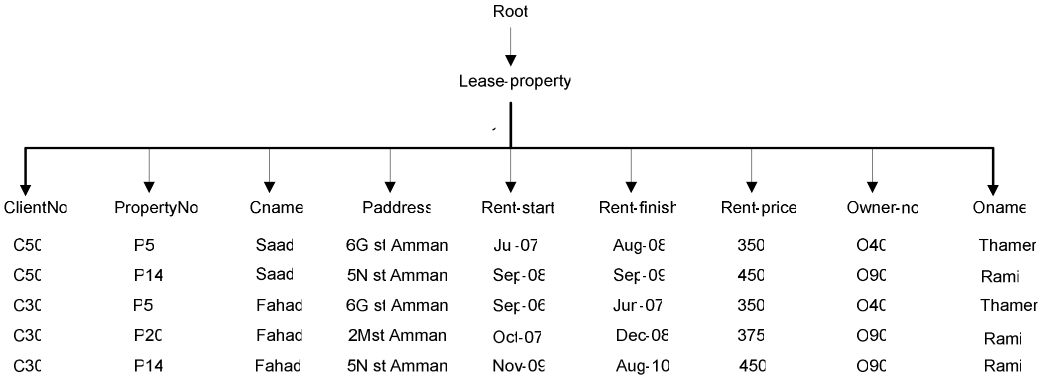
2.5 XML Schema Representation

Yu & Jagadish, (2008) referred to two types of representation in XML schema which is either flat or hierarchical representation.

2.5.1 Flat XML Schema

The flat XML data is common due to its simplicity way in publishing XML data. It has little characteristics and no nesting of elements and such databases model their data mainly as attributes. The flat XML schema consists of single complex element under the root which contains many children of simple elements in the same level (Lee at el., 2002).

Example 2.5.1: Consider an XML document tree in Figure 2.6 which shows a sample of flat XML database about leasing a property. It consists of complex element named leases property with unbounded occurrence. The leases property element contains the following simple elements (clientNo, Cname, propertyNo, Paddress, rent-start, rent-finish, rent-price, Ownerno, and Oname) as children of complex element lease property.



ClientNo	PropertyNo	Cname	Paddress	Rent-start	Rent-finish	Rent-price	Owner-no	Oname
C50	P5	Saad	6G st Amman	Jul-07	Aug-08	350	O40	Thamer
C50	P14	Saad	5N st Amman	Sep-08	Sep-09	450	O90	Rami
C30	P5	Fahad	6G st Amman	Sep-06	Jun-07	350	O40	Thamer
C30	P20	Fahad	2Mst Amman	Oct-07	Dec-08	375	O90	Rami
C30	P14	Fahad	5N st Amman	Nov-09	Aug-10	450	O90	Rami

Figure 2.6: Flat XML document tree for lease property database

2.5.2 Hierarchical XML Schema

Hierarchical XML schema is inspired from the concept of nested relation. It consists of many complex elements with many simple elements in different levels. It is structured as a hierarchical tree (Yu & Jadadish, 2008). The XML document tree in Figure 2.5 is an example of hierarchical XML database.

Chapter 3

Enhancement of Functional Dependencies Definition for XML

In this chapter, we present the definition of Functional Dependencies for XML (XFDs), and then we introduce our improvement definition of XFD**.** We also present the types of dependencies according to the type of representation. Finally, we introduce the notion of XML normal form (XNF).

3.1 Introduction

The hierarchical structure of XML document allows redundancy of data with its elements which may be nested and repeated. This will make the same information appeared in more than one place; which means the same elements appear at different sub-trees. Existence of such redundancy can lead to waste of storage space and to anomalies in recover information (Ahmad & Ibrahim, 2009). Similar to traditional databases, we can identify three kinds of update anomalies in a badly designed XML database: insertion anomaly, rewriting anomaly and deletion anomaly.

One strategy to avoid data redundancies is to design redundancy-free schema, which are formalized by means of data dependencies (Pankowski & Pilka, 2009). The data dependencies are considered as part of the real word semantics. They present the semantic information in the form of relationships between different elements in the XML documents (Arenas & Libkin, 2004). Similarly to relational database, the data dependency in XML can be categorized into Key dependencies and functional dependencies. One good strategy is that the data dependency should be modeled in the start of design stage for a correct and complete database representation of semantic (Zainol & Wang , 2010).

3.2 Functional Dependencies

A functional Dependency is considered as one of the most popular data dependencies for relational databases which describes the property that the values of some attributes of a tuple uniquely determine the values of other attributes of the tuple. Similarly is the definition for XML but the difference is that attributes and tuples are basic units in relational database, while in XML model they must be defined using paths of tree tuple (Ahmad & Ibrahim, 2009)

Just like in traditional databases, the concept of functional dependency for XML has played a centric role in providing richer data semantic which is important in normalizing XML schema. This concept has been widely investigated over the past few years (chen & Liao, 2010).

Although some proposals have been made, there seems to be no consensus on how to define XML functional dependencies (Chen & Liao, 2010). In general, there are two main approaches in XML research community which are different in how to specify the target elements of the constraint (Ahmad & Ibrahim, 2008). The first approach is the path-based (Vincent et al., 2004) where the target elements are implicitly encodes inside the functional dependency specification. The second approach is tuple-based (Arenas & Libkin, 2004) which specifies the target elements independent of each individual functional dependency specification. However, both approaches are valid ways for defining XML FDs, but the tuple-based approach has clearer semantics and is conceptually similar to the relational FD notion (Yu & Jagadish, 2008).

In this section we define the XML Functional Dependency (XFD) as given by (Lee et al., 2002; Yan & Lv, 2006) and then introduce our definition with some improvement to there definition.

Definition 3.2.1: Functional Dependency for XML (XFD) (Yan & Lv, 2006)

Given XSD $X = (CE, SE, A, F, root, C)$ then the functional dependency defined as:

$XFD = (P_h, [P_{x1} ,....., P_{xn}] \rightarrow [P_{y1}, ..., P_{ym}])$ where:

P_h is the header path of XFD which defines the longest common repeatable path for both the left hand side (LHS) and the right hand side (RHS), and it is starting with the root node. The header path specifies the scope of XFD in which the constraint holds, and defines the node set in which the functional dependency holds.

The scope of XFD specified by last element of header path, such that $last(P_h) \in CE$.

If $P_h \neq \phi$ and $P_h \neq root$, then the scope of XFD is called local which means that the scope of functional dependency is the sub-tree rooted $last(P_h)$; otherwise, when $P_h = root$ then it is called a global functional dependency which holds the scope overall the schema X.

$P_{x1} ,......, P_{xn}$ is called the left hand side paths of the XFD which determine the other side, and $last(P_{xi}) \in CE \cup SE \cup A \cup S$.

$P_{y1}, ..., P_{ym}$ is called the right hand side of the XFD which functionally depend on the left side, and $last(P_{yi}) \in CE \cup SE \cup A \cup S$.

Which mean, for any two instance of tree tuples *t1, t2* identified by the XFD header P_h, if all LHS tree tuples agree on their values, then they must also agree on the value of the RHS tree tuples such as: $t_1.P_{xi} = t_2.P_{xi}$ imply $t_1.P_{yi} = t_2.P_{yi}$.

Definition 3.2.2 Our Improvement Definition of XFD

Here we made some improvement to the above XFD definition to suit the hierarchical structure of XML schema by using the XPath expression and adding the level of last element of each path. The last element of the paths for both sides may be located in the same level or at different levels which are important in specifying dependency in hierarchical schema. We represented the XFD as follows:

Given XSD $X = (CE, SE, A, F, root, C)$ then the functional dependency defined as:

$XFD = (P_h, [./x1, l_{1...i}, ..., ./xn, l_{1...i}] \rightarrow [./y1, l_{1...i}, ..., ./ym, l_{1...i}])$, where:

P_h is the header path of XFD which defines the longest common repeatable path that is a prefix of both left hand side and right hand side and it is starting with the root node. The header path specifies the scope of XFD in which the constraint holds, and defines the node set in which the functional dependency holds.

The scope of XFD is specified by last element of header path, such as in the previous definitions.

$./x1, l_{1...i}, ..., ./xn, l_{1...i}$ are the set of last elements of paths for the left hand side of the XFD which determine the other side, and represent $last(P_{x1})$,.... , $last(P_{xn})$ respectively.

$l_{1...i}$ is to specify the level of the last element of the paths, where bounded from 1 to i.

$./y1, l_{1...i}, ..., ./ym, l_{1...i}$ are the set of last elements of paths for the right hand side of the XFD which functionally depend on the left hand side, and represent $last(P_{y1})$,…., $last(P_{ym})$ respectively.

$l_{1...i}$ is to specify the level of the last element of the paths, where bounded from 1 to i.

Our definition of XFD is necessary and sufficient to specify the constraint that enriches the XML schema. It defines the syntax and semantics precisely by allow dealing with not only string values but also elements of both types (complex elements and the simple elements) in the XSD schema. Furthermore, specifying the level of elements can help in capture the type of dependency as we will clarify in section 3.3.

The characteristics of functional dependency are useful for determining redundancy and normalization process, in some case if a specific value of left hand side are repeated in several tuples in table for some reason then the value of right hand side in these tuples are forced to be the same, such case may cause data redundancy which lead to some anomaly (Ahmad & Ibrahim, 2009).

Example 3.2.2.1: Let's consider the constraints information in the XSD schema for school management database shown in Figure 2.2 is as follows:

1- In the total document, the student number determines student name and age.

2- All in the school, course no determine course name.

3- In some departments, the room number determines the office name which valid in the entire document.

4- The department’s name determines the address of the office (supposed that every department locates in a certain building.

By applying the definition of XFD, the data constraints of XSD schema in Figure 2.2 can be represented the functional dependency in the following forms:

XFD(1) $(school/department/course/student,[./Sno,l4 \rightarrow ./Sname,l4,./age,l4])$

XFD(2) $(school/department/course,[./Cno,l3 \rightarrow ./Cname,l3])$

XFD(3) $(school/department/office,[./room\_no,l3 \rightarrow ./Oname,l3])$

XFD(4) $(school/department,[./dname,l2 \rightarrow .//office/address,l3])$

The XFD(1) imples that this XFD holds over the sub-trees rooted at student, this functional dependency states that student number (Sno) in level 4 can uniquely determines student name (Sname) and student age (age), in the same level and under the same sub-tree student; XFD(2) implies that the course number(Cno) in level 3 uniquely determines course name (Cname) in level 3 under the sub-tree rooted at Course; XFD(3) states that the room number (room-no) in level 3 can uniquely determines office name (Oname) in the same level and holds under the sub-tree rooted at the node office; and XFD(4) states that department name(Dname) in level 2 under sub-tree department can uniquely determines the element of the RHS (address) which is in level 3 and under sub-tree office.

To apply our definition of functional dependency according to XML tuple definition, consider the XFD(1) of Example 3.2.1

XFD(1) $(school/department/course/student,[./Sno,l4 \rightarrow ./Sname,l4,./age,l4])$

We defined a set of paths instances for school database in section 2.2.3.2.

Let the left hand side path that ends with Sno is

p(x1)=school/department/course/student/Sno

and the two paths of the right hand side are:

p(y1)=school/department/course/student/Sname,

p(y2)=school/department/course/student/age

and we define two tuples in Example 2.3.1 as follows

t1(X)= (n0,n1,CIS,n3,10,Database, n6, 1, Ahmad, 35, B, n34,100, Secratry, Build-1)

t2(X)= (n0,n1,CIS,n3,10,Database, n11, 2, Faris, 40, A, n34, 100, Secratry, Build-1)

From the definition of functional dependency we have: for any two instances of tuples *t1, t2* identified by the XFD header path P_h, if the last element of paths for

LHS in two tuples are agree on their values, then they must also agree on the values of the last elements of paths for RHS in the same tuples such as

$t_1.(./x_i)=t_2.(./x_i)$ imply $t_1.(./y_i)=t_2.(./y_i)$

For our example $t_1.(./x_i)=$ 1 and $t_2.(./x_i)=$ 2, which is mean the student number for tuple 1 not equal the student number for tuple 2, and hence student name in tuple1 not equal student name in tuple2 ($t_1.(./y_i)=$ Ahmad) ≠ ($t_2.(./y_i)$ =Faris).

So the functional dependency is achieved according to the constraint, which specify that for each student has unique student number.

Example 3.2.2.2: In similar manner we define the functional dependency to flat XML document for lease property database in Example 2.5.1 as follows:

XFD (1)

$(root/lease-property,$
$[./clientNo,l2,./propertyNo,l2 \rightarrow ./rent-start,l2,./rent-finish,l2])$

XFD (2)

$(root/lease-propert[./clientNo,l2 \rightarrow ./Cnam,l2])$

XFD (3)

$(root/lease-property[./propertyNo,l2$
$\rightarrow ./Paddress,l2,./rent-price,l2,./Ownerno,l2,./Oname,l2])$

XFD (4)

$(root/lease-property[./Ownerno,l2 \rightarrow ./Oname,l2])$

With the help of identified functional dependencies we identify a primary key for the schema which are clientNo, and propertyNo, both of them in level 2. In XFD(1) the elements start date of rent property (rent-start) in level 2 and finish date of rent property (rent-finish) in the same level depend fully on the composite keys (clientNo & propertyNo). XFD(2) the client number (clientNo) in level 2 uniquely determines client name (Cname) in the same level. XFD (3) the property number (propertyNo) in level 2 determines property address (Paddress), rental price (rent-price), the owner number (Ownerno), and the owner name (Oname) which are all in the same level 2.

3.2.3 Discussion about Definition of XFD

When comparing our XFD definition with previous researches definitions we can conclude the following:

- It is a good idea to extend the concept of relational model to define XFD such as in ((Arenas & Libkin, 2004); (Yu & Jagadish, 2008)), but it is important to consider the structural difference between the two models, since the hierarchical structure makes the information items related to XFD may appear at different levels of XML tree. Thus with our definition we reflect the feature of hierarchical structure by specifying the level number.

-It is important to specify the way to express the involved information items. Many researches ((Lee et al, 2002); (Vincent et al, 2004); (Yan & Lv, 2006)) using the path to express the node of XML document, but with expressive power of XPath language we can determine richer cases that involved information items; furthermore we can capture a set of nodes with complex and simple type in order to define the value equality for element nodes which is important in determining XFD.

-((Yu & Jagadish, 2008); (Ahmad & Ibrahim, 2009)) proposed that the XML documents has scope due to the nested tree structure of XML schema specifies by Global and local, and our definition also capture this characteristics.

-Finally, the functional dependency in relational data model consider string values only as relational attributes are simple data types, while our definition consider string values and complex element nodes as XML schemas not only have simple data types but also nodes of complex types.

3.3 Types of Dependencies

3.3.1 Types of Functional Dependencies for Flat XML Schema

As mentioned in Chapter 2, the flat XML schema representation consists of single complex element under the root which contains many children of simple elements. We analogous the flat XML schema to relational database structure, by considering the single complex element as a table, and its children nodes as attributes of the relational table. Therefore we extend some thoughts of normalization in relational database.

The following definitions of XFDs for flat XML schema, which we adopted from (Lv & Yan, 2007) are given. We have introduced some modifications to those XFDs to make them suitable to the hierarchical structure of XML schema.

3.3.1.1 Partial XFD for Flat XML Schema (PXFD/F)

Definition 3.3.1.1 PXFD/F

Let *P1, Pn* are two paths that ends with key elements *x1* and *xn* respectively and the levels for both keys are *li*, so *x1, xn* consider as composite keys, then the Full XML Functional Dependency in Flat schema (FXFD/F) is:

Let $XFD = (P_h[./x1, li,./xn, li \rightarrow ./y1, li,....,./ym, li])$

Where all last elements of the paths in the RHS are depend on both specified keys. Therefore the Partial XML Functional Dependency in Flat schema (PXFD/F) is:

$(P_h[./x1, li \rightarrow ./y1, li,........,./ym, li])$

Which means the elements in the RHS functionally depends on part of the composite keys. Example 3.3.1.1 clarifies the full and partial XFD for flat XML schema

Example 3.3.1.1: The XFD (1) in Example 3.2.2.2 is full XFD (FXFD/F)

$(root/lease - property[./clientNo,l2,./propertyNo,l2 \rightarrow ./rent - start,l2,./rent - finish,l2])$

While XFD(2) and XFD(3) are both Partial XFD (PXFD/F) $(root/lease - propert[./clientNo,l2 \rightarrow ./Cnam,l2])$, and

$(root/lease - property[./propertyNo,l2 \rightarrow ./Paddress,l2,./rent - price,l2,./Ownerno,l2,./Oname,l2])$

Hence the schema has redundancy due to the anomalies in partial functional dependency which is clear in tree representation in figure (2.6) that the client name (Saad) and (Fahad) are redundantly stored in document which may lead to update anomaly, for example if we wish to update client name (Fahad) that has number (C30), we have to update in the three sub-tree nodes (lease-property). The same redundancies appear through the third functional dependency.

3.3.1.2 Transitive XFD for Flat XML Schema (TXFD/F)

Definition 3.3.1.2 TXFD/F

Let the paths *Px* ends with *x* elemnt*, Py* ends with *y* element*,* and *Pz* ends with *z* element*,* and the levels for all final elements of paths are *li*

Let $XFD = (P_h,[./x,li \rightarrow ./y,li])$ and $XFD = (P_h,[./y,li \rightarrow ./z,li])$ then

$XFD = (P_h,[./x,li \rightarrow ./z,li])$

Which means that the path ends with element *z* is transitively depends on the path that ends with the element *x*. In another word the non key element transitively determines another non key element. This type of functional dependency is called Transitive XFD for Flat schema (TXFD/F). Example 3.3.1.2 clarifies the TXFD/F.

Example 3.3.1.2: The XFD (4) in Example 3.2.2.2 about lease property database $(root/lease-property[./Ownerno,l2 \rightarrow ./Oname,l2])$ is (TXFD/F).

The non-key element owner number (Owner-no) transitively determines other non-key element owner name (Oname), which cause clear redundancy in the document in Figure 2.6, that may lead to update anomalies. For example, if we want to update the name of an owner, such as the owner name (Rami) that has number (O90), we have to update these elements in all repeated sub-tree of (lease-property). If we update only in one sub-tree and not in the other, the XML database would be in an inconsistent state.

3.3.2 Types of Dependency for Hierarchical XML Schema

Zhao et al., (2009) classified XFD in hierarchical schema to Absolute and relative. They proposed that the functional dependency that has the elements of both sides in the same level is absolute functional dependency. While if the elements of both sides in different level is relative dependency, which means that the elements of RHS relatively depend on elements of the LHS.

3.3.2.1 Absolute XML Functional Dependency for Hierarchical Schema (AXFD/H)

AXFD/H holds in hierarchical schema if there is a dependency between simple elements of the same corresponding complex element at the nodes in the bottom of schema. This notion adopted from (Arenas & Libkin, 2004; Zhao et al., 2009)

Definition 3.3.2.1 AXFD/H

Let's consider the $XFD = (P_h,[./x1,l_{1...i},......,./xn,l_{1...i}] \rightarrow [./y1,l_{1...i},......,./ym,l_{1...i}])$ is absolute functional dependency when

1) The last element of the header path is the element that represents the root of sub tree that contains elements of both sides of dependency

2) The level of the last element of the paths in both sides of dependency is the same and refers to terminal level in the schema

The XFD(1) for the schema in Figure 2.2 of an example 3.2.2.1 is an absolute functional dependency.

XFD(1) $(school/department/course/student,[./Sno,l4 \rightarrow ./Sname,l4,./age,l4])$

The header path ends with the element student, which is the sub tree root to the elements Sno, Sname, and age. This dependency specifies that each student must have student number as identifier key that determines the name of student and the age. While the same student takes many courses in the department, then his name and age repeated for each course. That means redundancy in the name and age.

3.3.2.2 Relative XML Functional Dependency (RXFD)

The relative dependency holds when a subset of its LHS is a key from different level. That mean the RHS elements depend on LHS elements in different levels (Zhao et al., 2009**.** We integrated this definition with the definition of (Wu et al., 2002), so we classified relative dependency into two definitions: Relative Transitive XFD and Relative Full/Partial XFD.

3.3.2.2.1 Relative Transitive XFD (RTXFD/H)

Definition 3.3.2.2.1 (RTXFD/H)

Transitive functional dependencies between complex elements occur if there is an attribute or a simple element node has dependency with another simple element node from different level.

Let the paths *Px* ends with *x* element in level *i, Py* ends with *y* element in level *i+1,* and *Pz* ends with *z* element in level *i+1,*

Let's $XFD = (P_h, [./X, l_i \rightarrow ./Y, l_{i+1}])$ & $XFD = (P_h, [./Y, l_{i+1} \rightarrow ./Z, l_{i+1}])$ then $XFD = (P_h, [./X, l_i \rightarrow ./Z, l_{i+1}])$

This definition means the element (*x*) in level *i* transitively determines the element (*z*) in level *i+1*.

The transitive XFD between complex elements occur if theirs simple element has dependency with another simple element node from different level.

Example 3.3.2.2.1

Consider the schema in Figure 2.2 of an example 3.2.2.1 has the following functional dependencies:

$(school/department, [./dname, l2 \rightarrow ./office/room-no, l3])$,

$(school/department, [./office/room-no, l3 \rightarrow ./office/address, l3])$

These two functional dependencies represent the XFD(4) $(school/department, [./dname, l2 \rightarrow .//office/address, l3])$,

which assert the transitive dependency. Where the key (dname) in level 2 under the complex element department is relatively determines the simple element (address) in level 3 under complex element office. It is obvious from the corresponding level that the element dname is not in the same level of the RHS element address.

The relative dependency can cause redundancy when violated. In Figure 2.5 the tree representation of XML document for school management database, when the department name (CIS) has many offices nodes, the room number (100) which is the room name (secrtary) is located in building (buld-1), and the same department (CIS) has office with room number (101) which is the room named (lab) located in building (buld-1). So in the department (CIS), the address of offices will repeat in each office node, which is mean redundancy in repeating the address. In next chapter we remedy this redundancy.

3.3.2.2.2 Relative Full and Partial XFD (RFXFD and RPXFD)

Definition 3.3.2.2.2 (RFXFD and RPXFD)

Let $./X,l_i,./Y,l_{i+1},./Z,l_{i+2},./S,l_{i+3}$ be last elements of paths *Px, Py, Pz*, and *Ps* in levels *i, i+1, i+2,and i+3* respectively, and the elements *x, y, z* are keys for their parent complex elements. The following XFD is definition of relative full functional dependency (RFXFD/H).

Let $XFD = (P_h,[./X,l_i,./Y,l_{i+1},./Z,l_{i+2} \rightarrow ./S,l_{i+3}])$ is relative full XFD, then $XFD = (P_h,[./X,l_i \rightarrow ./S,l_{i+3}])$ or $XFD = (P_h,[./Y,l_{i+1},./Z,l_{i+2} \rightarrow ./S,l_{i+3}])$ is Relative Partial XML functional dependency (RPXFD/H). In the first relative partial XFD we have one key element in the LHS, such case is considered as special case of relative transitive, hence we determine the XFD that has more than one key element part of composite key in deferent level as RTXFD.

That means when the document has many keys for complex elements located at different levels, then if any simple element depends on more than one key part of the composite keys in other level, then it is relative partial XFD.

As the example of school management document dose not contain the RTXFD redundancy we consider another example shows in Figure 3.1 a tree representation of XML document for typical Project-Supplier-Part database which consists of the root element (PSJ), complex element (project) under the root, each project element has project name(Pname) as a key and another complex element (supplier), each supplier element has supplier name (Sname) as a key and a complex element (part) as final complex element, each part element has part number (PartNo) as key and two simple elements (price) and (quantity).Suppose we have a constraint that a supplier must supply a part at the same price regardless of projects. This information is useful to anyone using this XML database as it can alert them to violation of this integrity constraint. The following two XDF are defined for the schema in Figure 3.1:

XFD(1)

$(PSJ,[./project/Pname,l2,./\mathrm{sup}plier/Sname,l3,./Part/PartNo,l4 \rightarrow ./part/Quantity,l4])$

XFD(2)

$(PSJ/project,[./\mathrm{sup}plier/Sname,l3,./Part/PartNo,l4 \rightarrow ./Part/\mathrm{Pr}ice,l4])$

The first functional dependency represents relative full XFD. The second functional dependency represents relative partial XFD, which states that the price depend on supplier name and part number regardless of project. This dependency is violated as clear in the instance in Figure 3.1. Supplier "ABC Trading" sells part number "P700" at price "80" to project "Garden", but sells the same part to project "Road Work" at price "10".

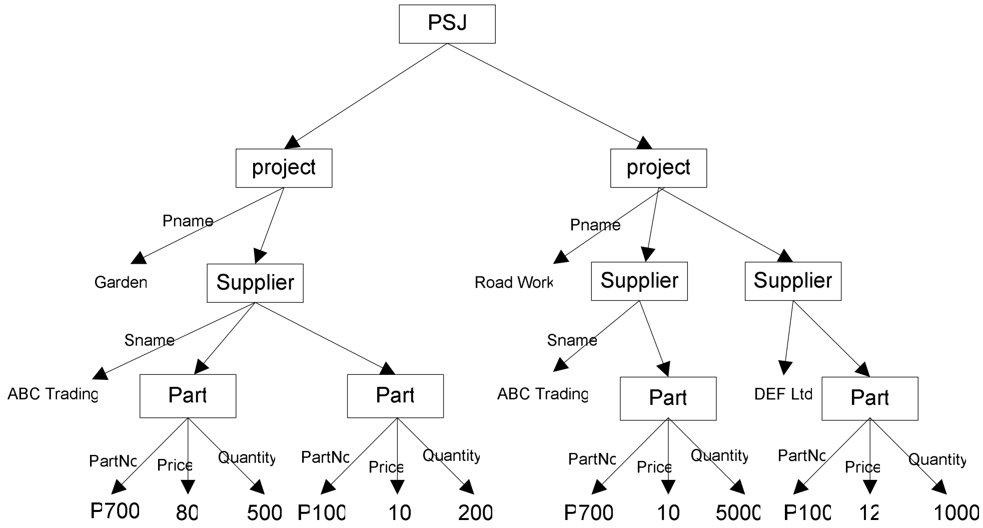


Figure 3.1: Tree representation for Project-Supplier-Part database

3.4 Normal Form for XML

We now give the definition of XML Normal Form (XNF) based on defined dependencies.

Given XSD $X = (CE, SE, A, F, root, C)$, and a set of XFD over the schema X.

The schema X is in normal form if and only if:

1) X has at least one key.
2) There is no non-trivial Absolute Partial or Transitive XFD for flat schema.
3) There is no non-trivial Absolute XFD for hierarchical schema.
4) There is no non-trivial Relative Transitive XFD.
5) There is no non-trivial Relative Partial XFD.
6) For any trivial XFD of the form $(P_h,[./X,li \to ./Y,li])$ satisfied by schema X, where *X* and *Y* are last elements of LHS paths and RHS paths respectively, then either X is a key or Y is part of the key in schema X.

Chapter 4

XML Schema Normalization

The main goal of normalization process is to convert an initial schema into one in a normal form to reduce anomalies and redundancies in the XML document. In this chapter, we present normalization rules that we used to transform the un-normal form of XSD into a normal one. We also, present the flowchart of normalization process and the case tool we developed to automate the process of XML database normalization. Finally, we present examples to illustrate how the XSD is restructured to XNF.

4.1 Introduction

As XML increasingly has become the more common for representation and natively storing of data on the web, it is unavoidable that the problem of storing redundant data, modifying, and maintaining are become touching for many applications. The problem of redundant data and operation anomalies occur in XML documents if their type structure are not well-formed. To avoid these problems, it is important to begin with building an XML application with designing a good XML schema (Pankowski & Pilka, 2008).

Similar to relational database design, normalization rules are used to help designers to design a good XML schema which can follow either of two methodologies: the conceptual approach (Wu et al, 2001; Zainol & Wang, 2010) or the normalization theory approach (Arenas & Libkin, 2004; Yu & Jagadish, 2008). Although normalization theory of relational database has matured, there is no such mature and systematic theory for XML world and it can not be applied directly in

XML schema due to the different in structural nature of XML from relational database. The hierarchical, irregular structure of XML make the task of designing XML documents becomes more challenging than in relational database (Lv & Yan, 2007).

The goal of normalization design of XML database schema is to convert an initial poorly designed schema into one of normal forms which eliminate redundancies and update anomalies (Wu, 2004). In our research we integrated normalization rules proposed by number of researchers, and used these rules to design and implement a case tool to perform the process of normalization of XML schema design.

4.2 Normalization Rules

In (Arenas & Libkin, 2004) an XNF decomposition algorithm was proposed that combines two basic ideas: creating a new element, and moving an attribute. These two ideas are the basic for the following elimination rules.

Rule-1: Eliminate Absolute XFD (EAXFD)

This type of elimination is used when there is a redundancy caused by a non-trivial PXFD/F, TXFD/F, and AXFD/H which holds between the elements under the same sub tree node.

Suppose *X* is a simple element key, and *Y* is a simple element, and both are under a complex element CE which means have the same level.

To eliminate the redundancy caused by $AXFD = (P_h, [./X, li \rightarrow ./Y, li])$, we do the following:

1- Create a new complex element name (new-CE), under the root.

2- Replicate the simple element key in LHS, and simple element node in RHS.

3- Make them as children to node (new-CE).

4- Make the copy of the LHS as key under the (new-CE).

5- Delete the simple element of RHS from original location.

Rule-1 procedure is illustrated by Example in Sub-Section 4.5.1.

Rule-2: Eliminate Relative Transitive XFD (ERTXFD/H)

This type of elimination works with the non-trivial Relative Transitive XFD that has one element key in LHS.

Let's $XFD = (P_h, [./X, l_i \to ./Y, l_{i+n}])$ & $XFD = (P_h, [./Y, l_{i+n} \to ./Z, l_{i+n}])$ then $XFD = (P_h, [./X, l_i \to ./Z, l_{i+n}])$ is RTXFD/H

To eliminate redundancy caused by non-trivial RTXFD/H, we do the following:

1- Replicate the Right hand side of XFD which is in level *i+n*.

2- Put it in the same level of the LHS element which is *i*.

3- Delete it from original location.

Rule-2 procedure is illustrated by Example in Sub-Section 4.5.2.

Rule-3: Eliminate Relative Partial XFD (ERPXFD/H)) (Enhancement based on the rule of decomposition algorithm (Arenas & Libkin, 2004))

This type of elimination works with the non-trivial Relative Partial XFD.

If $XFD = (P_h, [./X, l_i, ./Y, l_{i+1}, ./Z, l_{i+2} \to ./S, l_{i+2}])$ then $XFD = (P_h, [./Y, l_{i+1}, ./Z, l_{i+2} \to ./S, l_{i+2}])$ is Relative Partial XML functional dependency (RPXFD/H).

To eliminate redundancy caused by non-trivial RPXFD/H, we do the following:

1- Create new element-1 under the root.

2- Replicate the elements $./\,y, l_{i+1}$, $./\,Z, l_{i+2}$, and $./\,S, l_{i+2}$ with its sequence level of hierarchy.

3- Put the element $./\,y, l_{i+1}$ under the new element.

4- Create new element-2 under new element-1.

5- Put the elements $./\,Z, l_{i+2}$ and $./\,S, l_{i+2}$ under new element-2.

6- Make the elements $./\,y, l_{i+1}$ and $./\,Z, l_{i+2}$ in the new location as keys.

7- Delete the RHS element $./\,S, l_{i+2}$ from original location.

Rule-3 procedure is illustrated by Example in Sub-Section 4.5.3

4.3 Normalization Process

The flowchart of the normalization process is shown in Figure 4.1.

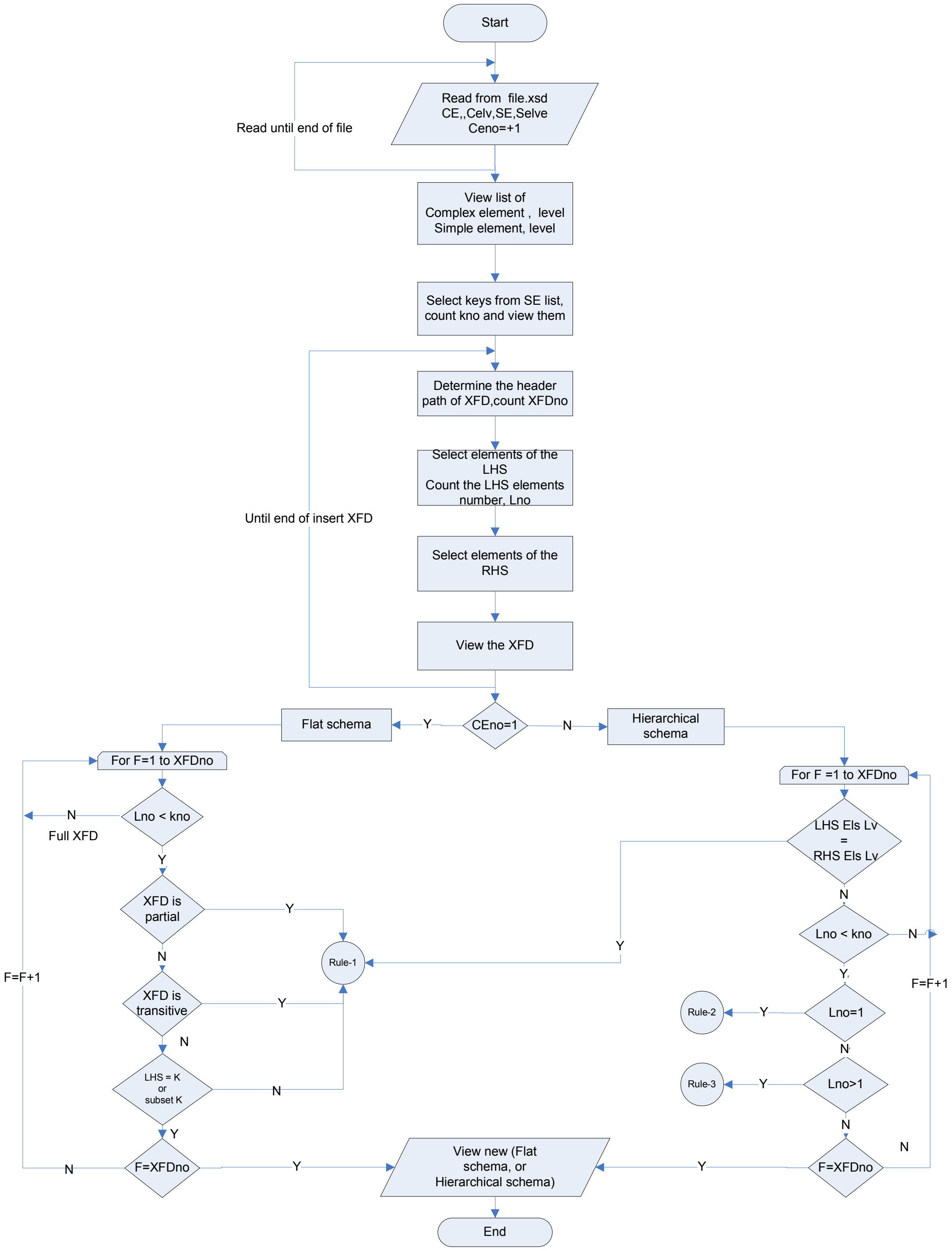


Figure 4.1: Flowchart of XSD Normalization process

4.4 XML Normalizer

Normalization in XML databases is an important stage in schema design. The process of doing that manually makes it difficult and takes so much time; furthermore the human may make mistakes in doing normalization.

In our thesis we have developed a case tool, called XML Normalizer that helps designers to perform the XML database schemas normalization quickly and accurately. This saves time and effort of XML database designers and thus frees them focus on other aspects of the XML database design process.

The main objectives of XML Normalizer are:

- To perform XML normalization process accurately.
- To reduce the time needed in perform the process of XML normalization.
- To avoid human error through normalize the XML schema process.

The XML Normalizer uses the three rules we presented in Section 4.2 to perform the normalization to XNF. A complete demo of the XML Normalizer is given in Appendix A.

4.5 Normalization Examples

In this section, we present examples to illustrate how the XSD is restructured according to XNF.

4.5.1 Example of Flat Schema Normalization

Consider the tree representation of XML document about lease property database shown in Figure 2.6, which has clear data redundancy due to the partial dependencies XFD (2): $(root/lease-propert,[./clientNo,l2 \rightarrow ./Cnam,l2])$ and XFD

(3): $$\begin{aligned}&(root/lease-property,[./propertyNo,l2\\&\rightarrow ./Paddress,l2,./rent-price,l2,./Ownerno,l2,./Oname,l2])\end{aligned}$$

, also it has transitive redundancy through

XFD (4) $(root/lease-property,[./Ownerno,l2 \rightarrow ./Oname,l2])$.

We apply rule-1 to eliminate these redundancies. To remedy XFD (2) the platform creates new element-1 under the root, copy (clientNo) and (Cname), put them under the new element-1, make (clientNo) as a key for new elemet-1, and delete Cname from lease-property complex element. To remedy XFD (3) the platform create new element-2 under the root, copy (propertyNo), (Paddress), (rent-price), (Ownerno), and (Oname), put them under new element-2, mark (propertyNo) as a key to new elemet-2 node, and delete (Paddress), (rent-price), (Ownerno), and (Oname) from lease-property node. Finally to remedy the transitive XFD (4), the platform creates new element-3 under the root, copy (Ownerno), and (Oname), put them under the new element-3, make (Ownerno) as a key to new element-3, and delete (Oname) from orginal location. The schema now is in normal form. Figure 4.2 shows an un-normalized XSD schema for lease property database and Figure 4.3 shows the normalized XSD schema for lease property database after applying elimination rules.

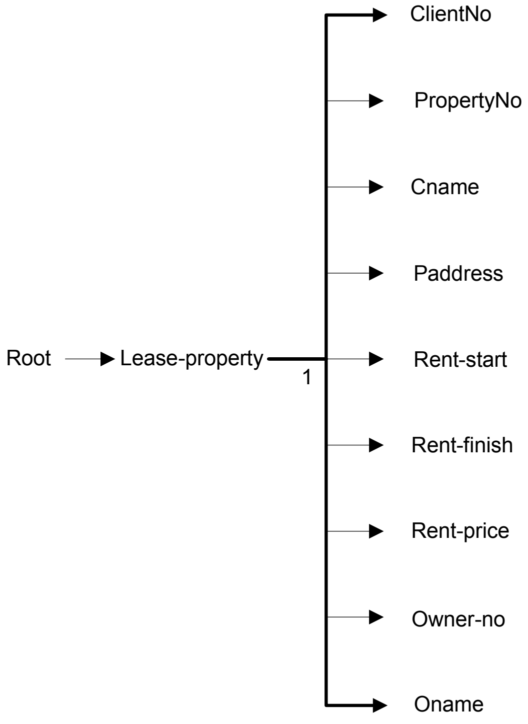


Figure 4.2: An un-normalized XSD schema for lease property database

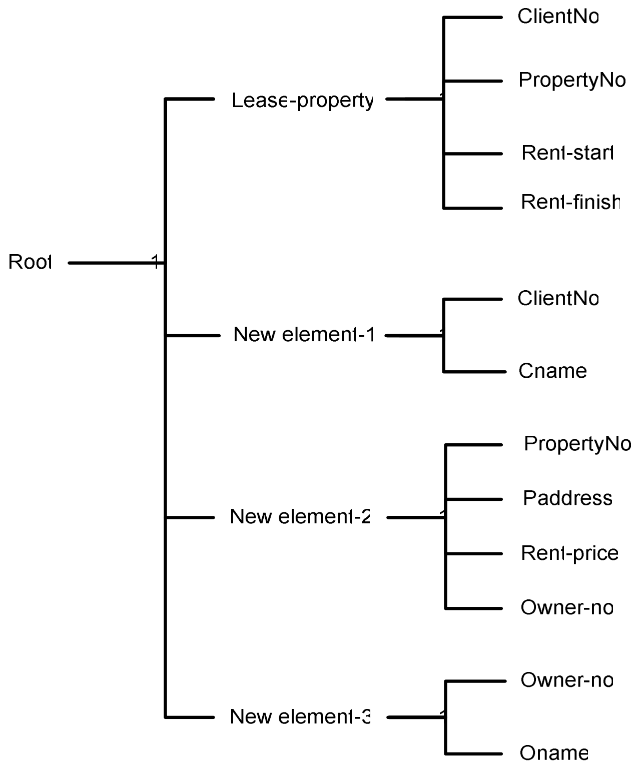


Figure 4.3: Normalized XSD schema for lease property database after applying rule-1

4.5.2 Example of Hierarchical Schema Normalization

The XSD schema in Figure 2.2 about part of school management database contains redundant data, where the information about the student's name and age is repeated every time, student takes new course and the address of room office is repeated for each office in department, because they are in the same department. Hence according to the constraint given

XFD(1) $(school/department/course/student,[./Sno,l4 \rightarrow ./Sname,l4,./age,l4])$

XFD(2) $(school/department/course,[./Cno,l3 \rightarrow ./Cname,l3])$

XFD(3) $(school/department/office,[./room\_no,l3 \rightarrow ./Oname,l3])$

XFD(4) $(school/department,[./dname,l2 \rightarrow .//office/address,l3])$

The XFD (1) is absolute dependency which causes redundancy. To remedy this problem the platform apply rule-1, create new element-1 under the root, copy (Sno), (Sname), and (age), put them under the new element-1, mark (Sno) as key in new element-1, delete (Sname), and (age) from original location. The other constraint that causes redundancy is XFD (4) that the department name determines the address of the office which is here in other level, means it is a relative transitive dependency. The address of the room is redundantly stored with each room in the same department. To remedy this redundancy, the platform copy (address), put it in the same level of the LHS of (dname) which mean under complex element department, and delete it from its original location. Figure 4.4 shows an XML document for un-normalized school schema and Figure 4.5 shows XML document for normalized school schema after applying elimination rules.

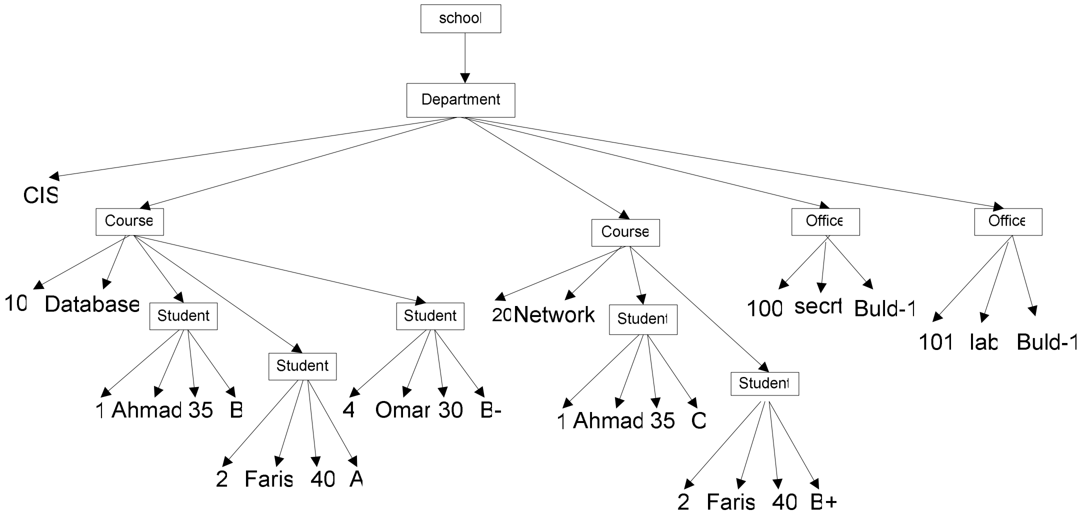


Figure 4.4: XML document for un-normalized school schema

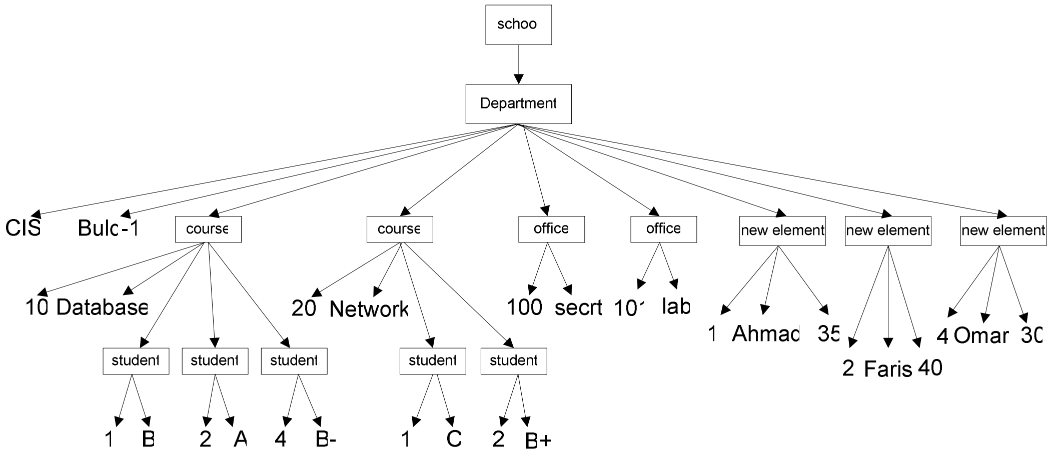


Figure 4.5: XML document for normalized school schema

Example 4.5.3: **Example of Hierarchical Schema normalization that apply rule-3**

Consider Project-supplier-part database shown in Figure 3.1. The database has two functional dependencies; the second one is representing relative partial XFD: XFD(2):

$(PSJ\,/\,project, [./\sup plier\,/\,Sname, l3, ./\,Part\,/\,PartNo, l4 \rightarrow ./\,Part\,/\,\mathrm{Pr}ice, l4])$, which states that the price depends on supplier name and part number regardless of project. This dependency is violated as clear in the instance in Figure 3.1. Supplier "ABC Trading" sells part number "P700" at price "80" to project "Garden", but sells the same part to project "Road Work" at price "10". To remedy this redundancy apply Rule-3, create new element-1 under the root, copy (Sname), (PartNo), and (Price), put (Sname) under the new elemet-1, create new element-2 under new element-1, put (PartNo) and (Price) under the new element-2, delete the (Price) from original location. Figure 4.6 shows the tree representation of XML document for normalized Project-supplier-part database. It is obvious now that supplier "ABC Trading" sells part number "P700" at the same price "80" to all projects according to specified constraint.

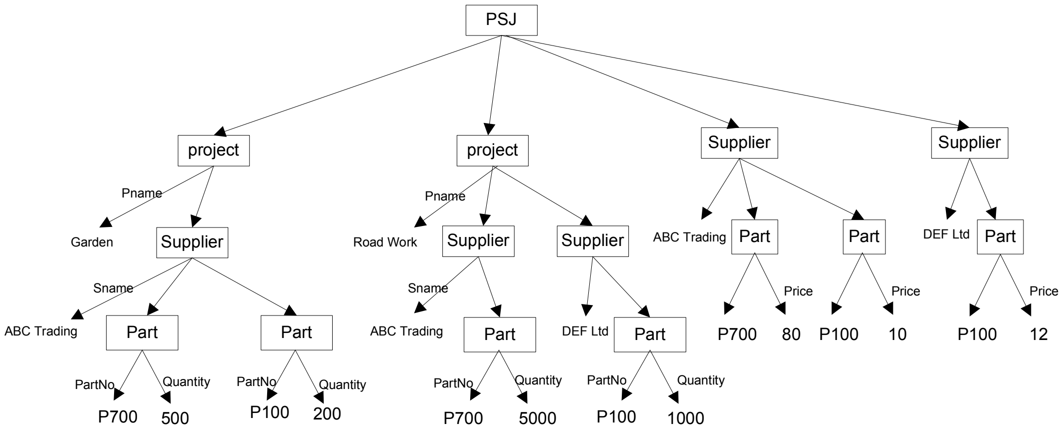


Figure 4.6 A tree representation of XML document for Project-supplier-part database after applying rule-3.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

The development of new Web application that requires efficient design and maintenance of large amounts of data makes it increasingly important to design good XML database to prevent data redundancies and update anomalies.

In this thesis, we used XSD as a schema for XML database as it is improvement over DTD and has more capability than DTD such as the ability to specify type constraints and keys constraint.

We improved the definition of XML functional dependency (XFD) according to the hierarchical structure of XSD schema through using XPath language and considering the level of the element. It is improvement upon previous proposals by defining the syntax and semantics precisely and captures a comprehensive set of XML data redundancies, more than that it considers string values and element nodes as XML schemas not only have simple data types but also nodes of complex types. The main goal of identifying XFD is to detect the possible redundancy they may cause and thus prevent this redundancy.

We propose an XML Normal Form (XNF), to determine the redundancy issue related to XFD. Our proposed normal form are generalizes to BCNF in relational database, it is also preserves the hierarchical structure for both the XSD schema and XML document, and satisfies user requirement.

Finally, we define set of normalization rules that eliminate redundancies, then design and implement the process of XML normalization through a semi-automated XML Normalizer tool. The XML Normalizer is very useful for designer to perform the XML database schemas normalization quickly and accurately. This may save the time and effort of XML database designers and thus let them focus on other phase of the XML schema design process. Furthermore, it helps avoid human error through manually normalization process of the XML schema.

We evaluate our approach through examples of XML database. The results demonstrate that the XML schema generated by XML Normalizer contribute to a normal form schema. The effect of normalization process on XSD appears clearly in a large document with a huge amount of data, while it can not be tangible in the same document with a little amount of data.

Another important issue to consider is that "if the normalization process or Normalizer gives a unique solution?", it is well known in normalization of relational schema that the decomposition process does not guarantee unique results as it depends on the order in which the dependencies are examined. However, the restructuring process does not necessarily give a unique results, it does provide an insights into the normalization process for XML schema.

5.2 Future Work

Our work on normalizing XML schema is based on some rules that detect data redundancies. The XML data redundancies have a richer semantics than redundancies in the relational context. The main problem for XML normalization is that no standard rule for normalization in XML. Although it is mature in other database model, it is not mature yet in XML database. Hence, there is much future work in this area.

1- It is worth to investigate of other types of redundancy such as those caused by multi-valued dependencies. The XML normalizer tool can then be extended to deal with various anomalies that may exist in hierarchical XML schema.

2- There is more complex situation where the redundancy is harder or impossible to recognize; hence it is desirable to investigate the problem of Normalization by taking into account the degree of relationship between elements.

3- It would be interesting to improve the XML Normalizer tool to implement the automated discovering of functional dependency and keys before normalization process.

4- It is also worth to investigate about inference rules for XML functional dependency and add it to implementation.

5- Finally, It is more interesting to evaluate the normal form on real XML dataset.

REFERENCES

Ahmad K. & Ibrahim H., (2008), 'Functional Dependencies and Inference Rules for XML', ***Information Technology, ITSim. International Symposium on***, IEEE, pp.1-6

Ahmad K. & Ibrahim H. (2009), 'Inferring Functional Dependencies for XML Storage', ***International Conference on Electrical Engineering and Informatics***, IEEE, PP 387-392.

Arenas M.& Libkin L. (2003), 'An Information-Theoretic Approach to Normal Form for Relational and XML Data', ***ACM, PODS***, PP.15-26.

Arenas M.& Libkin L. (2004), 'A Normal Form for XML Documents', ***ACM, Transactions on Database Systems***, Vol. 29, No 1, PP 195-232.

Arenas M.,(2005). *'Design principles for XML Data*', (PHD Dissertation), University of Toronto.

Arenas M. (2006), 'Normalization Theory for XML', ***SIGMOD Record***, Vol.35, No. 4, PP.57-64.

Bray T., Paoli J., Sperberg-McQueen C., & Maler E. (2000), ' ***Extensible Markup Language (XML) 1.0 (Second Edition)***', (On-Line), Available at:
http://www.w3.org/TR/2000/REC-xml-20001006,
Accessed in: October 2011

Buneman P., Davidson S., Fan W., Hara C., and Tan W., (2001), 'Keys for XML', WWW10, Hong Kong, ***ACM***.

Chen H. & Liao H. (2010), 'An Overview of Functional Dependencies in XML', ***International Conference on Education Technology (ICEIT), IEEE International Conference*** , PP 174-178,.

Clark J. & DeRose S., (1999), ***'XML Path Language (XPath)Version 1.0'***, W3C Recommendation 16 November 1999, (On-Line), Available at:
http://www.w3.org/TR/xpath/
Accessed in: October 2011

Connolly T., & Begg C., (2010), '***Database Systems A Practical Approach to Design, Implementation, and Management***', 5th Ed., Pearson Education, Inc., PP. 365-405.

Date C., (2004), ***'An Introduction to Database Systems'***, 8th Ed., Pearson Education, Inc.,pp.333-402.

Deitel P.J. & Deitel H.M. (2005), ***'XML and RSS'***, Internet & World Wide Web How to Program', 4th Ed., Pearson Education,Inc., New Jersey, PP. 508-532.

Elmasri R. & Navathe Sh. (2007), ***'Fundamentals of Database Systems'***, 5th Ed., Pearson Addison Wesley, PP 325-367.

Lee M., Ling T, & Low W.,(2002), 'Designing Functional Dependencies for XML', ***Springer, Advances in Database Technology, Lecture Notes in Computer Science*** **LNCS**, Vol. 2287, PP.124-141.

Libkin L., (2007), 'Normalization Theory for XML', ***Springer-Verlag Berlin Heidelberg, LNCS***, pp. 1-13.

Lv T., & Yan P.(2007), 'XML Normal Forms Based on Constraint-Tree-Based Functional Dependencies'***, Springer, Lecture Notes in Computer Science LNCS***, Vol. 4537, PP.348-357.

Lv T.,& Yan P. (2008),'Removing XML Data Redundancies by Constrain-tree-based Functional Dependencies', ***IEEE, ISECS International Colloquium on Computing, Communication, Control, and Management***, PP.595-599.

Michel F., (2007), ***'Representation of XML Schema Components'***, Master Thesis, University of California, Berkeley

Necasky M. & Pokorny J (2007), 'Extending E-R for Modelling XML Keys', ***IEEE***, PP 236-241.

Pankowski T., & Pilka T. (2008), 'Dealing with Redundancies and Dependencies in Normalization of XML data', ***IEEE, Proceedings of the International Multiconference on Computer Science and Information Technology***, PP.543-550.

Pankowski T. & Pilka T. (2009), 'Transformation of XML Data into Normal Form', ***Informatica 33***, PP.417-430.

Provost W., (2002), 'Normalizing XML, Part1', (On-Line), Available at:
http://www.xml.com/pub/a/2002/11/13/normalizing.html
Accessed in 6/29/2011.

Provost W., (2002), 'Normalizing XML, Part2', (On-Line), Available at:
http://www.xml.com/pub/a/2002/12/04/normalizing.html
Accessed in 6/29/2011.

Shahriar S. & Liu J.(2008), 'On Defining Keys for XML', ***Workshops.70, IEEE***, PP 86-91.

Shipman J., (2009), ***'Constructing a Document Type Definition (DTD) for XML'***, new mexico technology, (On-Line), Available at:
http://infohost.nmt.edu/tcc/help/pubs/dtd/dtd.pdf
Accessed in: August 2011

Tan Z., Xu J., Wang W., & Shi B. (2005), 'Strong Normalized XML Documents in Normalized Relations', ***IEEE, The Fifth International Conference on Computer and Information Technology***, PP.123-129.

Thompson H., Beech D., Maloney M., & Mendelsohn N. (2004), ***'XML Schema. W3C Recommendation'***, (On-Line), Available at:
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ ,
Accessed in: August 2011

Vincent M., Liu J., & Liu C. (2004), 'Strong Functional Dependencies and Their Application to Normal Forms in XML', ***ACM, Transactions on Database Systems***, Vol. 29, No. 3, PP.445–462.

Wu X., Ling T., Lee M., & Dobbie G.,(2001), 'Designing Semistructured Databases Using ORA-SS Model', ***Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE), IEEE***, Computer Society.

Wu X., Ling T, Lee S., Lee M., & Dobbie G. (2002), 'NF-SS:A Normal Form for Semistructured Schema', ***Springer, LNCS 2465***, PP.292-305.

Wu Y, (2004), 'Normalization Design of XML Database Schema for Eliminating Redundant Schema and Satisfying Lossless Join', ***Proceedings of the IEEE/ ACM International Conference on Web Intelligence***, PP 660-663.

Xia L., Fei-yue Y., Hong-juan Y., & Wen-tao P. (2006), ' Functional Dependency Maintenance and Lossless Join Decomposition in XML Model Decomposition', ***IEEE, Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKG '06***), PP 77-80 .

Yan P. & Lv T. (2006), 'Functional Dependencies in XML Documents', ***APWeb Workshops, LNCS 3842, Springer- Verlag Berlin Heidelberg***, PP 29-37

Yu C., & Jagadish H., (2008), 'XML schema refinement through redundancy detection and normalization', ***VLDB Journal 'The International Journal on Very Large Data Bases'***, Vl. 17, no. 2, Springer, pp 203-223.

Zainol Z., & Wang B.,(2010), 'XML Document Design via GN-DTD', ***European Journal of Scientific Research,*** Vol.44, No.2, pp.314-336.

Zhang Y. (2004). ***Multivalued Dependencies and a Normal Form for XML,*** Master thesis, University of Toronto

Zhao X., Xin J., & Zhang E. (2009), 'XML functional Dependency and Schema Normalization', ***IEEE, Hybrid Intelligent Systems, Ninth International Conference*** , PP.307-312.

APPENDICES

Appendix A: XML Normalizer

In this section we show how to follow the bottom-up approach for using normalization technique in which the XML database is designed based on the information taken from the data source, then specify the keys and functional dependencies by the designer to apply the normalization rules to get a good XML database with an accurate representation of data that has suitable set of relations.

Our tool is semi-automated the process of XML schema normalization. XML Normalizer is a case tool that has a Graphical User Interface (GUI) which is implemented by using Visual Basic under visual studio 2008, it is very simple and easy to use. The following are steps of using the tool after launching the application:

- The user should first press Browse Button
- The user selects the schema code file and presses the Read Schema button.
- The platform will analyze and parse the schema code to show hierarchical tree of entered schema; a list of complex elements name, a list of simple elements name and their levels respectively.
- The user selects the keys from the list of simple element in GUI and press the Insert Key button.
- Entering the header path.
- Select the left hand side elements then press Add to LHS button.
- Select the right hand side elements then press Add to RHS button.
- Finally, press the Normalization button,

Figure 1 shows the GUI of XML Normalizer. The main function for each button is described below:

Browse Button: Used to load the file that contains the XSD code which saved previously.

Read Schema Button: Used to analyze and parse the schema code then display the tree structure of loaded schema, a list of complex elements with their levels, a list of simple elements with their levels.

Insert Key button: used to display the list of keys selected from the list of simple elements.

Add to LHS Button: used to display left hand side elements of functional dependency selected from list of complex elements or list of simple elements.

Add to RHS Button: Select right hand side elements from list of simple elements, then display the functional dependency as: header path, LHS → RHS

Normalization Button: The platform first checks the type of XML representation which is either flat or hierarchical representation, then a specific algorithm implemented to perform the normalization process and finally display the tree of normalized schema

When the platform determine flat schema, it starts check the functional dependency, if the functional dependency is partial XFD or transitive XFD or one of the LHS is not key element then it calls Rule-1 procedure which is about EAXFD process. EAXFD procedure starts to create a new complex element under the root, copy the LHS and the RHS of the XFD and make them as children of the new complex element, make the LHS element as a key for the new complex element, and delete the RHS elements from the original location. The checking process of XFD is repeated for all the entered XFD to remedy the redundancy caused by anomalous dependency.

Otherwise when the platform determines hierarchical schema, it starts checking the functional dependency if both sides are in the same level and under the same complex element node in the bottom of the tree, then call Rule-1 procedure about EAXFD process. Otherwise if the elements of XFD are in different levels then checks if the functional dependency, is Relative Transitive XFD then call Rule-2 procedure s about ERTXFD/H process, or is Relative Partial XFD then call Rule-3 procedure about ERPXFD/H process.

If the LHS contains one element key or non key or complex element then call Rule-2 procedure.

New Button: used to begin a new GUI for XML normalizer.

Exit Button: To quit form the application.

Figure 1 Shows GUI for XML normalizer
Figure 2 Shows GUI for Lease Property database normalization process presented in Example 4.5.1.
Figure 3 Shows GUI for School management database normalization process presented in Example 4.5.2.
Figure 4 Shows GUI for Project-supplier-part database normalization process presented in Example 4.5.3 that use rule-3 in normalization process.

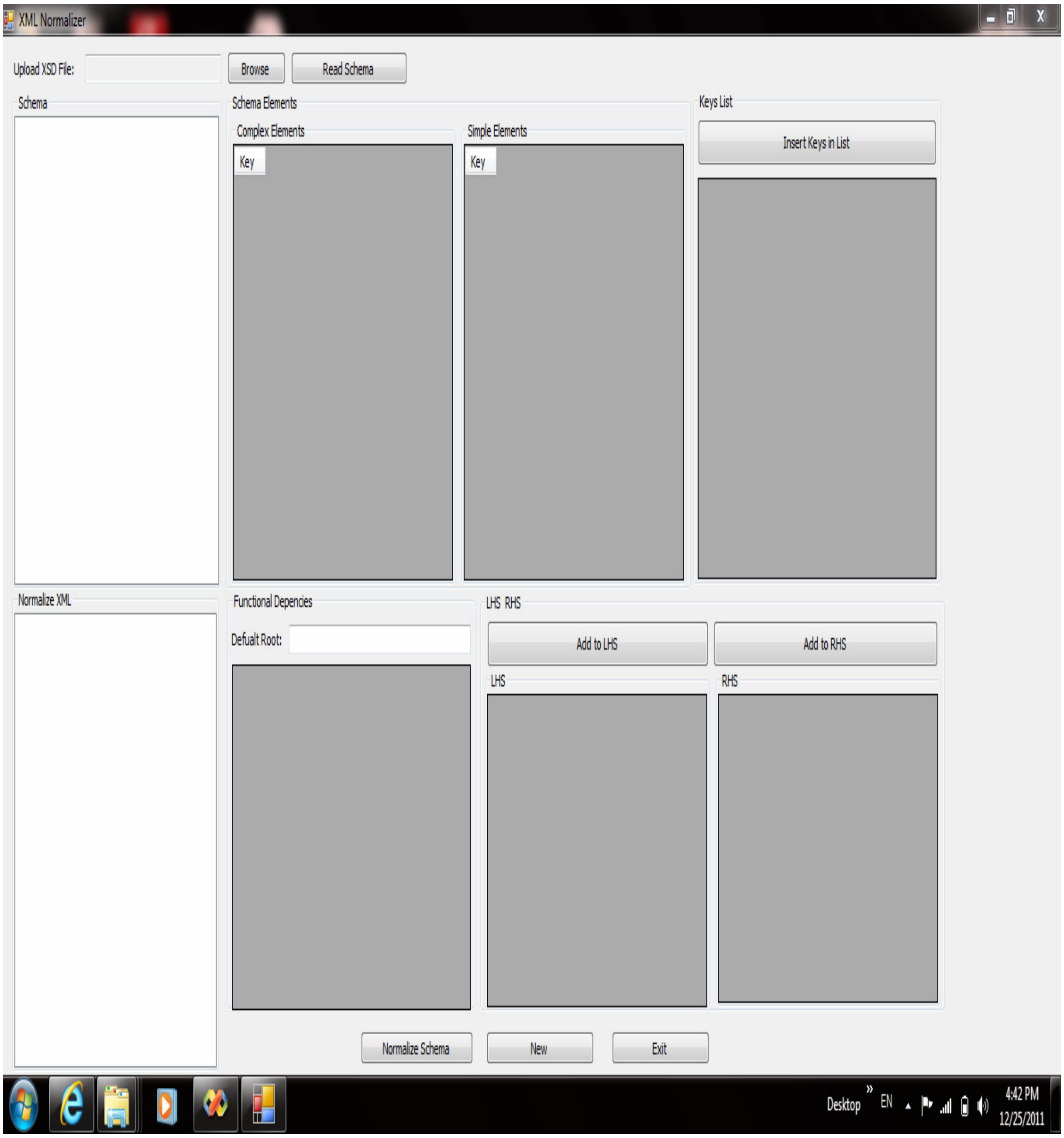
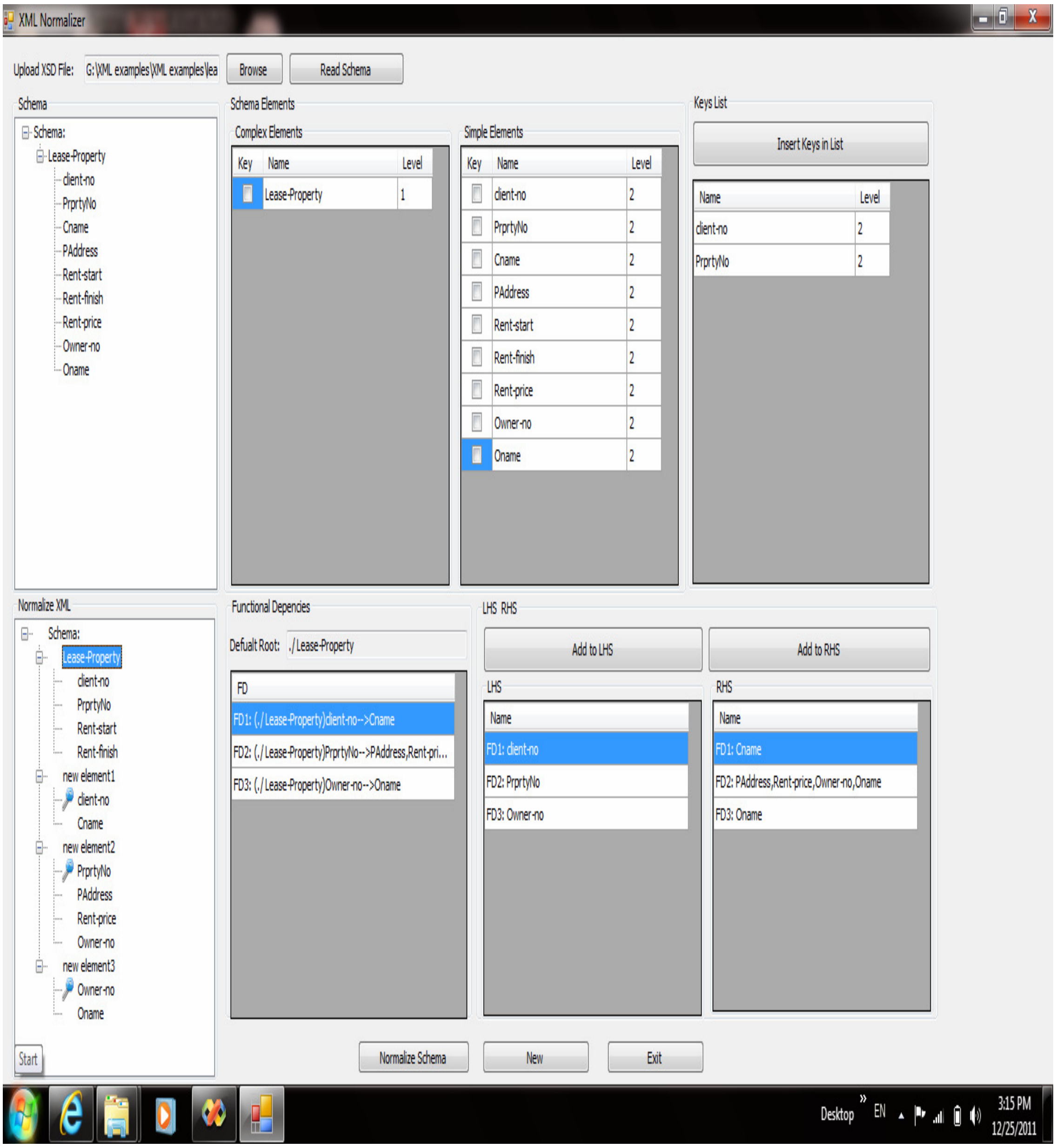


Figure 1: GUI for XML normalizer



Upload XSD File: G:\XML examples\XML examples\lea

Schema

- Schema:
  - Lease-Property
    - client-no
    - PrprtyNo
    - Cname
    - PAddress
    - Rent-start
    - Rent-finish
    - Rent-price
    - Owner-no
    - Oname

Schema Elements

Complex Elements

Key	Name	Level
□	Lease-Property	1

Simple Elements

Key	Name	Level
□	client-no	2
□	PrprtyNo	2
□	Cname	2
□	PAddress	2
□	Rent-start	2
□	Rent-finish	2
□	Rent-price	2
□	Owner-no	2
□	Oname	2

Keys List

Name	Level
client-no	2
PrprtyNo	2

Normalize XML

- Schema:
  - Lease-Property
    - client-no
    - PrprtyNo
    - Rent-start
    - Rent-finish
  - new element1
    - client-no
    - Cname
  - new element2
    - PrprtyNo
    - PAddress
    - Rent-price
    - Owner-no
  - new element3
    - Owner-no
    - Oname

Functional Depencies

Defult Root: ./Lease-Property

FD
FD1: (./Lease-Property)client-no-->Cname
FD2: (./Lease-Property)PrprtyNo-->PAddress,Rent-pri...
FD3: (./Lease-Property)Owner-no-->Oname

LHS RHS

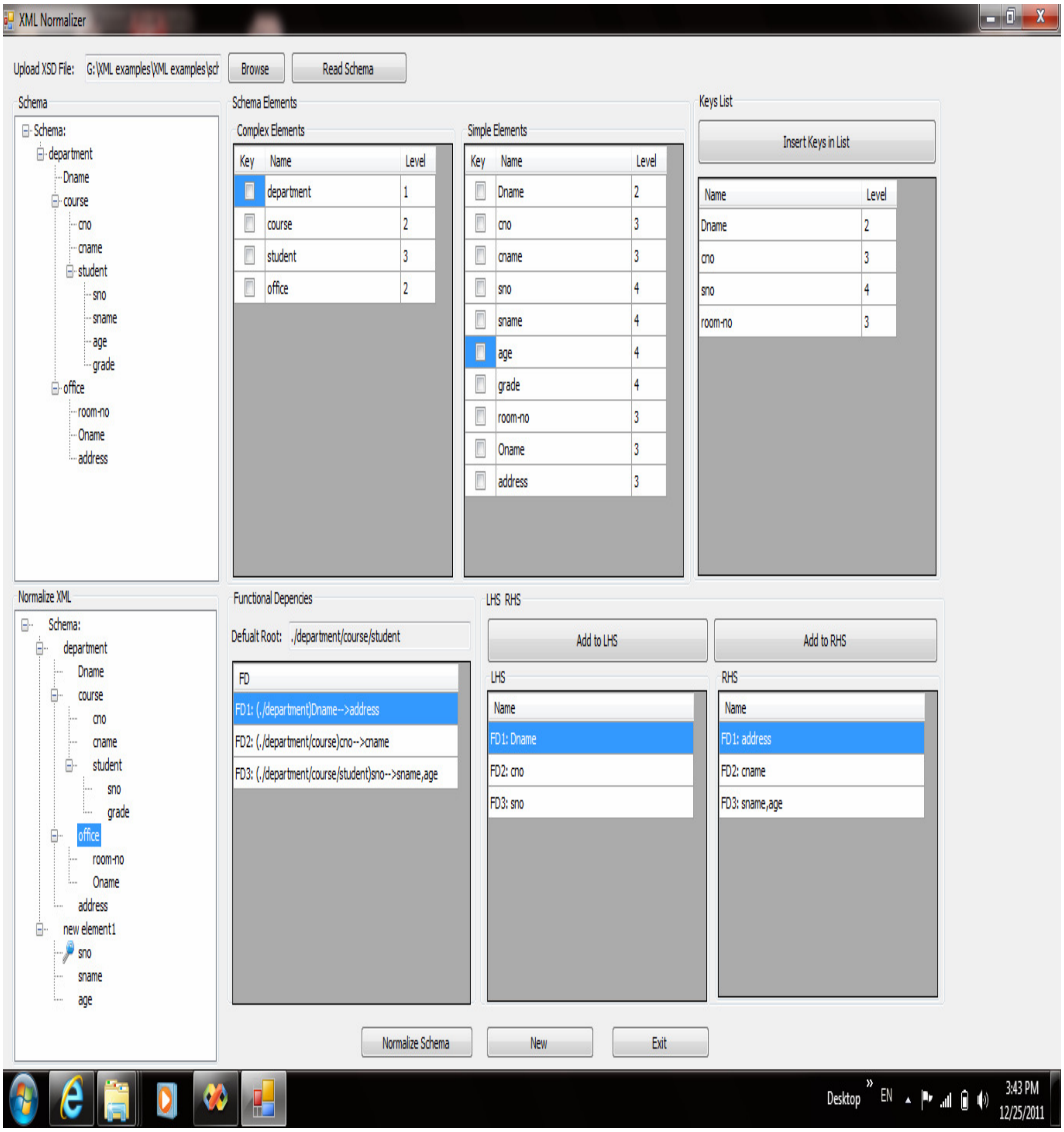
LHS

Name
FD1: client-no
FD2: PrprtyNo
FD3: Owner-no

RHS

Name
FD1: Cname
FD2: PAddress,Rent-price,Owner-no,Oname
FD3: Oname

Figure 2: GUI for Lease Property database normalization process presented in Example 4.5.1



Upload XSD File: G:\XML examples\XML examples\sch

```
Schema:
  department
    Dname
    course
      cno
      cname
      student
        sno
        sname
        age
        grade
    office
      room-no
      Oname
      address
```

Key	Name	Level
□	department	1
□	course	2
□	student	3
□	office	2

Key	Name	Level
□	Dname	2
□	cno	3
□	cname	3
□	sno	4
□	sname	4
□	age	4
□	grade	4
□	room-no	3
□	Oname	3
□	address	3

Name	Level
Dname	2
cno	3
sno	4
room-no	3

```
Schema:
  department
    Dname
    course
      cno
      cname
      student
        sno
        grade
    office
      room-no
      Oname
    address
  new element1
    sno
    sname
    age
```

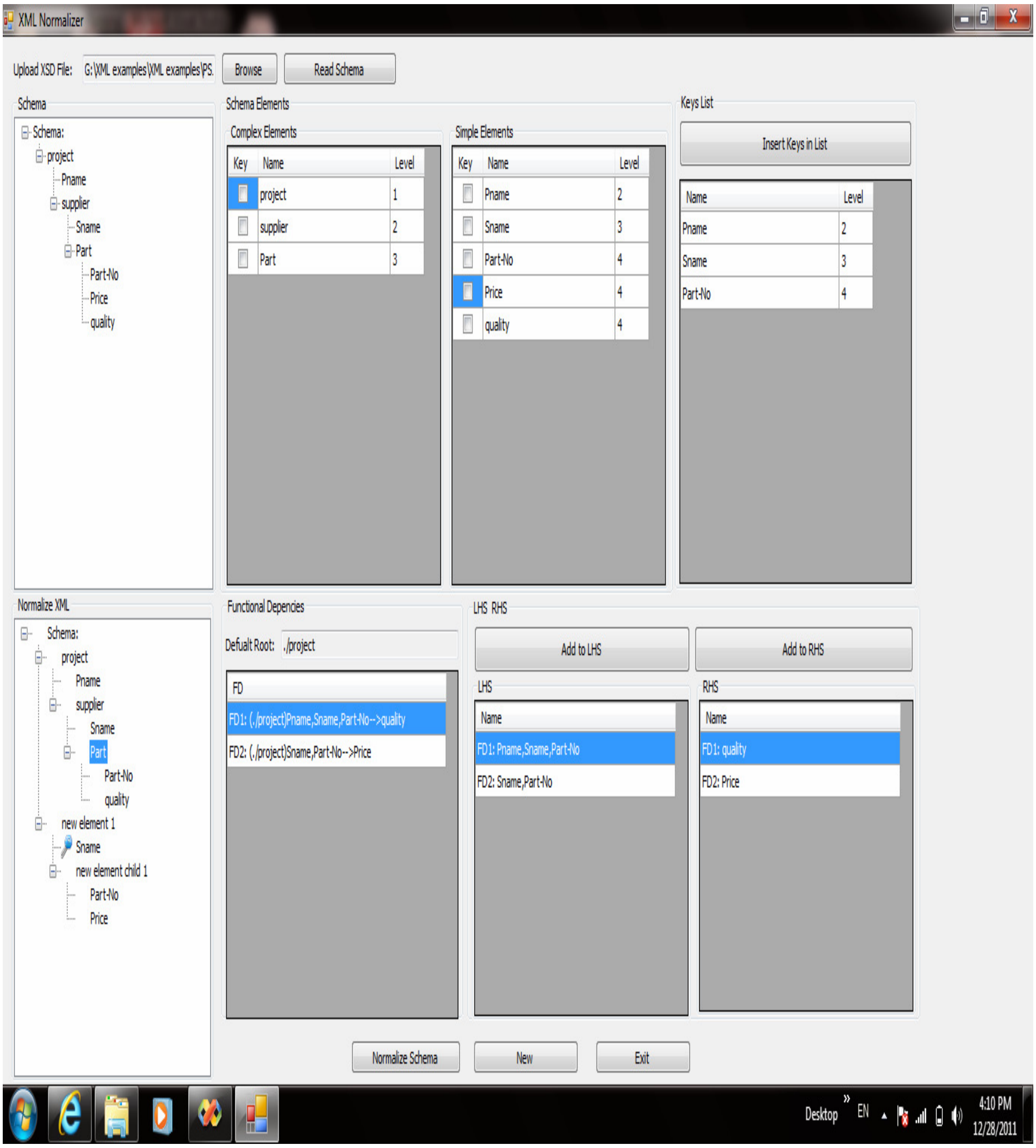
Defualt Root: ./department/course/student

FD
FD1: (./department)Dname-->address
FD2: (./department/course)cno-->cname
FD3: (./department/course/student)sno-->sname,age

Name
FD1: Dname
FD2: cno
FD3: sno

Name
FD1: address
FD2: cname
FD3: sname,age

Figure 3: GUI for School management database normalization process presented in Example 4.5.2



Upload XSD File: G:\XML examples\XML examples\PS.

Schema

```
Schema:
  project
    Pname
    supplier
      Sname
      Part
        Part-No
        Price
        quality
```

Schema Elements

Complex Elements

Key	Name	Level
☑	project	1
□	supplier	2
□	Part	3

Simple Elements

Key	Name	Level
□	Pname	2
□	Sname	3
□	Part-No	4
☑	Price	4
□	quality	4

Keys List

Name	Level
Pname	2
Sname	3
Part-No	4

Normalize XML

```
Schema:
  project
    Pname
    supplier
      Sname
      Part
        Part-No
        quality
  new element 1
    Sname
    new element child 1
      Part-No
      Price
```

Functional Depencies

Defualt Root: ./project

FD
FD1: (./project)Pname,Sname,Part-No-->quality
FD2: (./project)Sname,Part-No-->Price

LHS RHS

LHS

Name
FD1: Pname,Sname,Part-No
FD2: Sname,Part-No

RHS

Name
FD1: quality
FD2: Price

Figure 4: GUI for Project-supplier-part database normalization process presented in Example 4.5.3

Appendix B: Implementation of normalizing flat XML representation

```
PublicSub flatXML()
Dim strComplexElementRoot AsString = gridComplexElement.Item("Name",
0).Value
Dim strCurrentLHS AsString = ""
Dim strCurrentRHS AsString = ""
For i AsInteger = 0 To gridLHS.Rows.Count - 1
            strCurrentLHS = gridLHS.Item(0, i).Value
            strCurrentRHS = gridRHS.Item(0, i).Value
            strCurrentLHS = strCurrentLHS.Substring(5)
            strCurrentRHS = strCurrentRHS.Substring(5)
Dim strLHSArray AsString() = strCurrentLHS.Split(",")
Dim strRHSArray AsString() = strCurrentRHS.Split(",")
// Remedy Transitive & Partial Dependency
If strLHSArray.Length < gridKeysList.Rows.Count Then
                TreeView1.Nodes(0).Nodes.Add("new element"& i + 1,
"new element"& i + 1)
For j AsInteger = 0 To strLHSArray.Length - 1
Dim addedNode AsNew TreeNode
                    addedNode.Text = strLHSArray(j)
                    addedNode.Name = strLHSArray(j)
                    addedNode.ImageIndex = 1
                    TreeView1.Nodes(0).Nodes("new element"& i +
1).Nodes.Add(addedNode)
' TreeView1.Nodes(0).Nodes("new element" & i +
1).Nodes(strLHSArray(j)).ImageIndex = 0
Next
For l AsInteger = 0 To strRHSArray.Length - 1
Dim selectedNode As TreeNode = GetNode(strRHSArray(l),
TreeView1.Nodes)
                    TreeView1.Nodes.Remove(selectedNode)
                    TreeView1.Nodes(0).Nodes("new element"& i +
1).Nodes.Add(strRHSArray(l))
Next
ElseIf strLHSArray.Length = gridKeysList.Rows.Count Then
// If key not subset or equal the elemet of LHS
For x AsInteger = 0 To strLHSArray.Length - 1
If strLHSArray(x) <> gridKeysList.Item("Name", x).Value Then
                        TreeView1.Nodes(0).Nodes.Add("new element"& i
+ 1, "new element"& i + 1)
For j AsInteger = 0 To strLHSArray.Length - 1
Dim addedNode AsNew TreeNode
  addedNode.Text = strLHSArray(j)
  addedNode.Name = strLHSArray(j)
  addedNode.ImageIndex = 1
  TreeView1.Nodes(0).Nodes("new element"& i + 1).Nodes.Add(addedNode)
Next
For l AsInteger = 0 To strRHSArray.Length - 1
Dim selectedNode As TreeNode=GetNode(strRHSArray(l), TreeView1.Nodes)
     TreeView1.Nodes.Remove(selectedNode)
     TreeView1.Nodes(0).Nodes("new element"& i +
1).Nodes.Add(strRHSArray(l))
Next
ExitFor
EndIf
Next
EndIf
Next
EndSub
```

Appendix C: Implementation of normalizing hierarchical XML representation

```
PublicSub hierarchicalXML()

// checks the level of both sides of XFD
For j AsInteger = 0 To strLHSArrayLevel.Length - 1
If j = 0 Then
                    strLevelValue = strLHSArrayLevel(j)
Else
If strLevelValue <> strLHSArrayLevel(j) Then
                        boolLevelsEquals = False
ExitFor
EndIf
EndIf
Next

If boolLevelsEquals And strRHSArrayLevel.Length > 0 Then
For j AsInteger = 0 To strRHSArrayLevel.Length - 1
If strLevelValue <> strRHSArrayLevel(j) Then
                        boolLevelsEquals = False
ExitFor
EndIf
Next
EndIf
// To remedy the absolut dependency
If boolLevelsEquals Then'Check if all levels in RHS as LHS are equals
If strLHSArrayLevel(0) <> 2 Then
If strLHSArrayLevel(0) <> 3 Then
                        TreeView1.Nodes(0).Nodes.Add("new element"&
intNewRootCounter, "new element"& intNewRootCounter)

For j AsInteger = 0 To strLHSArrayNames.Length - 1
Dim addedNode AsNew TreeNode
                            addedNode.Text = strLHSArrayNames(j)
                            addedNode.Name = strLHSArrayNames(j)
                            addedNode.ImageIndex = 1
                            TreeView1.Nodes(0).Nodes("new element"&
intNewRootCounter).Nodes.Add(addedNode)
Next
For l AsInteger = 0 To strRHSArrayNames.Length - 1
Dim selectedNode As TreeNode = GetNode(strRHSArrayNames(l),
TreeView1.Nodes)
                            TreeView1.Nodes.Remove(selectedNode)
                            TreeView1.Nodes(0).Nodes("new element"&
intNewRootCounter).Nodes.Add(strRHSArrayNames(l))
Next

                        intNewRootCounter += 1
EndIf
EndIf
ElseIf strLHSArrayLevel.Length < gridKeysList.Rows.Count Then
// Remedy the Relative transative dependency
If strLHSArrayNames.Length = 1 Then
Dim strNodeID AsString = ""
Dim strNodeName AsString = strLHSArrayNames(0)
Dim strNodeLevel AsString = strLHSArrayLevel(0)
Dim strNodeParenID AsString = ""
Dim strNodeParentName AsString = ""

Dim strTypeOfLHS AsString = ""
```

```
For a AsInteger = 0 To gridComplexElement.Rows.Count - 1
If gridComplexElement.Item("Name", a).Value = strNodeName Then
                            strTypeOfLHS = "Complex"
ExitFor
EndIf
Next
If strTypeOfLHS = ""Then
                        strTypeOfLHS = "Simple"
EndIf
// if LHS is complex element
If strTypeOfLHS = "Complex"Then
For b AsInteger = 0 To strRHSArrayNames.Length - 1
Dim nodeDeletedNode As TreeNode = GetNode(strRHSArrayNames(b),
TreeView1.Nodes)
                            TreeView1.Nodes.Remove(nodeDeletedNode)

Dim nodeComplexNode As TreeNode = GetNode(strNodeName,
TreeView1.Nodes)

nodeComplexNode.Nodes.Add(strRHSArrayNames(b))
Next
ElseIf strTypeOfLHS = "Simple"Then

For c AsInteger = 0 To gridSimpleElement.Rows.Count - 1
If gridSimpleElement.Item("Name", c).Value = strNodeName Then
                                strNodeID =
gridSimpleElement.Item("NodeID", c).Value
                                strNodeParenID =
gridSimpleElement.Item("ParentID", c).Value
EndIf
Next

For d AsInteger = 0 To gridComplexElement.Rows.Count - 1
If gridComplexElement.Item("NodeID", d).Value = strNodeParenID Then
                                strNodeParentName =
gridComplexElement.Item("Name", d).Value
EndIf
Next

For d AsInteger = 0 To gridSimpleElement.Rows.Count - 1
If gridSimpleElement.Item("NodeID", d).Value = strNodeParenID Then
                                strNodeParentName =
gridSimpleElement.Item("Name", d).Value
EndIf
Next

If strNodeParenID = 1 Then
                            strNodeParentName = "Schema:"
EndIf

If strNodeParentName <>""Then
Dim nodeParenLHSNode As TreeNode = GetNode(strNodeParentName,
TreeView1.Nodes)

For e AsInteger = 0 To strRHSArrayNames.Length - 1
Dim nodeDeletedRHSNode As TreeNode = GetNode(strRHSArrayNames(e),
TreeView1.Nodes)

TreeView1.Nodes.Remove(nodeDeletedRHSNode)

nodeParenLHSNode.Nodes.Add(strRHSArrayNames(e))
```

```
Next
EndIf

Dim boolLHSElementIsKey AsBoolean = False
For f AsInteger = 0 To gridKeysList.Rows.Count - 1
If strNodeName = gridKeysList.Item("Name", f).Value Then
                                boolLHSElementIsKey = True
EndIf
Next
```

// if LHS is not key

```
IfNot boolLHSElementIsKey Then
Dim boolNodeExistInKeyToBeAdded AsBoolean = False
For g AsInteger = 0 To listKeyToBeAddedNodeID.Count - 1
If listKeyToBeAddedNodeID.Item(g) = strNodeID Then
                                    boolNodeExistInKeyToBeAdded =
True
EndIf
Next

IfNot boolNodeExistInKeyToBeAdded Then
                     listKeyToBeAddedNodeID.Add(strNodeID)
                     listKeyToBeAddedParentNodeID.Add(strNodeParenID)
                     listKeyToBeAddedName.Add(strNodeName)
                     listKeyToBeAddedLevel.Add(strNodeLevel)
EndIf
EndIf
EndIf
```

// **Remedy the relative partial dependency**

```
ElseIf strLHSArrayLevel.Length = 2 Then
Dim strNodeID AsString = ""
Dim strNodeName AsString = strLHSArrayNames(0)
Dim strNodeLevel AsString = strLHSArrayLevel(0)
Dim strNodeParenID AsString = ""
Dim strNodeParentName AsString = ""
Dim nodeParenNode As TreeNode = GetNode("Schema:", TreeView1.Nodes)
Dim addedNode AsNew TreeNode
                    addedNode.Text = "new element "& i
                    addedNode.Name = "new element "& i
                    nodeParenNode.Nodes.Add(addedNode)
Dim nodeParenNode1 As TreeNode = GetNode("new element "& i,
TreeView1.Nodes)
Dim addedNode1 AsNew TreeNode
                    addedNode1.Text = strLHSArrayNames(0)
                    addedNode1.Name = strLHSArrayNames(0)
                    addedNode1.ImageIndex = 1
                    nodeParenNode1.Nodes.Add(addedNode1)
Dim nodeParenNode2 As TreeNode = GetNode("new element "& i,
TreeView1.Nodes)
Dim addedNode2 AsNew TreeNode
                    addedNode2.Text = "new element child "& i
                    addedNode2.Name = "new element child "& i
                    nodeParenNode2.Nodes.Add(addedNode2)

Dim nodeParenNode3 As TreeNode = GetNode("new element child "& i,
TreeView1.Nodes)
Dim addedNode3 AsNew TreeNode
                    addedNode3.Text = strLHSArrayNames(1)
                    addedNode3.Name = strLHSArrayNames(1)
                    nodeParenNode3.Nodes.Add(addedNode3)
For e AsInteger = 0 To strRHSArrayNames.Length - 1
```

```
Dim nodeDeletedRHSNode As TreeNode = GetNode(strRHSArrayNames(e),
TreeView1.Nodes)
                    TreeView1.Nodes.Remove(nodeDeletedRHSNode)

                    nodeParenNode3.Nodes.Add(strRHSArrayNames(e))
Next

Dim boolLHSElementIsKey AsBoolean = False
For f AsInteger = 0 To gridKeysList.Rows.Count - 1
If strNodeName = gridKeysList.Item("Name", f).Value Then
                            boolLHSElementIsKey = True
EndIf
Next

IfNot boolLHSElementIsKey Then
Dim boolNodeExistInKeyToBeAdded AsBoolean = False

For g AsInteger = 0 To listKeyToBeAddedNodeID.Count - 1
If listKeyToBeAddedNodeID.Item(g) = strNodeID Then
                                boolNodeExistInKeyToBeAdded = True
EndIf
Next

For c AsInteger = 0 To gridSimpleElement.Rows.Count - 1
If gridSimpleElement.Item("Name", c).Value = strNodeName Then
                                strNodeID =
gridSimpleElement.Item("NodeID", c).Value
                                strNodeParenID =
gridSimpleElement.Item("ParentID", c).Value
EndIf
Next

For c AsInteger = 0 To gridComplexElement.Rows.Count - 1
If gridComplexElement.Item("Name", c).Value = strNodeName Then
                                strNodeID =
gridComplexElement.Item("NodeID", c).Value
                                strNodeParenID =
gridComplexElement.Item("ParentID", c).Value
EndIf
Next

For d AsInteger = 0 To gridComplexElement.Rows.Count - 1
If gridComplexElement.Item("NodeID", d).Value = strNodeParenID Then
                                strNodeParentName =
gridComplexElement.Item("Name", d).Value
EndIf
Next

For d AsInteger = 0 To gridSimpleElement.Rows.Count - 1
If gridSimpleElement.Item("NodeID", d).Value = strNodeParenID Then
                                strNodeParentName =
gridSimpleElement.Item("Name", d).Value
EndIf
Next
IfNot boolNodeExistInKeyToBeAdded Then
                       listKeyToBeAddedNodeID.Add(strNodeID)
listKeyToBeAddedParentNodeID.Add(strNodeParenID)
                       listKeyToBeAddedName.Add(strNodeName)
                       listKeyToBeAddedLevel.Add(strNodeLevel)
EndIf
EndIf
```