UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A FLEXIBLE SCHEMA-AWARE MAPPING OF XML DATA INTO RELATIONAL MODELS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

VAMSHI KRISHNA SUNCHU
Norman, Oklahoma
2016

A FLEXIBLE SCHEMA-AWARE MAPPING OF
XML DATA INTO RELATIONAL MODELS

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Chris Weaver, Chair

Dr. Christan Grant

Dr. K. Thulasiraman

DEDICATION

to

My parents

Sunchu. Narahari, and

Sunchu. Renuka

For

Encouraging me to follow my dreams

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor, Professor Chris Weaver, whose constant guidance helped me in completing this study, from the thesis proposal up to the final thesis manuscript. Without his guidance, endless advice, and persistent help, this study would not have been possible.

A special thanks to Dr. Samuel J. Huskey, Dr. Abbas June and all the team members of Digital Latin Library for all the advice and encouragement throughout the project.

I would like to express my deep appreciation to Dr. Christan Grant and Dr. K. Thulasiraman for serving on my committee and taking time to talk with me on many occasions.

I would also like to extend my thanks to my close friends and my roomies in the university for all the memories and inside jokes that we shared together and making the last two years more memorable.

No acknowledgement would be complete without thanking my beloved father and mother, they are my inspiration and motivation for everything. I would like to thank them for letting me be the person I am today, without their endless support, enduring love, constant guidance, and encouragement, I could not have made it. Furthermore, I would like to thank my sisters and cousins for their unconditional love and support.

Contents

1 Introduction **1**

2 Background Information **7**
2.1 Text Encoding Initiative 7
2.2 Extensible Markup Language 8
2.3 Constraints on XML Structure 9
2.4 XML Schema Definition 10
2.4.1 XSD Constructs 10
2.5 Relational Database Management Systems 13

3 Overview of Mapping Techniques **17**
3.1 Schema Oblivious Techniques 17
3.1.1 Edge Mapping 17
3.1.2 DTD- Independent Schema Mapping 19
3.2 Schema Aware Techniques 20
3.2.1 Shared Inlining 20
3.2.2 Constraints-Preserving Inlining 21
3.2.3 ShreX 23

4 Design and Implementation **25**
4.1 Suggest Schema 25
4.1.1 Excerpt Table Names and Columns 26
4.1.2 Extract Keys 26
4.1.3 Introduce Keys 27
4.1.4 Handling Recurring Elements 27
4.1.5 Generate XML Paths 28
4.2 Alter Schema 30
4.2.1 Split Tables 31
4.2.2 Merge Tables 32
4.2.3 Delete Table Columns 33
4.2.4 Delete a Table 35
4.2.5 Rename Tables 36
4.2.6 Reset Schema 37

4.3 Populate Tables 37
4.4 The user interface 39
4.4.1 Action Combo Box 40
4.4.2 Table Combo Boxes 40
4.4.3 Table Models 40
4.4.4 Text Editor 41

5 System Analysis using TEI **42**

6 Conclusion **47**

List of Figures

2.1 XML Schema code, defining attribute tags. 11
2.2 XML Schema code, defining element tags. 11
2.3 XML schema code, defining complex type element. 12
2.4 Occurrences of <key> and <keyref> elements in an XSD. 13
2.5 Example XML Schema Definition, for XML files containing customer order information. 16

3.1 Sample XML code, defining a Customer data. 18
3.2 DTD document, defining Customer data. 21
3.3 Shared Inlining Mapping of Sample Data. 21

4.1 Suggested Relational Schema for the example XSD in figure 2.5. . 29
4.2 Split tables operation on the "Customer" table. 31
4.3 "Customer" and "Customer-1" tables after Split Tables action. . . 32
4.4 Merge tables operation on tables "FullAddress" and "Phone-Home". 32
4.5 Updated table list and new table attributes after Merge Tables operation. 33
4.6 The columns of the ShipInfo table selected for deletion. 34
4.7 The "ShipInfo" table after Deleting columns. 34
4.8 The "Customer-1" table selected for deletion. 35
4.9 The "Customer-1" table removed from the new schema table list. 35
4.10 The "FullAddress-Phone-Home" table is being renamed as "CustomerAddress". 36
4.11 The new "CustomerInfo" table attributes after rename operation. 37
4.12 The new table list after performing reset operation. 38
4.13 "Customer" table with values. 38
4.14 Sample relational schema after performing a series of operations. 39
4.15 The user interface layout. 41

5.1 An XML Schema Definition file defining a TEI structure. 43
5.2 Sample TEI XSD code with recurring simple element. 45
5.3 Suggested relational schema of the TEI XSD shown in 5.1. 46

List of Tables

3.1 Edge Schema representation of the Customer Data in figure 3.1. . 19
3.2 Cardinality relationships and the Semantic Constraints. 23

Abstract

In the last several years, substantial contributions have been made in the development of techniques to transform and store eXtensible Markup Language (XML) [3] data into relational database models. Although there exists a rich literature in this field, none of the existing procedures provides a complete solution in a single framework. Many of the existing solutions describe ways to automate parts of the conversion. However, a purely automated approach has limitations. Due to the heterogeneous and hierarchical nature of XML for a given input, there can be numerous reasonable relational schemas and automated systems may not be able to reach all such possible schemas. *Our hypothesis is that a human-guided tool can help to identify appropriate relational schemas in complex XML documents for use in highly structured data applications.*

We propose an integrated system that provides an end-to-end solution for the user-guided mapping of XML into relational data. Our system accomplishes the translation in three stages: (1) Parse an XML Schema Definition (XSD) [15] file to suggest relational schemas; (2) allow users to alter the schema using a user interface to determine appropriate schemas; and (3) populate relational tables using an input XML document. The system extracts key information from the XSD input file and introduces new constraints, wherever required, as keys are crucial for querying the resulting relations. The user interface is an essential component

of our system. It enables users to perform additional, meaning-driven operations on the suggested schema to achieve application-specific relational schemas.

The Digital Latin Library (DLL) [1] is a large, collaborative project to build an open collaborative environment for exploring critical editions of Latin texts and facilitating scholarly conversations. In particular, new and existing visualizations of relational data are being developed to support these goals. The Text Encoding Initiative (TEI) [5] is often used for the representation of critical editions. TEI is a markup language to create digital versions of texts as highly structured XML documents. To help achieve the DLL objectives, we are applying our system to read TEI structure into a relational data model in the Improvise visualization environment [26, 27]. This data model, in turn, supports the development of a variety of visualization queries in the DLL software tools.

Chapter 1

Introduction

Critical editions are attempts by scholars to reconstruct texts from fragmentary sources with the most appropriate original meaning. In general, scholars choose an authoritative manuscript and "correct" it using variations from other manuscripts. Critical editions encourage readers to think about the work, deeply, beyond the particular manuscript presentation. They also provide details about the work's sources, historical context, form, and style. The Text Encoding Initiative (TEI) [5] defines guidelines to provide methods for encoding critical editions.

Scholars are interested in the hidden patterns of information in the reconstruction choices and reconstructed text of critical editions, but querying these patterns directly against a TEI document can be complicated and slow. Transforming this XML-based information in these documents into more structured form, such as a relational data model is highly desirable to help as scholars query more deeply to look for meaningful insights in critical editions. Storing the TEI-encoded critical editions in a highly structured form allows scholars to query and analyze the data by performing operations like sorting, grouping, filtering. These capabilities promise to help scholars greatly in reducing the time and ef-

fort needed to identify anomalies and interesting regions in a text. Since TEI is a formal markup language that relies on XML standards, our work concentrates on storing and querying XML data.

The conventional procedures to store and query XML data involve file systems, relational databases, and object-oriented databases. Storing XML data in a file system is convenient and portable but often does not support efficient data queries. Object-oriented databases are generally not structured for processing complex relational queries on large databases. There is considerable interest in using relational database systems, as they support complex queries and can be built on top of powerful and reliable data management services [10,13,14,16,18,21–25]. Moreover, due to the wide popularity of the XML format, a very large volumes of formatted XML data is available online. Correspondingly, there is a growing need for relational access and processing of XML data.

In the Digital Latin Library project, data visualization techniques will play a central role in a collaborative system enabling scholars to analyze, interact, and comment on Latin texts. Visualization is the graphical representation of data to help users obtain understanding and draw conclusions about data. Improvise, a visualization environment that enables users to build sophisticated interactive visualizations [26,27], is the basis for developing visualizations. Improvise accepts primarily relational data as its input. So, to develop data visualizations for critical editions, it is highly desirable to first transform Text Encoding Initiative (TEI) formatted documents into relational datasets.

Towards this aim, our thesis work contributes a system that provides a semi-automated, flexible, schema-aware mapping technique to transform XML data into its equivalent relational data model and allows users to choose and customize the most appropriate relational model for their data analysis and visualization

purposes.

One of the significant challenges for developers is to exchange data among various incompatible systems over the Internet. XML, (eXtensible Markup Language) is a common choice for data exchange and integration over the Internet because diverse applications can read documents in this format. XML is self-descriptive and a simple text-based format to specify structures in a document. The roots of XML evolved from an older standard format, called Standard Generalized Markup Language (SGML), in order to be more suitable for web use. XML is the standard choice for data exchange between different organizations for several reasons. XML tags form the foundation of XML; by using tags, one can describe the meaning of the content. New tags and attributes can be defined, document structures can be nested to any level of complexity, and documents can be validated against a schema specification.

As applications manipulate an increasing volume of XML data, there is a growing need for secure systems to store and query XML documents. A major hurdle of this approach is the need to overcome conflicts between XML and relational data models. Whereas XML data is hierarchical and ordered in nature, relational data models are flat, unordered, and potentially spread across multiple tables. Schema mapping, data mapping, and query mapping assist in overcoming these differences.

- Schema Mapping: This process can involve two different approaches, schema-oblivious and schema-aware. In schema-oblivious XML storage, the final relational schema is independent of the input XML document and employs a fixed, generic, relational database schema. Schema-aware techniques utilize input schema specification documents to help determine relational schemas.

To translate the ordered nature of the XML data model into a relational database, one can map any of the XML order encoding schemes by appending additional columns to handle ordinals of XML elements.

- Data Mapping: Data mapping shreds an input XML document to populate the tables of a relational database generated during the schema mapping phase. This phase is also known as XML document shredding. Various reliable XML parsers are available that shred input XML documents.

- Query Mapping: To query XML data stored in a relational database, one should map the XML queries into equivalent relational queries (SQL statements). This mapping is known as query mapping. Numerous researchers propose the use relational databases for storing and querying XML documents in order to receive the benefits from sophisticated technologies such as query optimization, transaction management, concurrency control, data security and many more available in database management systems.

In summary, to transform an XML document into relational tables takes three steps. The first step is to map the tree structure of the XML document to an equivalent, flat relational schema, using a suitable schema mapping approach. Then, the relational tables are populated by shredding the input XML document. Finally, at runtime, XML queries are translated into SQL and submitted to the RDMBS to fetch the desired result set.

There exists a rich literature of techniques to manage XML documents in relational databases. Many commercial database management systems support XML storage. Nevertheless, none of the mapping techniques or platforms are able to address all the translation ambiguities. Many approaches oppose the philosophy of accepting technical inputs from users [17]. The rationale is that XML

and relational data are too complex and diverse for most users to make appropriate decisions about schema structure. However, TEI document encode extremely complicated hierarchical and associative relationships. Fully automatic translation produces nearly unreadable collections of tables with many meaningless key identifiers.

To overcome this limitation, we propose an XML to relational transformation system that allows users to manually customize an automatically suggested schema. The suggested schema incorporates several mapping strategies from the existing literature and appends certain new strategies related to identity constraints. The system parses an input schema document and suggests a relational schema on which users can perform certain operations to specify an appropriate, meaningful schema for their application. The operations include split tables, merge tables, delete a table, delete columns, rename table, and reset schema. These operations are simple to perform and do not require users to have advanced knowledge of either XML or relational models. The system employs the a form-based user interface to help users perform the operations.

Once a user finalizes a schema, the system prompts them for a corresponding XML document. It then extracts values from the input document to populate tables of the schema. The system uses XML tag paths in the schema document as XPath strings to extract tag and attribute values for storage in tables.

The remaining chapters of the thesis are organized as follows:

- Chapter 2 provides a glimpse of several technologies employed in our system implementation. These technologies include TEI, XML, XML structure constraints, XML Schema Definition Language (XSD), and Relational Database Model Systems (RDBMS).

- Chapter 3 describes some of the more substantial contributions described in the literature about XML-to-relational mapping techniques, and strategies our system inherits from them. It also lists their drawbacks and how our current system advances the existing systems.

- Chapter 4 describes the design specifications, system implementation, user interface and schema operations of our system, with supporting examples.

- Chapter 5 describes the shredding of TEI documents using our mapping technique, along with a system analysis.

- Chapter 6 concludes by reviewing the contributions of thesis and outlining several future directions of our work.

Chapter 2

Background Information

It is a well-known fact that digital computers can only store and process binary system information i.e., bits. Hence, it is essential to transform data present in the form of letters, numbers, pictures and others into an equivalent binary system to make it suitable for computer processing. The transformation of data from other systems into bits is known as encoding and if this transformation is from text to binary then it is known as character encoding. Character encoding fails to convey information regarding meta-data like semantics or structure of a text. Text encoding provides meta information through the means of markup language.

2.1 Text Encoding Initiative

The Text Encoding Initiative (TEI) is a standard for the representation of texts in digital form. A community of scholars, mainly from humanities, social sciences, and linguistics, are organized into a consortium that develops and maintains this standard. Earlier versions of TEI offered a choice of using Standard Generalized

Markup Language (SGML) or eXtensible Markup Language (XML). In the latest version, TEI can be expressed only in XML. TEI is widely accepted by libraries, museums, publishers, and individual scholars to present texts for online research, teaching, and preservation.

Scholarly editions of texts often record some or all of the known variations among different witnesses to the text. Witnesses to a text may include authorial or other manuscripts, printed editions of the work. The TEI provides methods for encoding these critical editions.

2.2 Extensible Markup Language

Extensible Markup Language (XML) can be used to represent the structure of text documents. Its origin is from Standard Generalized Markup Language (SGML) and is defined by World Wide Web Consortium (W3C) XML 1.0 Specification. Both markup languages are machine-readable and human-readable, as they include tags that are distinguishable from the text. Tags are regular words, much like the keywords used in programming languages, making markup languages readily human-readable.

There are three main general categories of markup languages, presentational markup, procedural markup and descriptive markup. XML falls under the category of descriptive markup system, as it represents text components of documents rather than presentation or procedural information. Unlike other markup languages, XML places no limitations on the set of permissible tags and imposes limited rules on their usage, which makes the design of the language quite simple. As the name suggests, XML is extensible; a given set of tags can always have new tags added. The logical structure of an XML document includes a root ele-

ment, parent elements, child elements, attributes within elements and comments. The order in which elements occur in an XML document can be significant. XML documents are legal if they are well-formed and valid. A well-formed XML document has the following characteristics:

- only one root element, which contains all other elements;
- all Elements must be properly nested; and
- each opening tag must have a closing tag.

A valid XML document must satisfy following criteria:

- it is well-formed, and
- it is consistent with the grammatical rules specified in a schema definition document (which may be XSD, DTD, or other schema description format).

An XML document must be well-formed but being valid is optional.

2.3 Constraints on XML Structure

As mentioned earlier, XML is flexible and easy to compose, but these benefits of XML come with certain tradeoffs. For example, two documents containing the same information may be created with distinct XML tag names, as there is no restriction on the set of legitimate tags. This results in different XML documents representing the same data, making processing of these documents a more difficult task. To overcome such complications, we need supporting documents that specify the constraints on the legal structure and contents of XML documents. Schema language documents, like Document Type Definition (DTDs), Regular

Language for XML Next Generation (Relax-NG), and W3C XML Schema Definition (XSD) accomplish this objective. In addition, schema documents serve the purpose of validating XML documents.

2.4 XML Schema Definition

Of all the available schema specification formats, XSDs are quite popular because of their prominent features. They define building blocks of XML documents, elements, and attributes that can appear in an XML document; the default and fixed values of elements and attributes; and the number and order of child elements. Unlike other schema documents, XSDs are verbose and define the data types of attributes and elements. The syntax of XSD is similar to that of XML, which makes parsing XSD files accessible simple to implement and apply.

Though DTDs are one of the preferred schema specification formats they are slowly fading out as they suffer from certain drawbacks such as not allowing definition of the types of elements or attributes. They also have no way to specify constraints between elements, which us needed to determine relational key semantics. Considering these shortcomings, we chose XSD as the XML schema document for our schema-aware system. The following section expands on some of the vital components of XSD.

2.4.1 XSD Constructs

XML Schema Definition files allow a variety of valid components and component relationships. In this section, we introduce the constructs and tags of XSD files that are important for our purposes.

```
<xs:attribute name='CustomerID' type='xs:token'/>
<xs:attribute name='ShippedDate' type='xs:dateTime'/>
```

Figure 2.1: XML Schema code, defining attribute tags.

- Attribute: The attribute tag defines the name and type of attributes that may occur in an XML documents. Figure 2.1 depicts attributes in an XSD file. They are optional by default and may have a default or fixed value specified. An attribute is always declared as a simple type.

- Element: Like attribute, the element tag in an XSD file also provides information about the name, type, and occurrence of elements in the corresponding XML document. The tag also defines constraints including like minimum and maximum occurrences, default and fixed values. Figure 2.2 shows an example of elements in an XSD file. An element is of simple type if it contains just text and is of complex type if it holds other elements or has attributes.

- Complex Type: A complex type element is a type definition for XML elements that may contain elements and attributes. The following figure 2.3 illustrates a sample complex type element. It defines the structure, content,

```
<xs:element name='CompanyName' type='xs:string'/>
<xs:element name='ContactName' type='xs:string' />
<xs:element name="Phone" type="xs:integer" maxOccurs="unbounded"/>
```

Figure 2.2: XML Schema code, defining element tags.

```
<xs:complexType name='AddressType'>
  <xs:sequence>
    <xs:element name='Address' type='xs:string'/>
    <xs:element name='City' type='xs:string'/>
    <xs:element name='Region' type='xs:string'/>
    <xs:element name='PostalCode' type='xs:string' />
    <xs:element name='Country' type='xs:string'/>
  </xs:sequence>
</xs:complexType>
```

Figure 2.3: XML schema code, defining complex type element.

and attributes of an element. The content can includes special, reserved tag elements like <group>, <sequence>, <choice>, and <all>. Similarly, attributes of complex type can be defined using tags <attribute>, <attributeGroup>, and <anyAttribute>.

- Key: The <key> element is reserves as a means to define a primary key relationship, in the same manner as a primary key in the relational database model unique and appear explicitly in the XML document. A key element contains <name>, <selector> and <field> child elements. The <field> element specifies the element or attribute names that must be unique. The <selector> tag contains the XML path of the values that occur in a sibling <field>.

```
<xs:key name='CustomerIDKey'>
  <xs:selector xpath='Customers/Customer'/>
  <xs:field xpath='CustomerID'/>
</xs:key>
<xs:keyref name='CustomerIDKeyRef' refer='CustomerIDKey'>
  <xs:selector xpath='Orders/Order'/>
  <xs:field xpath='CustomerID'/>
</xs:keyref>
```

Figure 2.4: Occurrences of <key> and <keyref> elements in an XSD.

- Key Reference: Complementary to <key> the <keyref> element contains <name>, <selector>, <field>, and <refer> elements and is similar to a foreign key in the relational model. The tag <refer> holds a <key> element which is a referred element of <keyref>. Both <key> and <keyref> elements define identity constraints of XML documents. An example is shown in figure 2.4.

2.5 Relational Database Management Systems

Database often contain large amounts of data. The primary goals of a Database Management System (DBMS) are to provide convenient and efficient means to store and retrieve structured collections of information. Data management also facilitates the integrity of the information stored.

A data model provides a way to describe the design of a database. There are various data models available in widespread use. The relational data model is the most widely used data model. It uses a collection of tables to represent data and the relationships among those data. Relational Database Management Systems (RDBMS) often play a pivotal role in the growth of an organization as the amount and complexity of their data increases over time.

A relational schema defines the structure of a database, which including tables, columns/attributes, primary and foreign keys and type information. A relational database consists of a collection of tables, each of which is assigned a unique name. Each table can have multiple columns, and each column has a unique name and data type. Tables hold tuples or rows. Candidate keys specify alternative ways of uniquely identifying rows. One of the candidate keys of a relation can be chosen as the table's primary key. A foreign key is a set of one or more columns in which the values refer to a primary key of another table, such that each value of a foreign key is the same as a value of a primary key in the referred table.

To demonstrate our system functionality, we use the XML Schema Definition (XSD) shown in figure 2.5. The document defines the structure of an XML file that includes customers, orders, and shipping information. In the next chapter, we use this XSD file to illustrate how the system suggests a schema then supports operations for modifying that schema.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name='Root'>
  <xs:complexType>
   <xs:sequence>
    <xs:element name='Customers'>
     <xs:complexType>
      <xs:sequence>
       <xs:element name='Customer' type='CustomerType' minOccurs='0' maxOccurs='unbounded' />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name='Orders'>
     <xs:complexType>
      <xs:sequence>
       <xs:element name='Order' type='OrderType' minOccurs='0' maxOccurs='unbounded' />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:key name='CustomerIDKey'>
   <xs:selector xpath='Customers/Customer'/>
   <xs:field xpath='CustomerID'/>
  </xs:key>
  <xs:keyref name='CustomerIDKeyRef' refer='CustomerIDKey'>
   <xs:selector xpath='Orders/Order'/>
   <xs:field xpath='CustomerID'/>
  </xs:keyref>
 </xs:element>
 <xs:complexType name='CustomerType'>
  <xs:sequence>
   <xs:element name='CompanyName' type='xs:string'/>
   <xs:element name='ContactName' type='xs:string'/>
   <xs:element name='ContactTitle' type='xs:string'/>
   <xs:element name='FullAddress' type='AddressType'/>
   <xs:element name="Phone">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="Home" type="xs:string" minOccurs='1' maxOccurs="unbounded"/>
     <xs:element name="Work" type="xs:string" />
      </xs:sequence>
     </xs:complexType>
   </xs:element>
   <xs:element name='Fax' minOccurs='0' type='xs:string'/>
  </xs:sequence>
  <xs:attribute name='CustomerID' type='xs:token'/>
 </xs:complexType>
```

```
<xs:complexType name='AddressType'>
 <xs:sequence>
  <xs:element name='Address' type='xs:string'/>
  <xs:element name='City' type='xs:string'/>
  <xs:element name='Region' type='xs:string'/>
  <xs:element name='PostalCode' type='xs:string' />
  <xs:element name='Country' type='xs:string'/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name='OrderType'>
 <xs:sequence>
  <xs:element name='CustomerID' type='xs:token'/>
  <xs:element name='EmployeeID' type='xs:token'/>
  <xs:element name='OrderDate' type='xs:dateTime'/>
  <xs:element name='RequiredDate' type='xs:dateTime'/>
  <xs:element name='ShipInfo' type='ShipInfoType'/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name='ShipInfoType'>
 <xs:sequence>
  <xs:element name='ShipVia' type='xs:integer'/>
  <xs:element name='Freight' type='xs:decimal'/>
  <xs:element name='ShipName' type='xs:string'/>
  <xs:element name='ShipAddress' type='xs:string'/>
  <xs:element name='ShipCity' type='xs:string'/>
  <xs:element name='ShipRegion' type='xs:string'/>
  <xs:element name='ShipPostalCode' type='xs:string'/>
  <xs:element name='ShipCountry' type='xs:string'/>
 </xs:sequence>
 <xs:attribute name='ShippedDate' type='xs:dateTime'/>
</xs:complexType>
</xs:schema>
```

Figure 2.5: Example XML Schema Definition, for XML files containing customer order information.

Chapter 3

Overview of Mapping Techniques

XML to relational mapping techniques can be categorized into schema-aware and schema-oblivious based on the inputs used for this transformation. In the schema-aware approach, mapping techniques make use of XML schema specifications to determine the relational schema whereas the schema-oblivious techniques do not consider XML schema instead directly use XML document. Document Type Descriptors (DTD), XML Schema Definition Language (XSD), Relax NG are the various XML schema languages. This section reviews some of the mapping techniques and relates them with our system.

3.1 Schema Oblivious Techniques

3.1.1 Edge Mapping

Edge Mapping [14] is one of the earliest works done in the field of converting XML data into Relational data. The primary goal of this work is to build a system that supports storing and querying of simple XML schemes. This system does not prompt for any inputs from its users and does not require any of the

```
<Customer CustomerID="LETSS">
    <CompanyName>Let's Stop N Shop</CompanyName>
    <ContactName>Jaime Yorres</ContactName>
    <ContactTitle>Owner</ContactTitle>
    <Phone>
      <Home>(415) 555-5938</Home>
      <Work>(415) 555-5934</Work>
    </Phone>
</Customer>
```

Figure 3.1: Sample XML code, defining a Customer data.

supporting XML schema specification documents.

This system assigns an object id to each element present in the XML document. It adds an edge between the element and corresponding sub element or its data types. Each edge is labeled with the name of its sub element. The resulting relational schema stores all the edge information in a single table titled as the Edge table. The table columns include the source object id and target of an edge, the edge label, flag that signifies whether the edge is a reference or a data type and an ordinal number as the edges are numbered. Table 3.1 represents the resulting schema obtained by applying Edge mapping technique on the customer XML data shown in figure 3.1.

As mentioned earlier, this system only handles simple XML documents. Independent of the input XML document the output relational schema remains the same. As only one table holds all the information, querying on the table becomes

Edge Source	Ordinal	Name	Flag	Target
1	1	CustomerID	String	LETSS
1	2	CompanyName	String	Let's Stop N Shop
1	3	ContactName	String	Jaime Yorres
1	4	ContactTitle	String	Owner
1	5	Phone	Ref	2
2	1	Home	String	(415) 555-5938
2	2	Work	String	(415) 555-5934

Table 3.1: Edge Schema representation of the Customer Data in figure 3.1.

cumbersome if there is a high degree of nesting in the original XML document. This method is not a good way to store a large, deeply nested XML documents.

3.1.2 DTD- Independent Schema Mapping

The paper by Zin Mar Kyu et al. [17] suggests a schema oblivious mapping technique and explains the development of a relational schema from an input XML document. This process includes two different mappings DTD-independent schema mapping which extract table names and attributes to create a document schema and data mapping extracts XML and stores it into relational tables.

In DTD-independent schema mapping, the system parses through an input XML document and retrieves elements, attributes, and their values into element list and attribute list. The columns of these lists include name, value, and id or start id, end id, and parent id. It parses the element list to generate a new

list comprising of no repeated elements. From the latest element list the system extracts the table name list, all the elements with more than one child elements makes entry into this list. Elements in the table name list, attribute list and element list are mapped to create a relational schema. As mentioned earlier, each table name element has an associated start id, which acts as primary key. Data mapping makes use of element values from the initial element list to populate the tuples of relational schema tables.

This approach is reliable if the XML document does not have a supporting schema document, users are interested in regression mapping i.e., Relational to XML mapping. Though the paper provides one of the efficient schema-oblivious mapping techniques it fails in certain aspects. The resulting schema tables only hold customized primary keys, introduced by the system and each table has the end id column, which can be an overhead. The final relational schema might involve certain trivial tables. For example tables with their only columns as start id and end id and this paper does not outline a way to discard these tables.

3.2 Schema Aware Techniques

3.2.1 Shared Inlining

The mapping strategies proposed by Shanmugasundaram et al. [24] are the primary ones to illustrate that it is indeed possible to use commercial relational database systems to evaluate powerful queries over XML documents. It has overcome some of the problems of storing all the XML data in a single relational table. This strategy used Document Type Descriptors (DTDs) [11] as their schema language to generate a relational schema. Then it parses the XML document

```
<! ELEMENT Customers (Customer*)>
<! ELEMENT Customer (CompanyName, ContactName, ContactTitle)>
<! ELEMENT CompanyName (#PCDATA)>
<! ELEMENT ContactName (#PCDATA)>
<! ELEMENT ContactTitle (#PCDATA)>
```

Figure 3.2: DTD document, defining Customer data.

conforming input DTD and loads them into tuples of relational tables.The figure 3.3 shows the equivalent Shared Inlining Mapping of a DTD Customer data in the figure 3.2.

Our system incorporates one of these mapping approaches known as shared inlining, elements with multiple occurrences transforms into tables whereas elements with a single occurrence are columns of the table corresponding to its parent element. Generating the relational schema for this mapping strategy is more complicated than the simple edge schema but anything of this nature is essential for large, deeply nested XML documents.

3.2.2 Constraints-Preserving Inlining

The algorithm suggested by Dongwon Lee et al. [20] known as Constraints-Preserving Inlining Algorithm (CPI) converts an XML schema specification to

```
Customer (CompanyName, ContactName, ContactTitle)
```

Figure 3.3: Shared Inlining Mapping of Sample Data.

a relational schema while preserving semantic constraints of the original XML schema. Here the XML schema specification denotes the document type definition (DTD) document. Many of the XML-to-Relational transformation algorithms suggested till then was mainly focusing on the structural conversion [12–14, 24] ignoring the semantic constraints hidden in the original DTD. This algorithm identifies various semantics constraints in the original DTD and preserves them by rewriting the same in the final relational schema.

In a DTD declaration, there exist four possible cardinality relations between an element and its sub-elements.

1. (0,1): An element can have either zero or one sub-element.

2. (1,1): An element must have one and only one sub-element.

3. (0,N): An element can have zero or more sub-elements.

4. (1,N): An element can have one or more sub-elements.

From the above relations, three constraints can be inferred. The first constraint is if a sub-element can be null or not. This can be implemented in a relational database by using the "NULL" or "NOT NULL" clause. The second constraint restricts an element to have at most one sub-element and is known as singleton constraint [28]. This constraint is one kind of Equality-Generating Dependency (EGDs) and can be enforced by "UNIQUE" construct. Final constraint is for a given an element, whether its sub-element should occur or not. This is one of the categories of tuple-generating dependencies. TGDs in a relational model require that some tuples of a certain form be present in the table. Child and Parent constraints are two useful forms of TGDs.

Cardinal Relations	Not Null	EGDs	TGDs
(0,1)	×	✓	×
(1,1)	✓	✓	✓
(0,N)	×	×	×
(1,N)	✓	×	✓

Table 3.2: Cardinality relationships and the Semantic Constraints.

CPI algorithm extends the structural mapping of DTD to relational schema by introducing semantic constraints. In addition to CPI, Dongwon Lee et al. [18–20] suggest two more algorithms Nesting-based Translation (NeT) and Constraints-based Translation (CoT) for translating Relational schema to XML schema. This paper makes significant contribution to the mapping techniques but the XML schema used in these techniques is DTD, which is not the latest version of XML schemas. Parsing DTD document is cumbersome when compared to that of parsing XSD, latest XML schema version. DTD documents does not provide information corresponding to the data types of the elements, which is a major drawback as this information is very vital for determining relational schema.

3.2.3 ShreX

ShreX [8, 9] is an XML to Relational mapping framework, provided the first end-to-end solution to the relational storage of XML data problem. It makes use of XML schema to establish mappings from XML to relational schema. It contributed several new mapping strategies to the existing ones [10, 13, 14, 23] and provides an API to access the mapping information. A user can specify the mappings by including annotations in an XML schema. As there exist multiple methods to associate annotations, ShreX is capable of expressing a wide range of

mappings.

The system allows users to annotate either manually or by using the interface provided. The annotation processor parses an annotated XML schema to check the mapping validity and creates the corresponding relational schema. Then the system accepts an XML document as input and loads the tuples into the relational database.

Annotations specify how to represent elements of an XML schema definition language in the corresponding relational schema. The annotations supported by the system are modifying an attribute or an element into a new table, altering a table name or a column name or a column type and mapping an element or an attribute to a CLOB column.

Though ShreX design overlaps with the design of our system, there exist considerable differences between both the systems. Unlike ShreX, our system extracts the identity constraints from an input XSD, appends the same in the relational schema. It utilizes counters as keys when the input XML schema definition document fails to provide key constraint information. But, ShreX uses only counters as the primary and foreign keys in the relational schema irrespective of the information provided in the input document. Our system supports operations like Split a Table, Merge Tables, Delete a Table, Delete Columns and Reset Schema. Users can perform all these operations using the user interface. This makes performing these actions quite modest even for an amateur user.

Chapter 4

Design and Implementation

In this chapter, we present the design, implementation, and analysis of our system in following steps:

1. Using XML Schema Object Model (XSOM) [7], the system parses XML Schema Definition file and generates equivalent Relational Database Model.

2. Various schema operations our system supports enabling users to alter the relational schema.

3. Once the user finalizes schema, the next step is to fill the rows of the schema tables. For this step, the system uses DOM parser [2] and accepts an XML document as input.

4. The key segments of the user interface.

4.1 Suggest Schema

In this step, the System parses through the input XSD and extracts various data components. These components include obtaining table names, columns, keys,

and XML paths. Besides, it also handles a variety of elements by introducing new keys and tables wherever necessary. This step involves following segments:

1. Excerpt Table Names and Columns
2. Extract Keys
3. Introduce Keys
4. Handling Recurring Elements
5. Generate XML Paths

4.1.1 Excerpt Table Names and Columns

Initially, the system excerpts elements, attributes, and their data types. Elements can be either complex type or simple type. All elements with their types as complex are table names in the resulting suggested schema. The simple type child elements and the attributes of the complex type elements are columns of the respective tables. The Suggested schema in the Figure 4.1 shows the complex type element "FullAddress" and its child elements of the XSD file in figure 2.5 are mapped into a table name and corresponding columns in the suggested relational schema.

4.1.2 Extract Keys

Here, the system fetches the identity keys and associated element information from the schema document. Identity keys indicate either primary keys or foreign keys and or unique keys. The associated element information involves the table names to which each of these keys belongs. In the case of foreign keys, the

system extracts additional information about the referring table. The system appends this key data to the existing schema information. The suggested schema in figure 4.1 demonstrates this instance, the key with name "CustomerIDKey" in the example XML schema document is the primary key of the table "Customer".

4.1.3 Introduce Keys

The system loops through all the tables in the schema and checks if each of them has any keys associated with them because relational schema makes no sense without keys. If they do not possess either a primary key or a foreign key, the system checks parent table for the keys.

If the parent table holds a primary key, then the system adds this column as the new foreign key of the table with parent table as the reference and adds the same to the child table column list. The "CustomerID" key of the table "FullAddress" in the suggested schema is an example of this occurrence.

If the parent table includes a foreign key but not a primary key, then the system introduces a new primary key to the parent table. The name of this primary key includes the table name and the string "PKID" separated by a hyphen (-). The same key acts as the foreign key for the table with reference table being the parent table. Here, the columns are unique if their labels, data types and XML paths are same. The primary key "Order-PKID" of the table "Order" in the suggested schema illustrates this instance.

4.1.4 Handling Recurring Elements

Elements in the schema document can occur more than once. If elements with complex type are recurring then no problem persists as they form a separate table

and each of its occurrence in the XML document will be a separate row in the table. Whereas, a recurring simple element is not fair as a single table cannot support multiple occurrences of the same column.

If any of the simple type elements are recurring, the system introduces a new table. This recurring simple element and the primary key of the parent table are the new table columns. Here parent table indicates the parent of the simple element. The new table key is the foreign key and refers to the primary key of the parent table. The table "Phone-Home" of the suggested schema falls into this category

4.1.5 Generate XML Paths

While extracting elements and columns from the XML schema definition document. The system constructs XML paths [6] for all the parent elements or table names. These XML paths are useful for populating the table values.

If the parent element of a table is also a table in the schema, the table will inherit the XML path from its parent. The system implements this inheritance as most of the child and parent tables share the primary keys. For retrieving the table values from an XML document, the system considers the XML path of the parent instead of the actual XML path, as the primary key values are present in the path of the parent table. XML paths of all the tables generated by the system are shown in the suggested schema.

```
================================================================
1 - TableName, 2 - TableColumns:ColumnTypes, 3 - PrimarKey, 4 - ForeignKey:ReferencedTable, 5- XPaths
================================================================
1.1. Customer
1.2. CompanyName:string;ContactName:string;ContactTitle:string;Fax:string;CustomerID:token;
1.3. CustomerID:token
1.4.
1.5. /Root/Customers/Customer
----------------------------------------------------------------
2.1. FullAddress
2.2. Address:string;City:string;Region:string;PostalCode:string;Country:string;CustomerID:token;
2.3.
2.4. CustomerID:token:Customer
2.5. /Root/Customers/Customer
----------------------------------------------------------------
3.1. Phone-Home
3.2. Home:string;CustomerID:token;
3.3.
3.4. CustomerID:token:Customer
3.5. /Root/Customers/Customer
----------------------------------------------------------------
4.1. Phone
4.2. Work:string;CustomerID:token;
4.3.
4.4. CustomerID:token:Customer
4.5. /Root/Customers/Customer
----------------------------------------------------------------
5.1. Order
5.2. CustomerID:token;EmployeeID:token;OrderDate:dateTime;RequiredDate:dateTime;Order-PKID:integer;
5.3. Order-PKID:integer
5.4. CustomerID:token:Customer
5.5. /Root/Orders/Order
----------------------------------------------------------------
6.1. ShipInfo
6.2. ShipVia:integer;Freight:decimal;ShipName:string;ShipAddress:string;ShipCity:string;ShipRegion:string;
     ShipPostalCode:string;ShipCountry:string;ShippedDate:dateTime;Order-PKID:integer;
6.3.
6.4. Order-PKID:integer:Order
6.5. /Root/Orders/Order
----------------------------------------------------------------
```

Figure 4.1: Suggested Relational Schema for the example XSD in figure 2.5.

Our system is examined against various XML Schema documents whose structure resembles the one shown in the figure 2.5 and it works flawlessly generating relational schemas with the above features. The structure accommodates nested complex type elements with a varying number of occurrences, all kinds of identity keys and child elements of complex types with zero or multiple occurrences.

Figure 4.1 describes the suggested schema for the example XSD in figure 2.5. The schema information presented in the 4.1 is organized in the following order:

1. Name of the table.
2. List of attribute information separated by ";" and attribute information includes column name and its type.
3. Primary key (PK) with its data type separated by ":".
4. Foreign key (FK), data type and referred table partitioned by ":".
5. XML path that should used for populating tables.

4.2 Alter Schema

The suggested relational schema may not convince the user. So, our system allows users to perform a set of operations on the suggested schema using an inbuilt user interface. The set of operations to alter suggested schema include:

1. Split Tables
2. Merge Tables
3. Delete Table Columns
4. Delete a Table
5. Rename Tables
6. Reset Schema

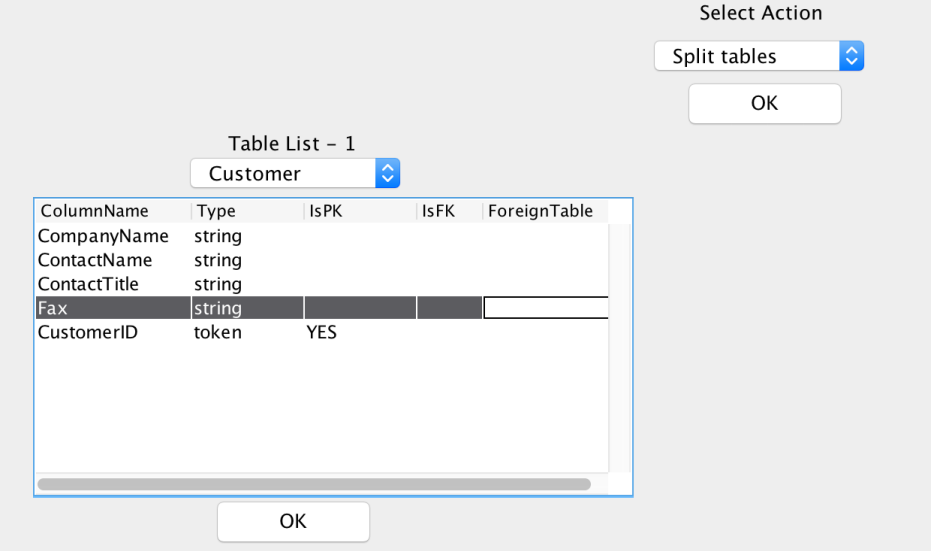

Figure 4.2: Split tables operation on the "Customer" table.

4.2.1 Split Tables

The number of columns in tables suggested by the tool might be large or some columns in a table are vital and they need to be a part of a separate table. To address scenarios like these, this tool allows the user to perform Split operation on Tables.

The user interface allows users to select multiple columns, which need to be part of the new table. Then, the system creates a new table, whose column list includes selected columns and a foreign key referring to the parent table. The parent table name appended with "-1" is the name of the new table. Here, the parent table is the one on which split action occurred. XML path of the new table is similar to that of its parent table. The primary key of the parent table will be the foreign key of the new table.

The figure 4.2 illustrates creating a new table with "Fax" as column by choosing Split table operation on the "Customer" table and the figure 4.3 shows the newly formed table "Customer-1" and altered table "Customer".

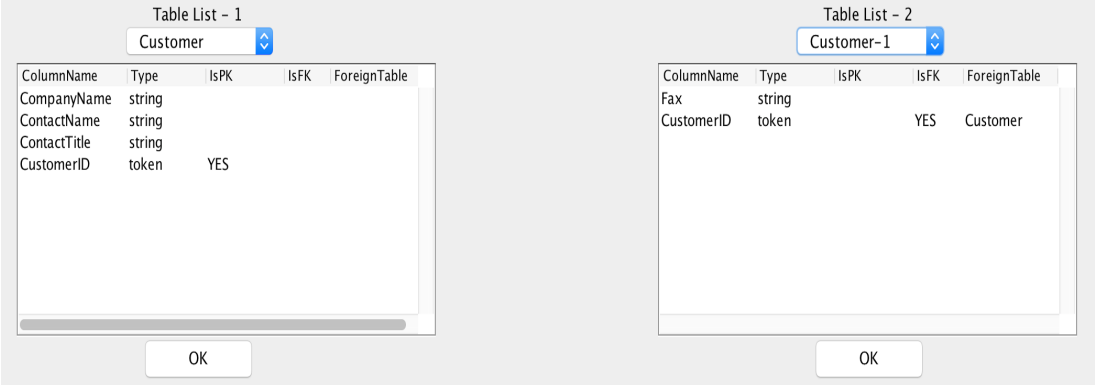

Figure 4.3: "Customer" and "Customer-1" tables after Split Tables action.

4.2.2 Merge Tables

In some instances, suggested schema is more effective if tables with similar properties merge to form a new table.

The user can perform this merge action by choosing two different tables with similar properties in the user interface. This forms the new table with its columns being an augmentation of all non-key columns from the selected tables and a primary or foreign key column. After this operation, the resulting schema includes the new merged table and excludes two tables used for the merge operation.

The label for the new table is the combination of two merged table names separated by a hyphen('-'). The similar properties correspond to tables with

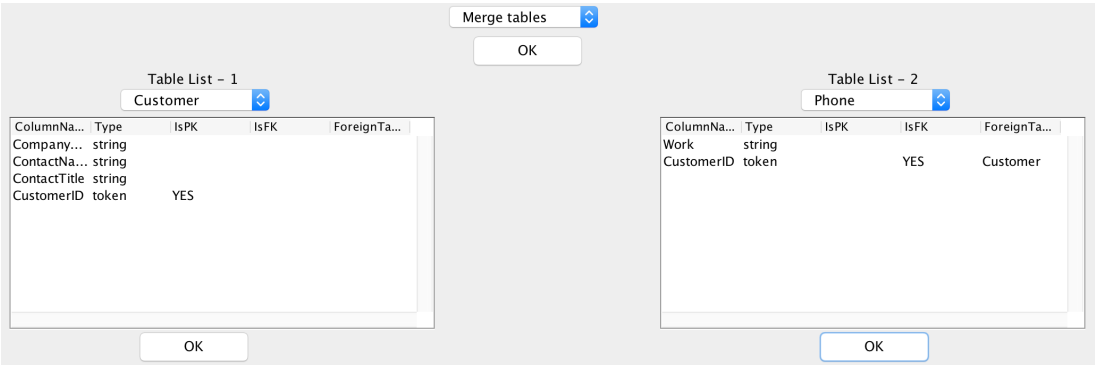

Figure 4.4: Merge tables operation on tables "FullAddress" and "Phone-Home".

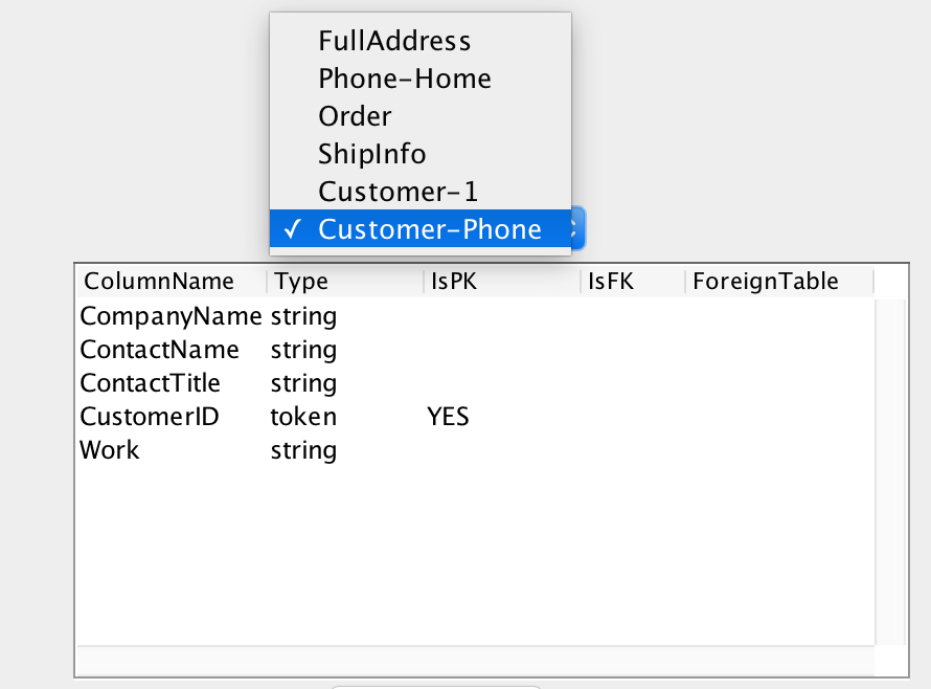

Figure 4.5: Updated table list and new table attributes after Merge Tables operation.

same primary keys or a foreign key of one table referring to the other table chosen for the merge operation.

In the figure 4.4, two tables “Customer” and “Phone” are chosen for merge operation and the resulting table list and new table “Customer-Phone” columns are shown in figure 4.5.

4.2.3 Delete Table Columns

The system gives a provision to delete some columns from tables. The user can choose a table of his interest from the user interface and then corresponding column list gets updated accordingly, the user can choose one or more columns to delete. The system updates the table with the new column list. This operation does not acknowledge deletion of primary and foreign keys.

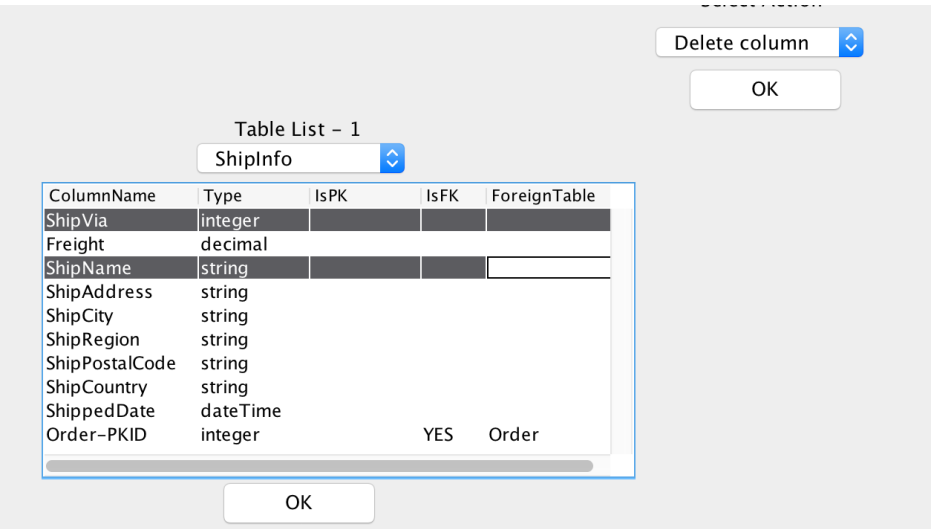

Figure 4.6: The columns of the ShipInfo table selected for deletion.

This task is helpful if any of the column lists in the suggested schema include keys with minimal importance. Figure 4.6 illustrates a user selecting columns of "ShipInfo" table to perform delete operation and the figure 4.7 shows the resulting columns of "ShipInfo" table.

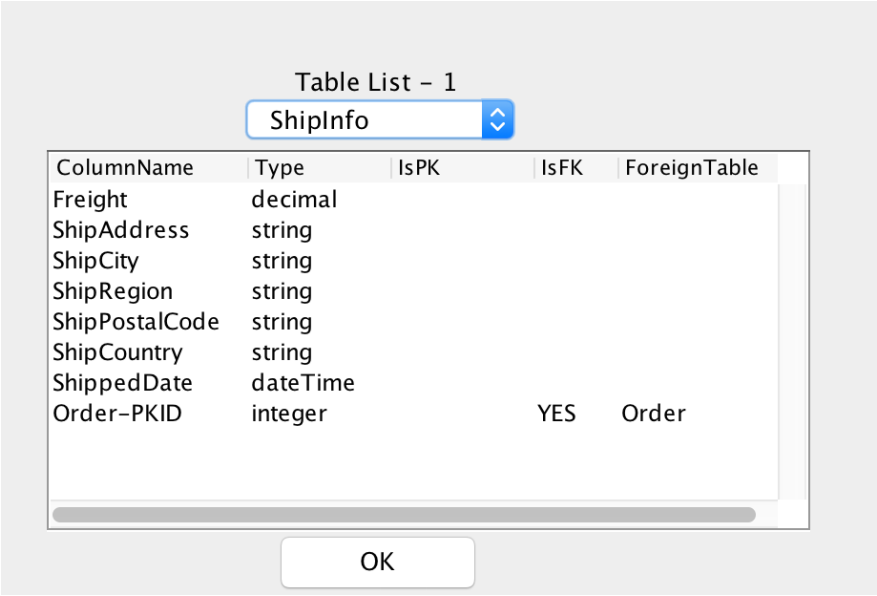

Figure 4.7: The "ShipInfo" table after Deleting columns.

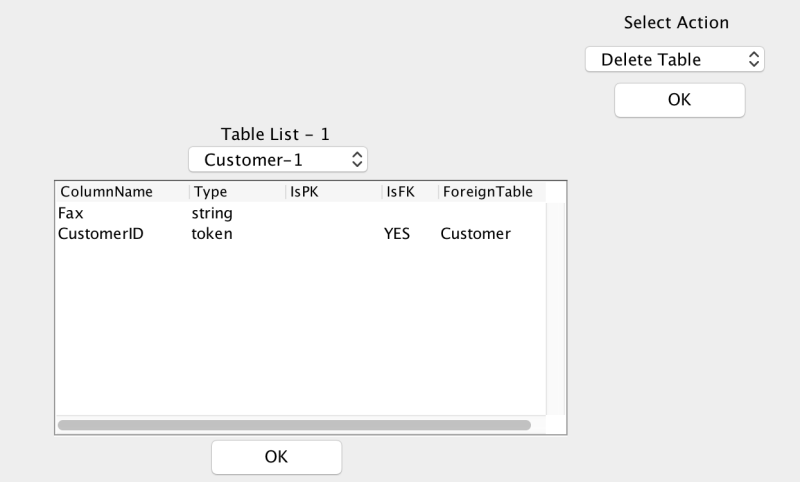

Figure 4.8: The "Customer-1" table selected for deletion.

4.2.4 Delete a Table

Users can delete tables from the schema if no other table refers to them and if they are not a part of suggested schema. To perform this action users can select a table from table list using the user interface and the system deletes the table from resulting schema accordingly.

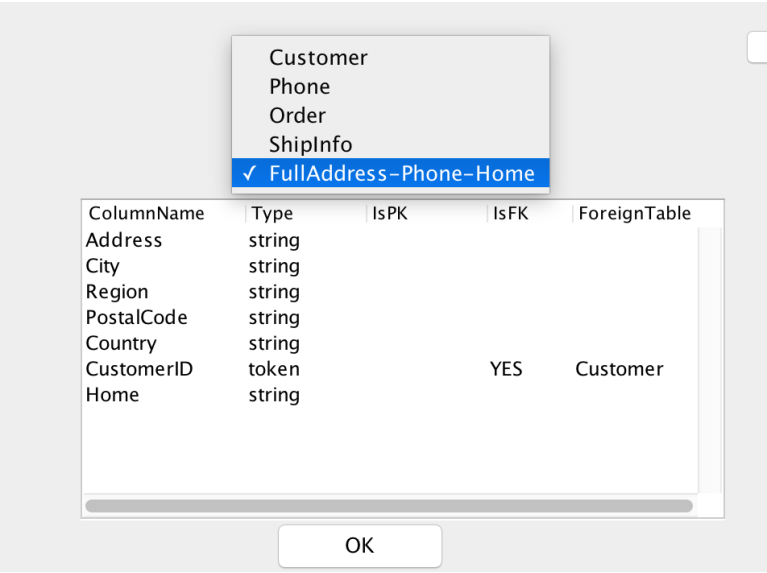

Figure 4.9: The "Customer-1" table removed from the new schema table list.

The figure 4.8 depicts deletion of the table “Customer-1” and the table list of the resulting schema is shown in the figure 4.9.

4.2.5 Rename Tables

In some of the previous schema operations, the system has generated some new tables and labeled them using some predefined conventions. These predefined conventions might not be the appropriate labels for the tables. To address these circumstances, this system provides flexibility to re-label the newly constructed tables with the choice of the user.

To perform this action, the user needs to select the “Rename” operations followed by the table to rename in the user interface and then provide a new label for the elected table in the text editor. Figures 4.10 and 4.11 shows the name of the table is changed from “FullAddress-Phone-Home” to “CustomerAddress”. A sample final schema after a series of operations is shown in the figure 4.14.

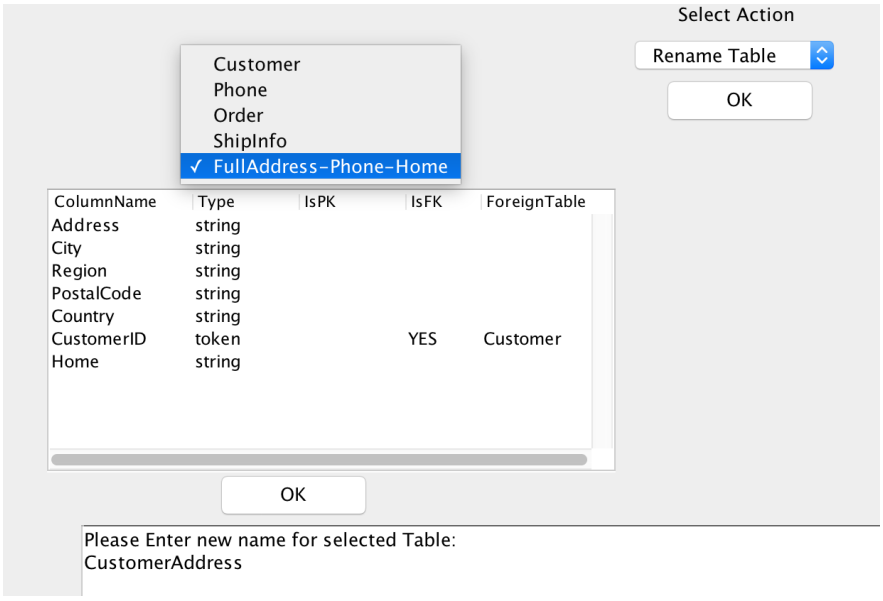

Figure 4.10: The “FullAddress-Phone-Home” table is being renamed as “CustomerAddress”.

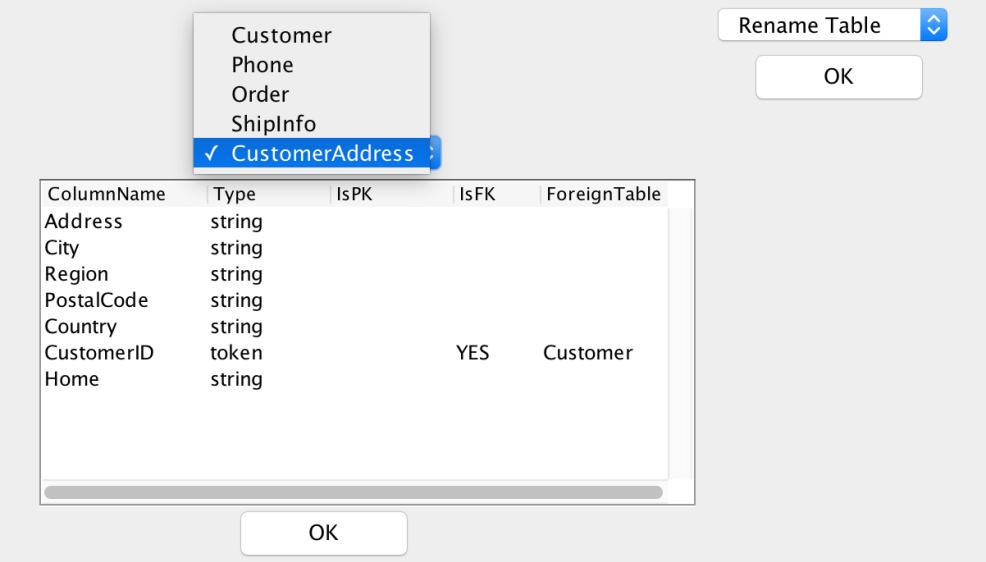

Figure 4.11: The new "CustomerInfo" table attributes after rename operation.

4.2.6 Reset Schema

This allows the user to regain the suggested schema after performing a set of operations on them. This helps the user to correct his actions by restoring the schema to the original one. The user can perform this action just by selecting "Reset" operation from the operations list from the user interface.

The figure 4.12 presents the resulting table list after performing the reset operation and the schema resembles same as the suggested schema shown in the figure 4.1.

4.3 Populate Tables

After performing a series of operations on the suggested schema, users confirms the final schema. The system is now accessible to populate the table values from an XML document. The XML document should obey the structure of the XML schema definition. Once the user finalizes the schema, the system requests for

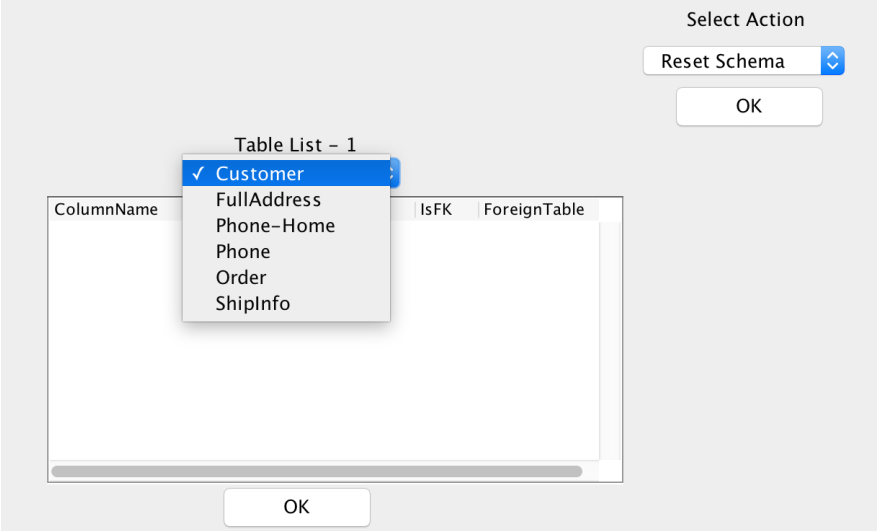

Figure 4.12: The new table list after performing reset operation.

the input XML document.

To populate the schema tables, the system loops through the final schema and retrieves table names and their individual XML paths. Using these paths, the system fetches all the node lists existing in the respective paths. Then, the system iterates through the column elements and excerpts the same from the elements of the node lists to fill the tables.

The Primary and Foreign key values introduced by the system that are not part of the input XML Schema document are populated using counters.

CompanyName	ContactName	ContactTitle	CustomerID
Great Lakes Food Market	Howard Snyder	Marketing Manager	GREAL.
Hungry Coyote Import Store	Yoshi Latimer	Sales Representative	HUNGC.
Lazy K Kountry Store	John Steel	Marketing Manager	LAZYK.
Let's Stop N Shop	Jaime Yorres	Owner	LETSS.

Figure 4.13: "Customer" table with values.

4.4 The user interface

The user interface is one of the key component of our system as it enables users in selecting appropriate schema by performing specific operations on suggested schema. For the implementation of the user interface, various concepts of Swing, a graphical user interface toolkit [4] are utilized. The user interface implemented by our system is shown in the figure 4.15.

```
=================================================================
1 - TableName, 2 - TableColumns:ColumnTypes, 3 - PrimarKey, 4 - ForeignKey:ReferencedTable, 5- XPaths
=================================================================
1.1. Customer
1.2. CompanyName:string;ContactName:string;ContactTitle:string;CustomerID:token;
1.3. CustomerID:token
1.4.
1.5. /Root/Customers/Customer
-----------------------------------------------------------------
2.1. Phone
2.2. Work:string;CustomerID:token;
2.3.
2.4. CustomerID:token:Customer
2.5. /Root/Customers/Customer
-----------------------------------------------------------------
3.1. Order
3.2. CustomerID:token;EmployeeID:token;OrderDate:dateTime;RequiredDate:dateTime;Order-PKID:integer;
3.3. Order-PKID:integer
3.4. CustomerID:token:Customer
3.5. /Root/Orders/Order
-----------------------------------------------------------------
4.1. ShipInfo
4.2. Freight:decimal;ShipAddress:string;ShipCity:string;ShipRegion:string;ShipPostalCode:string;ShipCountry:string
     ShippedDate:dateTime;Order-PKID:integer;
4.3.
4.4. Order-PKID:integer:Order
4.5. /Root/Orders/Order
-----------------------------------------------------------------
5.1. CustomerAddress
5.2. Address:string;City:string;Region:string;PostalCode:string;Country:string;CustomerID:token;Home:string;
5.3.
5.4. CustomerID:token:Customer
5.5. /Root/Customers/Customer
-----------------------------------------------------------------
```

Figure 4.14: Sample relational schema after performing a series of operations.

4.4.1 Action Combo Box

This combo box is a drop-down list for the operations supported by the system on the suggested schema. It allows the user to select his choice of action on the schema. The operation list includes "Split Tables", "Merge Tables", "Delete Columns", "Delete a Table", "Rename Table" and "Reset Schema". Based on the user's choice, the system updates the user interface. For instance, if the user opts to merge then the system activates both the table combo boxes allowing the user to select his choice of tables.

4.4.2 Table Combo Boxes

The user interface comprises of two table combo boxes, each of them displays the table lists. The nature of the operation decides if a table box is active or not. Primary Table box is active during almost all the operations, whereas the secondary one is active only during the merge operation. As the user selects a table, the system updates the table models with columns of the table.

4.4.3 Table Models

Two table models used in the user interface displays the comprehensive information regarding the selected table. This includes column names, column types, and if a column is a primary or a foreign key. They allow users to select the rows during "Split Tables" and "Delete Columns" actions. Once a user selects the table combo box the corresponding table model gets activated.

4.4.4 Text Editor

The text area occupies a major part of the user interface. Initially, it displays the suggested schema, as the user performs operations, the system updates the schema on the display. This text area also allows the user to input the new table name during the "Rename Table" operation. If the user selects the operation type as "Rename Table" and the table of his choice the display area clears the schema and waits for the user to enter the new table name. Once the user concludes rename operation, the text area gets back to the schema display mode. In this way, the text area serves multi-purpose in the user interface.

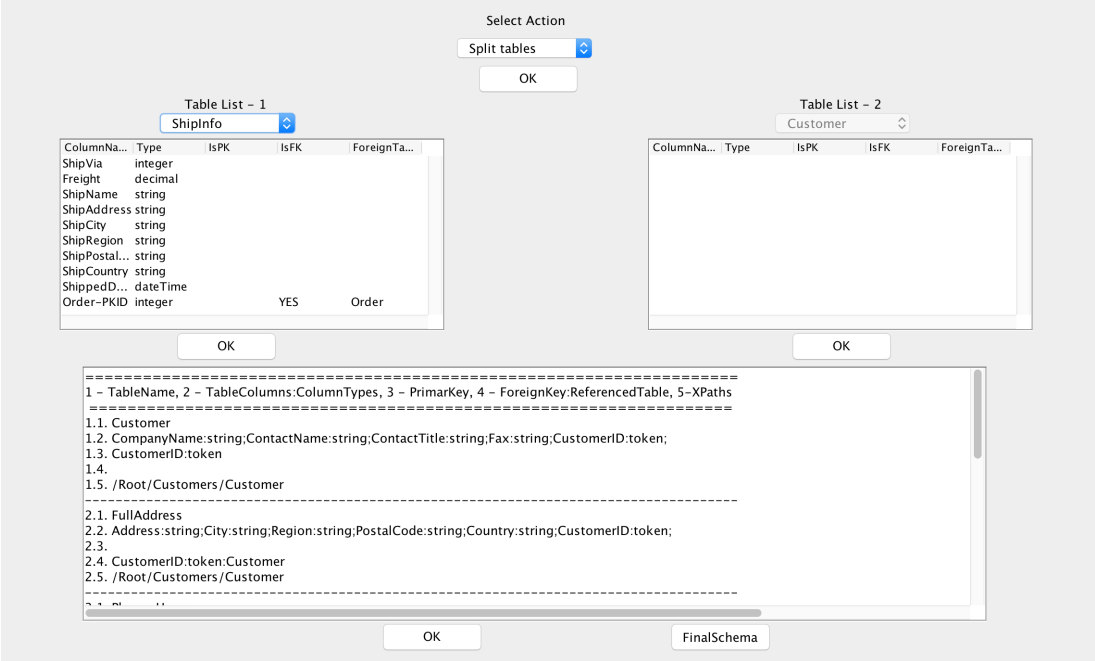

Figure 4.15: The user interface layout.

Chapter 5

System Analysis using TEI

XML Schema Definition files of XML and TEI documents with varying complexity levels are fed into our system to monitor its functionality. The following key points include the system effectiveness and limitations in converting these documents into relational data.

- The system successfully shreds input XSDs to suggest equivalent relational schema for all the input files that are relational in nature. Here, relational nature corresponds to hierarchical relation among the elements and supporting identity constraint information of the input schema documents. Consider the figure 5.1, it represents XML Schema Definition file of TEI document, which is relational in nature and its equivalent relational schema suggested by the system in the figure 5.3.

- It parses all the XML Schema Definition files containing tags like <complexType>, <sequence>, <choice>, <any>, <key>, <keyref>, <simpleContent>and other common tags like <element>, <attribute>. It fails to capture information from the input schema files if they include advanced

```
<xs:element name='text'>
 <xs:complexType>
  <xs:sequence>
   <xs:element name='listWit'>
    <xs:complexType>
     <xs:sequence>
      <xs:element name='listwit' type='listwitType' minOccurs='0' maxOccurs='unbounded' />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name='listBibl'>
    <xs:complexType>
     <xs:sequence>
      <xs:element name='listbibl' type='listbiblType' minOccurs='0' maxOccurs='unbounded' />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:key name='listBiblKey'>
  <xs:selector xpath='listBibl/listbibl'/>
  <xs:field xpath='type'/>
</xs:key>
 <xs:key name='listWitKey'>
  <xs:selector xpath='listWit/listwit'/>
  <xs:field xpath='head'/>
</xs:key>
</xs:element>
<xs:complexType name='listwitType'>
 <xs:sequence>
   <xs:element name='head' type='xs:string'/>
   <xs:element name='witness' type='witnessType'/>
 </xs:sequence>
 <xs:attribute name='xmlid' type='xs:string'/>
</xs:complexType>
<xs:complexType name='witnessType'>
 <xs:simpleContent>
   <xs:extension base='xs:string'>
   <xs:attribute name='xmlid' type='xs:string'/>
   </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name='listBiblType'>
 <xs:sequence>
   <xs:element name='biblId' type='xs:string'/>
   <xs:element name='bibl' type='biblType'/>
 </xs:sequence>
 <xs:attribute name='type' type='xs:string'/>
</xs:complexType>
<xs:complexType name='biblType'>
 <xs:sequence>
   <xs:element name='editor' type='xs:string'/>
   <xs:element name='pubPlace' type='xs:string'/>
   <xs:element name='publisher' type='xs:string'/>
   <xs:element name='date' type='xs:string'/>
 </xs:sequence>
 <xs:attribute name='xmlid' type='xs:string'/>
</xs:complexType>
```

Figure 5.1: An XML Schema Definition file defining a TEI structure.

element components like <restriction>, <fixed>, <default> tags.

- It utilizes the key information from input schema document to introduce the primary and foreign keys in the output suggested relational schema. Keys are essential for a good relational data model because they ensure row-level accessibility by allowing each record in a table to be precisely identified. Consider the figure 5.1, the key information of this XSD file includes two keys, <head> and <type>. Our system parses this information and appends these columns as the primary keys of the tables <listwit> and <listbibl> in the relational schema, the same is shown in the figure 5.3. Our system is tightly coupled with the identity constraints of the input XML Schema document to suggest relational schema and certain schema operations are also dependent on the input key information.

- Our system preserves and transforms the hierarchical nature of XML documents into relational nature by inheriting the primary key of the parent element table as foreign key of the child element table referring to the parent table. The system enables this inheritance during the suggested schema phase of mapping. The same is illustrated in the figure 5.3 the element <type> is the primary key of the table <listbibl> and foreign key of the table <bibl>.

- Element <witness> in the figure 5.2 is a recurring simple element with "xs:string" type. This implies that the simple element <witness> occurs multiple times within its parent element <listwit>, this is one of the prominent feature supported by the XML documents. If the system maps this column as part of the table <listwit> then the concept of primary key fails because after populating tables one can observe that same primary

```
<xs:complexType name='listwitType'>
 <xs:sequence>
   <xs:element name='head' type='xs:string'/>
   <xs:element name='witness' type='xs:string'
         minOccurs='1' maxOccurs='unbounded' />
 </xs:sequence>
 <xs:attribute name='xmlid' type='xs:string'/>
</xs:complexType>
```

Figure 5.2: Sample TEI XSD code with recurring simple element.

key value occurs multiple times as the table has multiple <witness>values against the same primary key value <head> in the example. To address this difference in nature between the XML documents and Relational Databases our system introduces new table for simple elements with multiple occurrences along with the primary key of parent table as the foreign key of the new table.

- Due to the heterogeneous nature of XML and Relational Databases, there might exist other possible relational schemas than the one suggested by the mapping technique. The system provides users provision to alter the suggested schema by performing certain operations mentioned in the section 4.2, providing flexibility to obtain the appropriate relational schema for the input XML Schema document. All these operations involve simple decision-making and easy to perform, making our XML to relational system a semi-automated tool as it automatically suggests relational schema for the input XML and TEI documents and then allows human intervention.

- Even though all the schema operations are easy to perform, users are expected to have a minimal knowledge regarding the concepts of relational tables for decision-making. For example, consider TEI or XML experts, as

they have a profound knowledge of concepts of hierarchical nature and the relationship among the elements, decision making becomes quite simple.

- Once a user finalizes schema, the system prompts for an XML document and this document is expected to be valid with respect to the input XML schema document. The system does not include an XML schema validator to check the validity of the input XML document. It throws an error and leaves the tables unpopulated if the input XML document is not a valid document.

```
=====================================================================
1 - TableName, 2 - TableColumns:ColumnTypes, 3 - PrimarKey, 4 - ForeignKey:ReferencedTable, 5- XPaths
=====================================================================
1.1. listwit
1.2. head:string;xmlid:string;
1.3. head:string
1.4.
1.5. /text/listWit/listwit
---------------------------------------------------------------------------------------------------------------------------------
2.1. witness
2.2. witness:string;xmlid:string;head:string;
2.3.
2.4. head:string:listwit
2.5. /text/listWit/listwit
---------------------------------------------------------------------------------------------------------------------------------
3.1. listbibl
3.2. biblId:string;type:string;
3.3. type:string
3.4.
3.5. /text/listBibl/listbibl
---------------------------------------------------------------------------------------------------------------------------------
4.1. bibl
4.2. editor:string;pubPlace:string;publisher:string;date:string;xmlid:string;type:string;
4.3.
4.4. type:string:listBibl
4.5. /text/listBibl/listbibl
---------------------------------------------------------------------------------------------------------------------------------
```

Figure 5.3: Suggested relational schema of the TEI XSD shown in 5.1.

Chapter 6

Conclusion

XML is a major standard for transmitting information over the Internet. Relational databases are a ubiquitous data storage and processing model. Transforming data between these two models is a key research area. This Thesis introduces a flexible, schema-aware mapping technique for converting XML data into an appropriate relational models by combining automatic and manual schema design. Our system generates relational data models for input TEI critical texts which are input to the the Improvise visualization environment. This approach helps in storing, querying, and analyzing input XML data and assists scholars to analyze TEI data and draw conclusions.

The first half of this thesis document introduced the motivation for this project, challenges in the conversion process, a summary of associated technologies, and a summary of current approaches to the problem. Advantages and disadvantages of several existing mapping techniques, and some of their prominent features incorporated into our system, were described. By using existing approaches, we conclude that a schema-aware mapping technique an most appropriate and effective method of conversion. Having a supporting schema document

describing XML as input is essential; XML Schema Definition documents are a suitable choice among the several schema supporting documents.

In the second half discusses the design and implementation of the mapping technique followed by the key components of the system. Introduces the principal strategies of our mapping techniques to develop suggested schema, the set of actions users can perform to obtain an appropriate schema and the system analysis.

The contributions of this thesis are as follows:

- A comprehensive data translation system shreds input XML documents into an equivalent relational database model using a schema-aware mapping technique.
- Unlike existing systems, our system allows users to choose the best suitable schema by performing simple operations.
- To the best of our knowledge, this system is the first one to consider identity constraints of the input XSD file in suggesting the relational schema.
- The system design is modular and easily accommodates changes and extensions to the system.
- By providing an XML Schema Definition file, a Text Encoding Initiative file can also be converted into its corresponding relational model.
- The system has the ability to accomplish the mapping of complex input XML files into relational databases quite efficiently.

The current system reflects an attempt to design an efficient XML to relational mapping system with simple human intervention. Work in the TEI, XML, and Relational Database areas are so broad and diverse that there is a lot of scope for introducing new features into our system. Special consideration was taken during the design of the system to make it extensible for adding future features. A few of the potentially interesting extensions for our project include:

- Introducing new schema operations beyond the existing operations, such as adding primary keys, adding foreign keys, ordering columns, and rename columns.

- The current user interface is simple and can be augmented with new features, such as a visual tool to display the XML trees of selected tables.

- Allowing users to perform data wrangling operations, such as to split existing column values using separators.

- Generate XML and XSD documents from the populated relational tables; in other words, an efficient mapping technique to transform Relational data back to XML. This is much simpler direction to process.

- Capability to shred input XML Schema documents involving other elements or tags that are not taken into consideration in the current design, such as "default", "fixed" , "restriction".

- Analyze the performance of the system and compare it with existing, fully automated approaches.

Bibliography

[1] The digital latin library. http://digitallatin.org/.

[2] Document object model (dom). https://docs.oracle.com/javase/tutorial/jaxp/dom/index.html.

[3] Extensible markup language (xml) 1.0 (fifth edition). https://www.w3.org/TR/2008/REC-xml-20081126/.

[4] Java package swing. https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[5] Text encoding initiative. http://www.tei-c.org/.

[6] Xml path language (xpath). https://www.w3.org/TR/xpath/.

[7] Xml schema object model (xsom) java library. https://xsom.java.net/.

[8] S. Amer-Yahia, F. Du, and J. Freire. Shrex: Getting started. http://shrex.sourceforge.net/howto/howto.html.

[9] S. Amer-Yahia, F. Du, and J. Freire. A comprehensive solution to the xml-to-relational mapping problem. In *Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, WIDM '04, pages 31–38, New York, NY, USA, 2004. ACM.

[10] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From xml schema to relations: a cost-based approach to xml storage. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 64–75, 2002.

[11] J. Bosak, T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, and J. Clark. Guide to the w3c xml specification ("xmlspec") dtd, version 2.1. https://www.w3.org/XML/1998/06/xmlspec-report.htm.

[12] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. Xperanto: Publishing object-relational data as xml. In *In WebDB*, pages 105–110, 2000.

[13] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with stored. *SIGMOD Rec.*, 28(2):431–442, June 1999.

[14] D. Florescu and D. Kossmann. Storing and querying xml data using an rdbms. *Bulletin of the Technical Committee on Data Engineering*, 22(3):27–34, September 1999.

[15] X. S. W. Group. W3c xml schema definition language. https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.

[16] G. Kappel, E. Kapsammer, and W. Retschitzegger. Integrating xml and relational database systems. *World Wide Web*, 7(4):343–384, Dec. 2004.

[17] Z. M. Kyu and T. T. S. Nyunt. Storing dtd-independent xml data in relational database. In *Industrial Electronics Applications, 2009. ISIEA 2009. IEEE Symposium on*, volume 1, pages 197–202, Oct 2009.

[18] D. Lee and W. W. Chu. Cpi: Constraints-preserving inlining algorithm for mapping xml dtd to relational schema. *J. Data and Knowledge Engineering (DKE)*, 39:3–25, 2001.

[19] D. Lee, M. Mani, F. Chiu, and W. W. Chu. Nesting-based relational-to-xml schema translation. In *In Intl Workshop on the Web and Databases (WebDB*, pages 61–66, 2001.

[20] D. Lee, M. Mani, and W. W. Chu. Effective schema conversions between xml and relational models. In *IN EUROPEAN CONF. ON ARTIFICIAL INTELLIGENCE (ECAI), KNOWLEDGE TRANSFORMATION WORKSHOP (ECAI-OT*, pages 3–11, 2002.

[21] S. Lu, Y. Sun, M. Atay, and F. Fotouhi. A new inlining algorithm for mapping xml dtds to relational schemas. *Lecture Notes in Computer Science*, 2814:366–377, 2003.

[22] M. Mani and D. Lee. Xml to relational conversion using theory of regular tree grammars. In *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers*, pages 81–103, London, UK, UK, 2003. Springer-Verlag.

[23] A. Schmidt, M. L. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of xml documents. In *Selected Papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*, pages 137–150, London, UK, UK, 2001. Springer-Verlag.

[24] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[25] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered xml using a relational database system. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 204–215, New York, NY, USA, 2002. ACM.

[26] C. Weaver. Improvise. http://www.cs.ou.edu/ weaver/improvise/.

[27] C. Weaver. Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 159–166. IEEE Computer Society, October 2004.

[28] P. T. Wood. Optimising web queries using document type definitions. In *Proceedings of the 2Nd International Workshop on Web Information and Data Management*, WIDM '99, pages 28–32, New York, NY, USA, 1999. ACM.