

Converting Relational Database to XML Schema and Vice Versa Using ContextMap

Sanaz Rahmati

**A Thesis in the
Department of Computer Science and Software Engineering**

**Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science**

**Concordia University
Montreal, Quebec, Canada**

February 2006

© Sanaz Rahmati, 2006



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-14334-7

Our file Notre référence

ISBN: 0-494-14334-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

Converting Relational Database to XML Schema and Vice Versa Using ContextMap

Sanaz Rahmati

An important issue in modern information systems and e-commerce applications is providing support for inter-operability of independent data sources. A broad variety of data is available on the web in distinct heterogeneous sources, stored under different formats: database formats (e.g., relational model), semi-structured formats (e.g., DTD, SGML or XSD schema), scientific formats, etc. Integration of such types of data is an increasingly important problem. Nonetheless, the effort involved in such integration, in practice, is considerable: conversion of database schemas from one format to another requires writing and managing complex data transformation programs or queries.

Realizing the importance of converting structured schemas to semi-structured schemas and vice versa, we have developed a method that successfully performed the conversion. Our approach benefits highly from the reverse engineering of structured or semi-structured schema into a ContextMap representation and leads to both identifying and understanding all components of an existing database system and the relationships among them.

We have successfully developed a prototype, called CODAX, for the schema conversion process. While more sophisticated techniques are required in this context, we believe the ideas proposed in this work lend themselves to useful analysis and tracing tools for schema conversion.

Acknowledgment

I am deeply grateful to my supervisors, Professor Wojciech M. Jaworski, and Professor Peter Grogono for their encouragement, support, valuable advice, generous attention, and constant help at Concordia University. I am greatly honored to study under their supervision.

My thanks also go to Halina Monkiewicz and Veronica Jacobo-Gutierrez for their support.

My special thanks are also due to the faculty and staff in the Computer Science Department at Concordia University, who provided me a good opportunity to learn and progress.

Finally, I would like to thank my parents and my husband for their support and encouragement in pursuing my studies.

Table of Contents

Introduction.....	1
1.1 A Motivating Example.....	4
1.2 Contributions of the Thesis.....	8
1.3 Thesis Outline	11
Background and Related Works	13
2.1 Converting Relational Database to XML Schema.....	18
2.2 Converting XML to Relational Schema	25
Using ContextMap Terminology as a Solution	30
3.1 ContextMap Terminology.....	31
3.2 High Level Modeling.....	33
3.3 Structural System and Data Representation.....	34
3.4 Syntax and Process	34
3.5 CONTEXT+ Tool	38
Converting Relational Database Schema to XML (XSD)	39
4.1 Mapping Relational Database to ContextMap Representation.....	39
4.1.1 Mapping Relational Database Schema to ContextMap Representation Algorithm.....	48
4.2 Converting ContextMap model to XML Schema.....	52
Converting XML Schema to Relational Database	61
5.1 Mapping XML to ContextMap Representation	62
5.1.1 Mapping XML Schema to ContextMap Representation Algorithm.....	69
5.2 Converting ContextMap model to Relational Database	71

A System Prototype	74
6.1 System Design	74
6.2 System Requirements	77
6.3 System Illustration	78
6.3.1 Converting Relational Database to XML Schema.....	79
6.3.2 Converting XML to Relational Database Schema.....	87
6.4 Architecture	93
6.5 CODAX Performance.....	96
Conclusions and Future Work	99
7.1 Conclusions.....	99
7.2 Future Work.....	101
References.....	102
Appendices.....	107
Appendix-A: Database Schema Examples	107
Appendix-B: CODAX Implementation Specification.....	117
Appendix-C: ContextMap Terminology and Notation.....	119
C-1: ContextMap Notation	119
C-2: 4P-able Capability.....	120
C-3: Context+ Functionalities.....	122

List of Figures

Figure 1-1 - NorthWind application relational database schema.....	5
Figure 1-2 - Fragments of the XML schema XSD for NorthWind.....	7
Figure 1-3 - ContextMap schema for relational database	7
Figure 1-4 - ContextMap schema for XML database	8
Figure 2-1 - Summary of reverse engineering research.....	16
Figure 2-2 - An example of an entity-relationship diagram	24
Figure 2-3 - Initial RID graph of the relational schema.....	25
Figure 3-1 - ContextMap model of ERD diagram and RID graph	36
Figure 4-1 - ContextMap schema of the NorthWind relational database schema	40
Figure 4-2 - ContextMap model of NorthWind relational database schema (1)	42
Figure 4-3 - Summary of relationship between database objects, set members, and set names	43
Figure 4-4 - ContextMap model of NorthWind relational database schema (2)	46
Figure 4-5 - ContextMap of NorthWind relational database schema (3)	47
Figure 4-6 - Fragments of the XML schema (XSD) for NorthWind.....	54
Figure 4-7 - Fragments of the XML schema (XSD) for NorthWind application for viewing tables and fields.....	56
Figure 4-8 - Fragments of the XML Schema (XSD) for NorthWind application for viewing field name and properties.....	57

Figure 4-9 - Fragments of the XML Schema (XSD) for NorthWind application for viewing table constraints.....	58
Figure 4-10 - Fragments of the XML schema (XSD) for NorthWind indicating the steps.....	60
Figure 5-1 - ContextMap schema for NorthWind XML schema.....	62
Figure 5-2 - ContextMap model of NorthWind XML schema (1)	64
Figure 5-3 - ContextMap model of NorthWind XML schema (2)	65
Figure 5-4 - ContextMap model of NorthWind XML schema (3)	66
Figure 5-5 - Summary of relationship between database objects, set members, and set names	67
Figure 5-6 - Part of NorthWind DDL example in Oracle	73
Figure 6-1 - CODAX modules.....	77
Figure 6-2 - CONTEXT+ form.....	79
Figure 6-3 - Dialog to choose source database	80
Figure 6-4 - Dialog to perform reverse engineering process	81
Figure 6-5 - ContextMap model of relational database schema	82
Figure 6-6 - Export ContextMap model to XML schema (1).....	83
Figure 6-7 - Export ContextMap model to XML schema (2).....	84
Figure 6-8 - Export ContextMap model to XML - choosing file name.....	85
Figure 6-9 - Generate script	86
Figure 6-10 - View generated XML schema script	87
Figure 6-11 - Dialog to perform reverse engineering process, XML=>CTX.....	88
Figure 6-12 - Dialog to perform reverse engineering process, select XML file.....	89

Figure 6-13 - Generated ContextMap schema from XML file	90
Figure 6-14 - Generate RDB schema from ContextMap	91
Figure 6-15 - Generate RDB schema from ContextMap, Select file name	92
Figure 6-16 - View generated RDB schema script	93
Figure 6-17 – Architecture of CODAX	95
Figure 6-18 - Process model of CODAX.....	96
Figure 6-19 - The comparison between running time and number of related objects	97
Figure A-1 - XML Schema for the NorthWind Database.....	113
Figure A-2 - Oracle relational schema for the NorthWind database	116
Figure A-3 - User interface and class specification	118
Figure A-4 - ContextMaps notation.....	120

Chapter 1

Introduction

Nowdays, the number of organizations which use the same subject of information and reside in various physical locations is increasing. Those organizations use a broad variety of data available in heterogeneous data sources.

Data itself is stored under different formats and types such as: structured (relational model), semi-structured (DTD, SGML or XSD schema), object oriented databases, scientific formats, etc. Information systems with heterogeneous data sources have a number of specifications such as: unified interface to the data and data schemas, more complex maintenance procedures as compared to an information system which contains just one data source type, and so on. In order to support interoperability of individual data sources and to eventually provide unified interface to the system, database schema conversion is an important issue. Transformation of such data sources to a unique model is an increasingly significant problem.

Since conversion of database schemas from one data source type to another one requires producing complex database conversion processes, the effort involved in such transformation, in practice is considerable.

Having a unique representation helps business analysts to deal better with different data source specifications. Moreover, having a well structured and accurate model

form different data sources helps the business owners, DBAs, and analysts to understand the structure of the data sources, and to apply the necessary changes accurately. Also, this unique model leads to performing efficient modifications on heterogeneous data sources, by decreasing time and cost.

Realizing the importance of conversion different data sources and integration into one unique format, we have developed a prototype that successfully performs the conversion from structured schemas to semi- structured schemas and vice versa using the ContextMap representation.

Our approach highly benefits from reverse engineering of structured or semi-structured schema into the ContextMap representation, and leads to identifying and understanding all components of an existing database system and the relationships between them.

Basic steps are identified in the process of converting relational databases into XML schemas and vice versa as follows:

At the first step, reverse engineering is employed to deduce information about functional dependencies, keys and inclusion dependencies. The process involves constructing the related ContextMap representation from an existing database based on the structured or semi-structured schema. Using the ContextMap model helps analysts and DBAs to understand database models and the interrelationships between different data sources.

At the second step, the obtained ContextMap representation is transformed into the target schema in a process known as forward engineering.

Our approach manages all type of relationships allowed in the ContextMap representation, including many-to-many and one-to-many relationships. Moreover, by using different types of constructed ContextMap schemas our approach manages two-way conversion. However, in some papers [WLAB04], [MLM01b], [LMCC01], [LMCC02], only one-way conversion is contributed.

Through the functions of ContextMaps, analysts and DBAs can integrate different database systems into a single domain and understand the business flow between different systems, instead of going through different database specifications. Using the merge facility in the CONTEXT+ environment to merge different database schemas gives DBAs the ability to retrieve characteristics of each table and its relationships [WMJK02], as does the use of mining functions. The abovementioned prerequisites can be divided into the following sub-functions.

- Construct a ContextMap Schema: Through the constructed ContextMap schema, a complex database system such as structured and semi structured, can be modeled, and implemented in a good manner, therefore the all relations between tables becomes traceable. Also this unique format helps analysts and DBAs to deal more efficiently with different databases.
- Query the ContextMaps: By querying the selected sets, the ContextMaps can explore the general and detail view of database schemas, analyze the relationship, complexity of relations, and the constraints of the database tables.

- Merge different database nodes by ContextMaps: Once the different database sources are modeled in ContextMap schema, then the multiple ContextMaps generated from different sources can be merged using CONTEXT+ functionality [WMJK02].

1.1 A Motivating Example

We use a simple NorthWind [NSLB] database example to illustrate both the problem and our solution approach. In our example, we have both XML and relational database schema definition for the NorthWind [NSLB] database.

The relational database, which contains *PRODUCTS*, *ORDERS*, *ORDER DETAILS*, *CUSTOMERS*, *CATEGORIES*, *SUPPLIERS*, *SHIPPERS*, and *EMPLOYEES* tables, is shown in figure 1-1.

Figure 1-2 illustrates a small part of XML data file for NorthWind database.

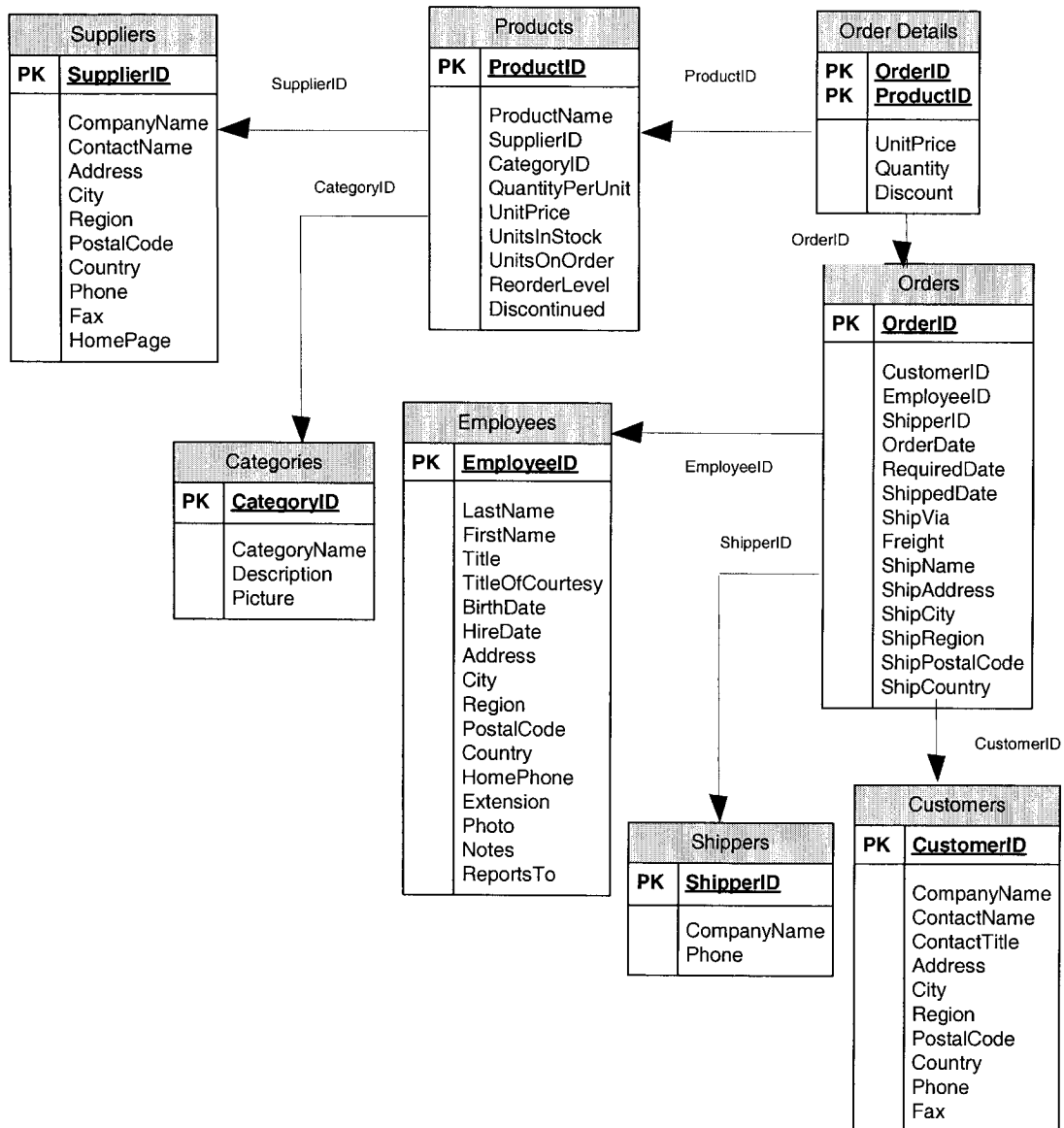


Figure 1-1 - NorthWind application relational database schema

Using this example we will illustrate the concept of converting a relational database to an XML schema and vice versa using the reverse engineering process, and show how the schemas will be mapped and represented in ContextMap unified notation. Details of schema mapping of figures 1-1 and 1-2 into ContextMap representation, and algorithms will be discussed in chapters 3 and 4.

...

```
<xs:element name="Categories">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CategoryID">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:maxInclusive value="2147483647"/>
            <xs:minInclusive value="-2147483648"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CategoryName">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="15"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Description" nillable="true">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="8000"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Picture" nillable="true">
        <xs:simpleType>
          <xs:restriction base="xs:base64Binary">
            <xs:maxLength value="2147483647"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="Products" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="PK_Categories">
    <xs:selector xpath="."/>
    <xs:field xpath="CategoryID"/>
  </xs:key>
  <xs:unique name="Categories_UniqueKey_0">
    <xs:selector xpath="."/>
    <xs:field xpath="CategoryName"/>
  </xs:unique>
  <xs:keyref name="FK_Products2" refer="PK_Categories">
```

• • •

Figure 1-2 - Fragments of the XML schema XSD for NorthWind

We store all the metadata information in the ContextMap environment. ContextMap is a canonical concept with notation for representing knowledge, which will be introduced in chapter 2. Figures 1-3 and 1-4 exhibit the ContextMap schema for relational database and XML database, where the ContextMap schema will be used as a conceptual model to store and map metadata information.

[illegible]

Figure 1-3 - ContextMap schema for relational database

schema models, ContextMap schema is constructed for both XML, and relational database schemas.

Let us discuss the reason to choose the ContextMap as a framework for mapping the schemas. There are many works on Schema Mapping and Converting Schemas through different methods such as reverse engineering using ERD (entity-relationship) diagrams, ERR (extended entity relationship) diagrams and so on [Alh03], [CFIL00], [MFKX00], [MLM01b], [WLAB04], [SD02].

ContextMap technology is useful in building more generic, effective and efficient maps from XML and relational databases, as it provides:

- High level system and data modeling as well as abstraction.
- Structural system and data representation.
- Formal and expandable syntax and notation.
- 4P-able (pattern-able, plug-able, process-able, perform-able) capability [WMJK02].

ContextMap technology has formal, flexible, expandable flexible syntax and notation, which allow efficient recovery and modeling of generic schemas for processes, objects and views in database systems.

Moreover having different database schemas transformed to the ContextMap model helps the analysts, and DBAs to deal only with one unique notation and understand the database schema design more accurately.

All of the above are challenging needs, which motivated us in the development of the concepts and algorithms in this research, where as listed as follows:

- a) We studied reverse engineering and schema mapping in the scope of ContextMap, and also proposed some novel solutions for reverse engineering and schema mapping, and identified issues to be resolved.
- b) We proposed a solution technique performing schema conversion in both ways- that is, from relational database to XML schema, and vice versa.
- c) For purposed of schema conversion, we proposed some algorithms to perform reverse engineering process and to map structured or semi-structured schemas to the ContextMap model. We also proposed some algorithms for forward engineering purpose.
- d) We studied and enriched the ContextMap representation by adding more notations and constructed different schemas for structured and semi-structured database schemas.
- e) Using ContextMap schemas, we transformed different database sources into ContextMap model; using merge functionality we can integrate different maps, which will eventually converted into different database schemas.
- f) We have successfully developed a prototype called CODAX, which stands for **C**onverting **R**elational **D**atabase to **X**ML, and vice versa,

to show the viability of the ideas and techniques proposed in this research.

1.3 Thesis Outline

The rest of this thesis is organized as follows. In chapter 2 we provide a background and review of related works on reverse engineering and schema mapping and conversion. This includes a description of different competing approaches for reverse engineering and schema mapping.

In chapter 3, we illustrate a brief introduction to ContextMap terminology and notation.

Chapters 4 and 5 include our main contribution in this research. In chapter 4, we illustrate the converting of relational database schema to XML, by introducing algorithms for the reverse engineering process to construct a ContextMap model from the existing relational database and for describing the ContextMap representation, and finally by introducing some functions and algorithms to perform transformation from ContextMap schema to XML. In chapter 5, we introduce different algorithms to transform XML schema to the ContextMap schema, which is constructed for XML, and finally in order to extract relational database schema from ContextMap model.

In chapter 6, we introduce our system prototype, and illustrate its features and capabilities through an example application. The architecture and performance is also illustrated in this chapter.

Chapter 7 includes concluding remarks and possible future directions.

The last part, Chapters 8, and 9 illustrates references and appendixes.

Chapter 2

Background and Related Works

Organizations are turning to system reengineering as a means of upgrading their existing information systems in situations where it appears to be a less expensive, and faster, alternative to system replacement [WLAB04].

Reverse engineering is viewed as a critical part of the whole system re-engineering process, because successful system re-engineering depends largely on effective reverse engineering. In general, reverse engineering can be defined as the process of discovering how a system works. It requires identifying and understanding all components of an existing system and the relationships between them. It helps to adjust or redo components of a system in order to improve its functionality and performance. The output from the reverse engineering process can also be reused as a source of enterprise architecture components. One such major component of an information system is the database metadata information. Consequently, database reverse engineering mainly deals with schema extraction, analysis and transformation, and it is necessary to semantically enrich and document a database, and also to avoid throwing away the huge amounts of data stored in existing relational

databases, if the owner of an existing database wants to re-engineer, or maintain and adjust the database design.

In the same way as for any other database reverse engineering process, must rely on a rich set of models. These models must be able to describe data structures at different levels of abstraction, ranging from physical to conceptual, and according to various modeling paradigms.

In addition, statically describing data structures is insufficient. We must be able to describe how one schema evolves into another one. For instance, a physical schema leads to a logical schema, which in turn can be translated into a conceptual schema.

Transitions, which form the basis of DBRE (Database Reverse Engineering), can be explained in a systematic way through the concept of schema transformation.

Over the past few years, researchers have produced several papers providing methods for transforming a relational database into a conceptual model [BSN92], [CBS94], [CBS96], [DA87], [J94], [MM90], [NA87], [PTB96], [PB94], [S98], [Alh03]. Each exhibits its own methodological characteristics, specific assumptions, inputs, and produces its own conceptual model. The following table illustrates the summary of reverse engineering research.

<i>Research</i>	<i>Assumption</i>	<i>Input</i>	<i>Output</i>	<i>Characteristics</i>
Batini <i>et al.</i> (1992). [BSN92]	<ul style="list-style-type: none"> • BCNF or 3NF. • Attribute naming consistency. • No homonyms. • Specific PK & Candidate Key. 	<ul style="list-style-type: none"> • Inclusion Dependencies. • Relation schemas. 	<ul style="list-style-type: none"> • Entities. • Binary relationships. 	Based on Navathe & Awong's paper, but further simplified. Drawback of requiring semantic input earlier in the process.

<i>Research</i>	<i>Assumption</i>	<i>Input</i>	<i>Output</i>	<i>Characteristics</i>
Chiang <i>et al.</i> (1996). [CBS96]	<ul style="list-style-type: none"> • 3NF. • Attribute-naming consistency. • No error on key attributes. 	<ul style="list-style-type: none"> • Relation schemas. • Data instances. • Inclusion dependencies. 	<ul style="list-style-type: none"> • Entities. • Binary relationship. • Generalization. • Aggregation. 	Requires knowledge about attribute name. Proposes a framework for the evaluation of DBRE methods. Clearly identifies the cases in which human input is required.
Davis & Arora (1987). [DA87]	<ul style="list-style-type: none"> • 3NF • No homonyms & synonyms. 	<ul style="list-style-type: none"> • Relation schemas. • Foreign key constraints. 	<ul style="list-style-type: none"> • Entity (sets). • Dangling keys. • Binary relationships. • <i>n</i>-ary relationships. 	Ignores inheritance. Aims at an invertible transformation from relational schema to conceptual schema.
Johannesson (1994). [J94]	<ul style="list-style-type: none"> • 3NF. • Domain-independent queries. 	<ul style="list-style-type: none"> • Relation schemas. • Functional dependencies. • Inclusion dependencies. 	<ul style="list-style-type: none"> • Generalization • Entities. • Binary relationships. 	Based on the well-established concepts of relational database theory. Drawback of needing all keys and inclusion dependencies. Simple and automatic mapping process.
Markowitz & Makowsky (1990). [MM90]	<ul style="list-style-type: none"> • BCNF. 	<ul style="list-style-type: none"> • Relation schemas. • Key dependencies. • Referential integrity dependencies. 	<ul style="list-style-type: none"> • Entity. • Binary relationships. • Generalization. • Aggregation. 	Presents theoretically-sound treatment of the mathematical basics. Requires all key functional dependencies and key-based inclusion dependencies.

<i>Research</i>	<i>Assumption</i>	<i>Input</i>	<i>Output</i>	<i>Characteristics</i>
Navathe & Awong (1987). [NA87]	<ul style="list-style-type: none"> • 3NF, or some 2NF. • Attribute-naming consistency. • No FK ambiguities. • Specified candidate keys. 	<ul style="list-style-type: none"> • Relation schemas. 	<ul style="list-style-type: none"> • Entity • Binary relationships. • Categories. • Cardinalities. 	<p>Drawback of requiring semantic input earlier in the process.</p> <p>Resolves the most common situations rather than claiming exhaustiveness.</p>
Petit <i>et al.</i> (1996). [PTB96]	<ul style="list-style-type: none"> • 1NF. • Unique attributes. 	<ul style="list-style-type: none"> • Relation schemas. • Data instance or code. 	<ul style="list-style-type: none"> • Entities. • Relationships. • Generalization. 	<p>Copes with demoralized relational schemas in DBRE process.</p> <p>Analyzes equi-join queries in application programs.</p> <p>No restriction on the naming of the attributes.</p>
Premelani & Blaha (1993, 1994). [PB94]	<ul style="list-style-type: none"> • Non-3NF. • Semantic understanding of application. 	<ul style="list-style-type: none"> • Relation schemas. • Observed patterns of data. 	<ul style="list-style-type: none"> • Class. • Association. • Generalization. • Multiplicity. • Aggregation. 	<p>Requires high level of human input.</p> <p>Provides guidelines for coping with design optimizations.</p> <p>Emphasizes analysis of candidate keys rather than primary keys.</p>
Soutou (1997, 1998). [S98]	<ul style="list-style-type: none"> • No attribute-naming uniqueness. • Unknown dependencies. 	<ul style="list-style-type: none"> • Data schema. • Data instance. • Data dictionary. 	<ul style="list-style-type: none"> • Cardinality. • Constraints on <i>n</i>-ary relationship. 	<p>Fully automates process for relational databases.</p>
Reda Alhajj (2002). [Alh03]	<ul style="list-style-type: none"> • 3NF 	<ul style="list-style-type: none"> • Data Schema. • Data Instance. • ERD. 	<ul style="list-style-type: none"> • RID graph. 	<p>Fully automates process using efficient DBRE agent.</p>

Figure 2-1 - Summary of reverse engineering research

Our research extends previous database reverse engineering research in figure 2-1 as summarized here:

Our approach utilizes information obtained from the output of multiple analytical processes, which describe how a conceptual schema can be retrieved from a structured or semi-structured database system. The aforementioned processes are a type of analysis of relational database and XML schema, and semantic understanding. The obtained information is used in reverse engineering algorithms.

While the majority of previous researches assumed the relations of the input database to be at least in 3NF, our research is based on the practical assumptions that there are no constraints on functional and inclusion dependencies or on attribute naming consistency or uniqueness. For modern relational databases where primary key and functional dependency constraints exist in the metadata (i.e., schema definition), these assumptions are not necessary.

In chapters 3 and 4, we describe our semantic metadata extraction rules and algorithms, and detail the implementation of our prototype. In this section we describe some of the related works, which have used database reverse engineering methods to convert structured schemas to (XML) semi-structured schemas and vice versa.

2.1 Converting Relational Database to XML Schema

The conversion from relational to XML has recently received significant attention. It is an important problem for the following reasons [WLAB04]:

- Users may want to publish their relational data as XML. Since XML has become a standard for information exchange, users want to provide an XML view of their underlying relational data. In this case, the relational schema is transformed into XML schema, and then exchanged with other web applications if necessary.
- It is more efficient to directly query on XML documents by using existing XML query language, rather than to translate XML query syntax to SQL and query on the relational database and then translate the query result back to XML. Therefore, providing an XML schema makes it easier for the end users or applications to access data.
- Web applications may want to exchange their data with other web applications. As XML has become a de-facto standard for information exchange, the relational data needs to be transformed into XML documents and then exchanged with other web applications. In this case, the relational schema is transformed into XML schema and relational data is likewise transformed into XML data.

The significance of this conversion problem has motivated several researchers; various approaches have been proposed to solve this problem.

In XPERANTO and Agora, users provide the relational schema and the queries against the XML view to the system. The XML schema is generated from these queries; they are also used to translate operations on the XML view to the relational data, as well as to translate results. In the approaches mentioned above, the successful conversion is closely related to the quality of the target XML schema onto which a given input relational schema is mapped. However, the mapping from the relational schema to the XML schema is done manually and by experts. Therefore, when large numbers of relational schemas need to be translated into XML documents, a large investment of human effort is initially required to design the target schemas.

The solution taken by XML Extender [CX00] from IBM provides users with the relational schema as input, as well as with the target XML schema. Users of this tool need to manually supply the mapping between the relational and the XML schema. The tool has the feature to convert operations from XML to relational and obtain the results as XML.

There are some other works, which map non-relational models to XML models. The work described in [MLM01b, FPB01, and KL01] studies the transformation from XML to EER model and vice versa, and also some works regarding generation of an XML schema from a UML model are presented in [BCFK99, C01]. Mani *et al.* [LMCC01, LMCC02] have investigated how to come up with an efficient XML schema. We will discuss them individually as follows:

- **Agora:** the Agora [MFKX00] system employs XML as the user interface format, while all data flows inside the query processor consist of relational tuples. Queries are first simplified by normalization, and then translated to

SQL. The SQL query is optimized and executed over the view that has been created from the XML documents. Tuples that form the result are tagged into XML elements, thereby producing the final XML result. This makes the underlying relational engine transparent to the user.

Agora uses a subset of the Quilt query language and stores XML into the relational database using the structure-oriented approach. Agora is implemented on top of the Le Select data integration system [LS01], developed in the Caravel project. The goal in designing Agora was to investigate the feasibility and the attainable performance of a system that processes XML queries based on relational technology.

- **XPERANTO:** The goal of the XPERANTO [CFIL00] project at the IBM Almaden Research Center is to serve as a middleware layer that supports the publishing of XML data. Clients and customers who would like to deal directly with XML data, rather than being forced to deal with the data source's particular (e.g. object-relational) schema and query language can use this tool. XPERANTO creates the XML view over the internal relational database, and provides an XML-based query interface, which translates XML queries into corresponding SQL queries. It then transforms the results back to XML. XPERANTO consists of the following components: XML-QL Parser, Query Rewriter, SQL Translator, and XML Tagger. For instance, XQuery is parsed and translated to an internal representation called XQGM. XQGM, which stands for XML Query Graph Model, used as an input for the SQL Translator that generates the query representation in SQL. The SQL query is executed in

the database and returns the answer to the XML Tagger, which creates the resulting XML document for the users.

In this approach, only basic features of object-relational systems are included for querying purposes. Integrity constraints and more advanced features are not considered in this approach.

- **ER to XML:** ER to XML [MLM01a] is a semantic approach using XML schema; the goal of this research is to formalize a core set of features found in various XML schema languages into X-Grammar (a grammar notation commonly found in formal language theory). The important building blocks of any XML schema language such as element-sub-element relationships can be captured in X-Grammar. Informally, XGrammar takes the structural specification feature from DTD, and the data typing feature from XML schema. The conversion rule is to generate XGrammar from a given XML model, then convert XGrammar to an EER model, or vice versa. A similar approach, described by Fong *et al.* [FPB01], uses a database reverse engineering approach. It constructs the semantic model in the form of an ERD model from the logical schema capturing user's knowledge, and then does forward engineering to produce the XML document. However, this approach [FPB01] only deals with catalog-based databases. Similarly, the work described in [KL01] offers a way to translate ERD schema into DTD, by introducing a set of rules on how to translate constructs from the ERD model into DTD. In this approach the authors claim that their translation preserves almost all semantic information.

- UML-to-XML:** The approach done by Booch *et al.* [BCFK99] utilizes a graphical notation in UML (Unified Modeling Language) for designing XML schemas. UML is a standard object-oriented design language. In this approach all the elements and data types in XML schema are mapped to classes annotated with stereotypes that reflect the semantics of the related XML schema concept. Also a sequence number is used for content model elements to indicate the order of document types. XML schemas may contain anonymous groups. In this approach, special stereotypes are introduced to indicate that the class represents an anonymous grouping of elements in UML. Similarly, the work done in [C01] describes an approach based on XMI rules for transforming UML to XML Schema. It also defines a UML profile, which addresses most XML schema concepts, except those simple content complex types, global elements and attributes, and identity constraints. Regarding semantic equivalence, the profile has some weaknesses in its representation of model groups, i.e., sequence, choice, and all.
- NeT and CoT:** The work described in [LMCC01, LMCC02] studies how to come up with an “efficient” XML schema. The approach represents three algorithms for converting a relational model to XML: FT (Flat Translation), NeT (Nesting-based Translation) and CoT (Constraint-based Translation). The native translation algorithm FT translates a “flat” relational model to a “flat” XML model in one-to-one manner. FT does not utilize the non-flat features of the XML model, which can be used through regular expression operators. In order to remedy this problem, NeT is presented; it derives nested structures

from a flat relational model by the use of the nest operator and generates a more precise and intuitive XML schema from relational inputs.

However, NeT is only applicable to a single table at a time, and cannot obtain the big picture of a relational schema where many tables are interconnected with each other. CoT addresses this problem; it uses semantic constraints to come up with a more intuitive XML schema for the entire relational schema.

Some testing has been done on NeT and CoT algorithms. According to the experimental results with DTDs, NeT and CoT can present better output in terms of both accuracy and size.

- **Publishing XML:** The work described by Shanmugasundaram et al. [SSB01] Provides a solution to effectively structure and tag data from one or more tables as a hierarchical XML document. In this approach the use of new scalar and aggregate functions in SQL for constructing complex XML documents directly in the relational engine is explored. The results of their experimental study show that constructing XML documents inside the relational engine can have a significant performance benefit. The XML publishing task is separated into three subtasks: 1. Data extraction. 2. Data structuring. 3. Data tagging.
- **COCALEREX** [WLAB04]: the work, which is described by Chunyan Wang, studies the conversion for both catalog-based and legacy relational databases. To convert legacy database into XML, the first step is to obtain all possible information to construct the ERD model, which can be done by applying the reverse engineering technique. In this approach, some algorithms [Alh03] are adopted to extract the ERD model from the given legacy relational database,

then convert the ERD to a RID graph, and finally generate the XML schema from the RID graph.

Figure 2-2 illustrates the example of the entity-relationship diagram, which will be converted to RID diagram in figure 2-3.

The main difference between this work and our work is that, instead of using ERD and the converting it to an RID and then to the target schema, which is XML, we perform the reverse engineering process in one phase and convert the relational database schema into ContextMap model, and transform the ContextMap model into to an XML schema. The ContextMap model gives the users the ability to manipulate schema information more easily, as compared to the RID graph.

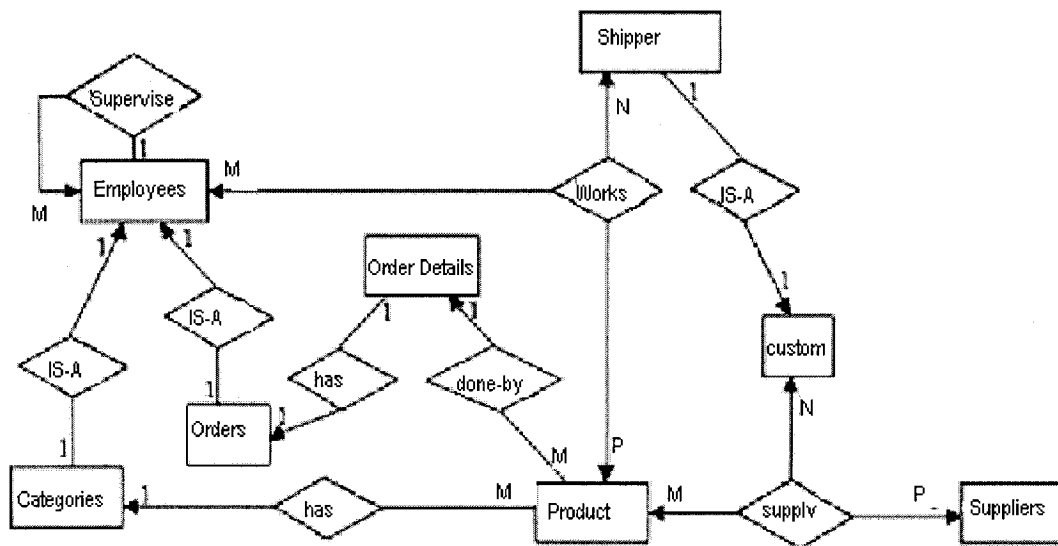


Figure 2-2 - An example of an entity-relationship diagram

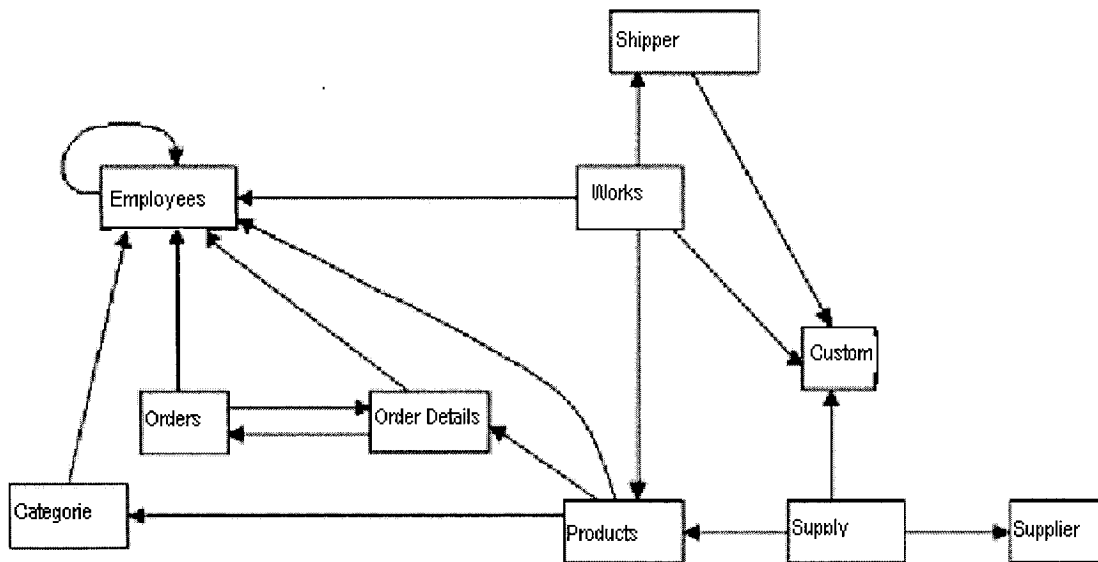


Figure 2-3 - Initial RID graph of the relational schema

Our approach is different from the others. We focus on the conversion of relational databases to XML using reverse engineering and forward engineering processes. To convert a relational database into XML schema, we first need to obtain all possible information to construct a ContextMap model, which is possible by applying the reverse engineering technique. In our approach, we adopt some algorithms to construct the ContextMap model from the given relational database and convert the ContextMap model to an XML schema. Also, using the ContextMap model as a conceptual model allows the users to deal with only one unique notation and thus understand the different database design and specifications in one unique format.

2.2 Converting XML to Relational Schema

Recently, more research has been addressed to the particular issues of the conversion from XML to relational schema. On the business side, database vendors are extending

their database products to support the XML type. Also research has been done in [SD02], [XR02], [DW00] to come up with an efficient relational database.

In this section, we present an overview of the XML to Relational schema mapping currently available from two selected commercial Database Management Systems: IBM DB/2 and Microsoft SQL Server.

- **MXM and IMXM** [SD02]: designed by Sihem Amer-Yahia and Divesh Srivastava, MXM and IMXM are a mapping schema and an interface API to define and query XML-to-Relational mappings.

A mapping is expressed as an instance of MXM. MXM is declarative, concise and captures existing XML-to-Relational mappings. Mappings can be expressed for documents for which no schema information is provided, and for documents that conform to either a DTD or an XML Schema. IMXM is an interface that allows querying of information contained in a MXM mapping. MXM is extensible and can incorporate new XML-to-Relational mappings. IMXM is implemented on top of this repository and, is used for generating a relational schema and loading XML documents into the corresponding relational database. The main weakness of this approach is that the generated schema is not accurate.

- **Rainbow** system [XR02]: Rainbow, which is described by X. Zhang, creates a middle layer between XML and relational databases for loading and extracting of XML documents and mapping XML schemas to relational schemas. In this system, XQuery is used to specify a mapping. While this language is powerful

enough to express existing mappings, its specification would be much longer and it is not clear how to use it for this purpose. Also, no interface API is provided.

- **ER 2000** [DW00]: The work presented in the paper describes XML-to-Relational transformations that preserve constraints on DTDs. Our mapping schema proposal also handles constraints.
- **IBM DB/2** [CX00]: IBM DB/2 is equipped with the XML Extender tool for storing XML documents. DAD (Document Access Definition) is used for XML-to-Relational schema mapping. DAD uses the following elements for mapping, and to describe the structure of XML documents:
 - *Element node*, which specifies the elements in an XML document.
 - *Attribute node*, which specifies the attributes.

RDB (Relational Database Node) node is another subelement, which is used to describe its mapping to the relational schema. Both *element node* and *attribute node* have *RDB node* as their subelement. *RDB node* has the subelements *table*, *column*, and *condition*, which specify a table, a column, and a relationship between tables in a relational schema. In the DAD mapping, XML elements are mapped to relational tables or columns, XML attributes are mapped to database columns, and relationships between XML elements are mapped to relationships between relational tables. The principles for this mapping are as follows:

1. Identify the structure of the XML documents using *element node* and *attribute node*.
2. Identify the mapping to a relational schema using *RDB node*.

3. Identify the relationships between tables in the *RDB node* sub-element of the root *element node*.

4. Identify the table and column, to which an element or an attribute is mapped, in the *RDB node* sub-element of each non-root *element node* and *attribute node*.

- **Microsoft SQL Server:** Microsoft SQL Server introduces the XML Bulk Load utility for storing XML documents. The XML-to-Relational mapping uses XDR (XML-Data Reduced) schema. XDR uses four elements, such as *ElementType*, *AttributeType*, *element*, and *attribute*, to specify the structure of XML documents, and two attributes, *relation* and *field*, as well as one element, *relationship* to specify the mapping to a relational schema.

In order to declare XML elements and attributes, *ElementType* and *AttributeType* are used. In addition, the attributes *relation* and *field*, respectively, specify a table and a column, and the element *relationship* specifies the relationship between two tables in the relational schema. Like the DAD, the XDR schema maps XML elements and attributes to relational tables or columns, and relationships between XML elements to relationships between relational tables. The steps specifying the mapping are almost identical to that of DAD.

1. Specify the structure of XML documents using the elements *ElementType*, *AttributeType*, *element*, and *attribute*.

2. Specify the table and column to which an element or an attribute is mapped, using the attributes *relation* and *field*.

3. Specify the relationship between tables using the element *relationship*.

Our approach to this problem is different from the existing approaches described earlier. IBM DB/2 and Microsoft SQL Server, require users to use a new language, such as DAD, XDR, and RXL. Our conversion from XML to relational schema handles all constraint mappings, and is straightforward in comparison to all works mentioned above. In our approach, we have developed algorithms to perform reverse engineering and produce the ContextMap model, which is used for XML schema, and to convert the ContextMap model to relational schema.

Chapter 3

Using ContextMap Terminology as a Solution

A methodology is needed, both to represent relational (structured) and XML (semi structured) database schemas, and also to guide the business analysts and database administrators in the whole design process.

ContextMaps originally called jMap (jointed map), was first introduced by Dr. W.M. Jaworski [WMJ99]. The technology was initially developed for recovering and refining knowledge from legacy systems. By using the concept of a spreadsheet structure, it is feasible to describe and process conceptual information. Different information can be easily integrated into one consistent map using ContextMap notation, thus it can be considered to be a kind of high-level notation technology. The use of ContextMaps notation allows efficient recovery and modeling of generic schemas. This technology can be applied in many domains, such as modeling, mining, evaluation of contexts, enterprises, methods, processes, projects, artifacts, databases, websites, information system, knowledge models with generic templates, domain experts, and proprietary notational technology.

ContextMap methodology has formal, flexible, expandable syntax and notations that can efficiently recover and model generic schemas for processes, objects, and views in these systems.

The basic element in ContextMaps is the context tuple, which is a generic association of set members cast in roles. A ContextMap could consist of an unlimited number of context tuples.

Although ContextMap could be implemented, deployed and used in a number of different manners in computer program, database, and modeling methodologies etc., For the sake of simplified visualization, it is generally represented as an extended spreadsheet.

3.1 ContextMap Terminology

The ContextMap terms used in this thesis include definitions, acronyms and abbreviations [WMJK02].

Figure A-4, ContextMap notation table, will be used to explain the ContextMap terminology.

- **ContextMaps:** represent the relationship between different information sets and provide the functionality of arrays, graphs, relational tables, etc.
- **JMaps:** Abbreviation of Jointed map. The previous name of ContextMap.
- **CONTEXT+:** A set of tools, which was developed for processing ContextMaps.

- **Context tuple:** A generic association of set members cast in roles. In the extended spreadsheet a column of roles and the related set members define context tuple.
- **KTuple:** Abbreviation of Knowledge Tuple. It consists of set, set member and Role Tuples, has its own Schema and contains Identifier, Type and Descriptors.
- **Schema:** Work frame of ContextMaps. The ContextMaps integrate concepts and concept instances with abstract architecture.
- **Set:** Sets are shown inside { } in the rightmost column.
- **Set Member:** The members below the bold { } in the rightmost column.
- **Set Roles:** the upper case letters in spreadsheet cells, such as letter “E”, “G”, and “L”.
- **Set Member Roles:** the lower case letters or digits in spreadsheet cells, such as “f”, “t”, and “l”.
- **Cardinality of Roles:** the number of non empty Roles in every row.
- **Cardinality of Set Member:** the number of set members under each set.
- **Atom:** Anything in a rightmost column: set name, value under {Set}.

3.2 High Level Modeling

ContextMap is a formal representation method for information systems with a set of predefined formal notation. It consists of an unlimited number of context tuples; i.e., a generic association of set members cast in roles. In the extended spreadsheet, a column of roles and the related set members define context tuples. Graphically, a context tuple is represented by a compounded edge and the connected compounded nodes.

A directed edge object consists of tail object, middle object and head object. While context tuples represent system behaviors, processes, tasks, procedures and programs, the aggregation of the context tuples forms ContextMaps. The ContextMap allows modeling, mining and evaluating context, processes and views of information systems with generic templates and by domain experts.

For example, the size of each structured or semi structured database schema script may vary from a few lines to thousands of lines. However, no matter how little or how big the script is, and how many database object the database schema consists of, it could be high level modeled or abstracted into the above mentioned concept sets which have been illustrated in figure 1-3 and figure 1-4.

3.3 Structural System and Data Representation

Not only can ContextMap represent individual data, it can also represent the structural information among those data, as well as relations and operations on data. For example, for a computer program, the operation – control flow or processes are implicitly hidden by syntax and notation of programming language or semantic model itself. If a user is not quite familiar with a certain language or the semantic model, it is very hard for him to understand the relation and operations on those data.

However, if the user deals with ContextMap representation, understanding the program or semantic model becomes quite easy and straightforward.

3.4 Syntax and Process

ContextMap technology uses formal and expandable syntax and notations, i.e. the syntax of ContextMaps is based on the Relationship-Oriented paradigm, defined by relating sets and set members [WMJK02]. In ContextMaps, the relationships are represented by kTuples (i.e. vertical columns in the map). The kTuple consists of set, set member and role tuples. This construction is the fundamental structure defined by the concepts and instances related by roles.

The relating mechanism is implemented by allocating roles to sets in schema and their instance to set components in the map. Compared to diagrams, maps are very

compact, and offer a rich context within limited space of a computer screen. Maps are created or edited within an organized electronic sheet (e.g. MS Excel spreadsheet) that assures efficient manipulation of relationships (columns) and heavy reuse of components (rows).

ContextMap is a graphical technology and notation for specifying, visualizing and modeling generic schemas with information systems. ContextMap notation is an essential element in this technology [WMJ95]. It can be widely employed in many fields such as:

- Information system architecture.
- Recovery and reuse of system patterns.
- Evolving information systems.
- Software evaluation and renewal.
- Automation of system design.
- Modeling of web sites and knowledge hubs.
- Systems workstations.

ContextMaps notation is illustrated in figure A-4. In practice, some symbols can be used for different meanings, and in special cases, users can define new symbols for themselves. The defining regulations are flexible, and easy to understand. In order to more clearly and in detail explain ContextMap notation, figure 3-1 illustrates how the entity-relationship diagram (figure 2-2) and the RID graph (figure 2-3) would be represented.

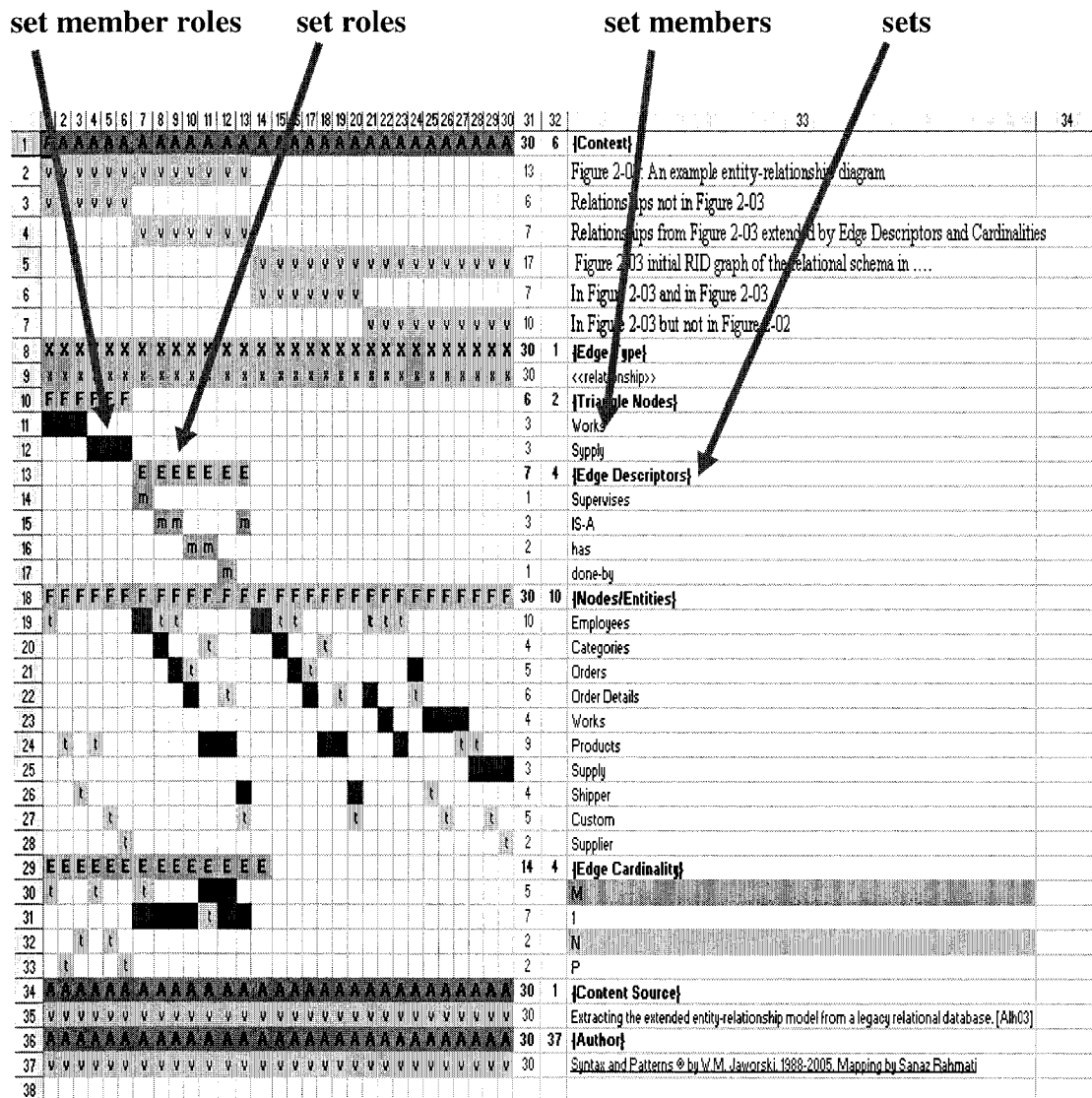


Figure 3-1 - ContextMap model of ERD diagram and RID graph

1. We define these sets {Context}, {Edge Type}, {Triangle Nodes}, {Edge Descriptors}, {Nodes/Entities}, and {Edge Cardinality} for representing the “ERD Diagram and RID graph” shown in figure 3-1. In this figure, the red arrow indicates the set {Edge Descriptors}.
2. We pick the correct notation based on the basic elements of ContextMap

notations in figure A-4 such as nodes, edges and flows. In this example, these schemas “A”, “X”, “E”, and “F” are used.

3. We expand each set by filling in its members. In figure 3-1 members under set {Edge Descriptors} are indicated by the red arrow.

In the above ContextMap, figure 3-1, nodes can be represented as “Sets” or “Components”. The following describes the syntax of the ContextMaps:

- The bold { } is called *set*, such as {Context} and {Edge Type}
- The elements under the *set* is called *set member*, such as “Supervises” and “IS-A”, which is indicated in figure 3-1.
- The contents in columns (1-30), shown in figure 3-1, are called *Context Tuples*.
- The single uppercase capital letters in *Context Tuples* are *set roles*, such as the letter A, X, E, and F which is indicated in figure 3-1.
- The lower case letters or digits in *Context Tuples* are *set member roles*, such as f, t, l, b, m and v, which are indicated in figure 3-1.
- The column 31 with numbers is the count of the *set member roles*.
- The column 32 with numbers is the count of the *set members*.

All information is kept in the ContextMap repository. With the CONTEXT+ tool, each ContextMap can be driven into a specific view and be traced back to the general view.

If users need to develop large ContextMap models, they can hide irrelevant columns and rows, editing visible cells and inserting new columns and new rows.

In general, sets appear on the right of the map between bold curly brackets. Each column has to be read vertically using the ContextMap syntax. For each “Value” in the spreadsheet, the user can read up or down a column and across towards the right of the map to find which *role* and *set member* the “Value” is referring to. Employing this, the user can also obtain the schema of ContextMaps by hiding *set members* and irrelevant columns and get useful information by applying the query tool.

3.5 CONTEXT+ Tool

The CONTEXT+ tool provides users with four different functionalities (Mode, Query, Output, and Run). The tool can be used to:

- View different mappings, by using different query functions gives users the ability to process and analyze complex maps more easily.
- Merge different maps generated from different domains.
- Apply specific colors for noticeable display.
- Calculate the cardinality of each set and set member, in order to analyze the complexity of each set and its member.
- Use schema functionality, to understand the basic design elements.
- Generate structured and semi-structured database schema scripts.

Chapter 4

Converting Relational Database Schema to XML (XSD)

In this chapter, we consider how relational database can be converted into XML schema. The background and related works are already discussed in Section 2.1. In Section 4.1, we represent our approach and algorithms to extract ContextMap model from a relational database schema and in section 4.2, we go through the forward engineering approach for converting a ContextMap model to XML schema.

4.1 Mapping Relational Database to ContextMap Representation

In this section we present the proposed process for mapping a relational database into a ContextMap schema.

{Table} (Row 6: F) has number of {Field} (Row 32: N).

{Field} (Row 32: N) has a specific {Type} (Row 88: F).

{Table} (Row 6: F) has a specific {Tablespace Type} (Row 85: N).

{Tablespace Type} (Row 85: N) has {Tablespaces} (Row 93: N).

{Table} has {Primary Key} constraints (Row 15: N).

{Table} has {Foreign Key} constraints (Row 24: N).

The detailed explanations of each part of the map are given in the following sections.

In the figure 4-2, each set members is shown in rightmost column under the set name.

In figure 4-3 we represent the relationship between database objects, set members, and set names in our context.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
2	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
3	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
4	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
5	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
6																									
7																									
8																									
9																									
10																									
11																									
12																									
13																									
14																									
15																									
16																									
17																									
18																									
19																									
20																									
21																									
22																									
23																									
24																									
25																									
26																									
27																									
28																									
29																									
30																									
31	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
32																									
33																									
34	t																								
35	t																								
36	t																								
37																									
38																									
39																									
40																									
41																									
42																									
43																									
44																									
45																									
46																									
47																									
48																									
49																									
50																									
51																									
52																									
53																									
54																									
55																									
56																									
57																									
58																									
59																									
60																									
61																									
62																									
63																									
64																									
65																									
66																									
67																									
68																									
69																									
70																									
71																									
72																									
73																									
74																									
75																									
76																									
77																									
78																									
79																									
80																									
81																									
82																									
83																									
84	F	F	F	F	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
85																									
86																									
87																									
88																									
89																									
90																									
91																									
92																									
93	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
94	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
95	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
96	#	12	#	12	21	#	#	#	#	#	#	#	14	14	14	#	#	#	#	200	10				
97	5	5	5	5	8	8	8	8	8	8	8	5	6	6	6	6	8	5	8	6	70	52			

Figure 4-2 - ContextMap model of NorthWind relational database schema (1)

<i>Database Object</i>	<i>Set Member</i>	<i>Set Name</i>
Table	<i>PRODUCTS, ORDERS, ORDER DETAILS, CUSTOMERS, CATEGORIES, SUPPLIERS, SHIPPERS, and EMPLOYEES</i>	{Table}
Table Field	<i>Product_ID, ProductName, Quantity, and ...</i>	{Field}
Table Field Type	<i>dbMemo, dbDate, dbText, and dbLong</i>	{Type}
Table Primary keys	<i>PK_Products, PK_OrderDetails, PK_Orders, PK_Customers, PK_Suppliers, PK_Categories, PK_Employees, and PK_Shippers</i>	{Primary Key Constraints}
Table Foreign keys	<i>FK_OrderDetails1, FK_OrderDetails2, FK_Orders1, FK_Orders2, FK_Orders3, FK_Products1, and FK_Products2</i>	{Foreign Key Constraints}
Tablespace types	<i>System, and User_ Data</i>	{Tablespace Types}
Tablespace parameters	<i>PCTFREE 10, PCTUSED 40, PCTUSED 50, and ...</i>	{Tablespaces}
Views	<i>Types, Tables, and References</i>	{View}

Figure 4-3 - Summary of relationship between database objects, set members, and set names

Back to figure 4-2, we use set member roles “v”, “f”, and “t” in order to associate tables to their fields, and fields to their types. Set member role “k” is used to associate primary key constraints with its related fields, and also set member roles “v”, “f”, “t”, and “b” is used to associate foreign key constraints in a binary table relationship in figure 4-4.

Moreover, in figure 4-5, set member role “f” is used to connect tables to its tablespace types and tablespaces.

For instance, in order to find a related field for the Orders table, it is first necessary to find set member Tables, under {View} set; moving horizontally, we will find the related set member roles “v” for the set member Tables. It means whatever columns indicated with “v” represent the table information area in the map.

Moving down vertically in this area (column: 5 till column: 12), we will have set member role “f” which indicates specific table name; in our example it is located at (column: 6 till row: 7) for Orders table.

If we traverse the map vertically again we will have set member role “t”, which indicates the related fields for Orders table. Note that some fields are marked with “k”, instead of “t”, which indicates primary key of the table.

The same method has to be applied to find the related fields, types, foreign keys, tablespace types, and tablespaces for Orders table. Therefore the related fields and their types for Orders table are: *OrderID (dbLong)*, *CustomerID (dbLong)*, *EmployeeID (dbLong)*, *OrderDate(dbDate)*, *RequiredDate (dbDate)*, *ShippedDate (dbDate)*, *ShipVia (dbLong)*, *Freight (dbLong)*, *ShipName (dbText)*, *ShipAddress*

(dbText), *ShipCity (dbText)*, *ShipRegion (dbText)*, *ShipPostalCode (dbText)*, *ShipCountry (dbText)*. The Primary Key Constraint is *PK_Orders*, which is linked to *OrderID* field. The Foreign Key Constraint for *Orders* table is *FK_Orders1* which links *Orders* table to *Customers* table through *CustomerID* field, and also *FK_Orders2* links *Orders* table to *Employees* table through *EmployeeID* field, finally *FK_Orders3* links *Orders* table to *Shippers* table through *ShipperID* field.

The Tablespace Type for related table is *User_Data*, with following Tablespace parameters *PCTFREE 10*, *PCTUSED 70*, *INITRANS 1*, and *MAXTRANS 355*.

[illegible]

Figure 4-4 - ContextMap model of NorthWind relational database schema (2)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	11	3	{View}			
2	V	V	V	V																4		Types			
3					V	V	V	V	V	V	V	V								8		Tables			
4													V	V	V	V	V	V	V	7		References			
5					F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	15	8	{Table}			
6																				3		OrderDetails			
7													t							5		Orders			
8														t						4		Products			
9															t					2		Customers			
10																			t	2		Suppliers			
11																				2		Categories			
12																t				2		Employees			
13																	t			2		Shippers			
14					N	N	N	N	N	N	N	N								8	8	{Primary Keys Constraints}			
23													E	E	E	E	E	E	E	7	7	{Foreign Keys Constraints}			
31	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	19	52	{Field}			
84					N	N	N	N	N	N	N	N								8	2	{Table Space Types}			
85																				2		System			
86																				2		User_Data			
87	F	F	F	F																4	4	{Type}			
92					N	N	N	N	N	N	N	N								8	10	{Table Spaces}			
93																				4		PCTFREE 10			
94																				1		PCTUSED 40			
95																				1		PCTUSED 50			
96																				1		PCTUSED 60			
97																				1		PCTUSED 70			
98																				4		INITRANS 1			
99																				1		MAXTRANS 155			
100																				1		MAXTRANS 200			
101																				1		MAXTRANS 255			
102																				1		MAXTRANS 355			
103	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	11	1	{AUTHOR}			
104	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	11		Syntax and Patterns © by W.M. Joworski, 1988-2001			
105	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V			Mapped by Sanaz Rahmati			
106	#	12	#	12	21	#	#	#	#	#	#	#	14	14	14	#	#	#	#	200	10				
107	5	5	5	5	8	8	8	8	8	8	5	6	6	6	6	8	5	8	6	70	52				

Figure 4-5 - ContextMap of NorthWind relational database schema (3)

4.1.1 Mapping Relational Database Schema to ContextMap

Representation Algorithm

In this section, we present the proposed reverse engineering process of mapping the relational database schema into ContextMap schema. The process is described in three algorithms. The first step in schema mapping is to construct ContextMap schema. For this purpose, a set of specific set roles $R = (R_1, \dots, R_m)$ and sets $S = (S_1, \dots, S_n)$ are used to build ContextMap schema from relational database schema. The algorithm below accepts specific sets and set roles as input, and generates ContextMap schema for relational database schema.

Input: S, R	<i>/* Sets and Set Roles */</i>
Output: RDB_WorkSheet	<i>/* Schema for RDB_WorkSheet */</i>
Algorithm:	<i>Construct ContextMap Schema for RDB Schema</i>
With a given worksheet	
For each S_i in S where i in $[1..n]$	
For each R_j in R where j in $[1..m]$	
Insert relational database sets (S_i) where S_i in ($\{Table\}, \{Field\}, \{Type\}, \{Primary Key Constraints\}, \{Foreign Key Constraints\}, \{View\}$).	
Insert the related R_j in front of each relational database sets (S_i).	
End.	

The second algorithm accepts available database tables in the system called T, represented as $T = (T_1, \dots, T_m)$ and the related fields $F = (F_1, \dots, F_o)$, having field types

TY= (TY₁, ...,TY_p), primary key constraints PK= (PK₁, ...,PK_q), and foreign key constraints FK= (FK₁,..., FK_r) and it inserts the information in the ContextMap representing relational database schema, as set members in its related set defined as S= (S₁, ...,S_n) sections.

<p>Input: S, T, F, TY, PK, FK <i>/* Sets and Database Objects */</i></p> <p>Output: RDB_WorkSheet <i>/* RDB_WorkSheet with inserted Set Members*/</i></p> <p>Algorithm: <i>Create Set Members from RDB Schema</i></p> <p>With the given sheet</p> <p>For each S_i in S where i in [1..n]</p> <p> If S_i = {Table}</p> <p> For each T_j in T where j in [1..m]</p> <p> Insert T_j as set member under (S_i) set.</p> <p> End If</p> <p> If S_i = {Field}</p> <p> For each F_j in F where j in [1..o]</p> <p> Insert F_j as set member under (S_i) set.</p> <p> End If</p> <p> If S_i = {Type}</p> <p> For each TY_j in TY where j in [1..p]</p> <p> Insert TY_j as set member under (S_i) set.</p> <p> End If</p> <p> If S_i = {Primary Key Constraints}</p> <p> For each PK_j in PK where j in [1..q]</p> <p> Insert PK_j as set member under (S_i) set.</p> <p> End If</p> <p> If S_i = {Foreign Key Constraints}</p> <p> For each FK_j in FK where j in [1..r]</p> <p> Insert FK_j as set member under (S_i) set.</p> <p> End If</p> <p> If S_i = {View}</p> <p> Insert set members "Types", "Tables", and "References" under (S_i) set.</p> <p> End If</p> <p>End.</p>

The algorithm below accepts available database tables in the system called T , represented as $T = (T_1, \dots, T_m)$ and the related fields $F = (F_1, \dots, F_o)$, field types $TY = (TY_1, \dots, TY_p)$, primary key constraints $PK = (PK_1, \dots, PK_q)$, and foreign key constraints FK noted as $FK = (FK_1, \dots, FK_r)$ and it traverses through the map and inserts the associations which are set member roles, between each set member, as a result it constructs the context tuple section in the map.

Input: T, F, TY, TR, PK, FK */* Set Members */*

Output: $RDB_WorkSheet$ */* Schema for $RDB_WorkSheet$ */*

Algorithm: *Construct Set Member Roles based on input Set Members, and Build Association*

With the given sheet

For each T_k in {table} set where k in $[1..m]$

For each F_j in F where j in $[1..o]$

Call InsertAssociation (T_k, F_j). */*Create the association between T_k and F_j */*

For each TY_x in {Type} set where x in $[1..p]$

Call InsertAssociation (F_j, TY_x). */*Set the association between F_j , and TY_x */*

For each PK_j in PK where j in $[1..q]$

Call InsertAssociation (T_k, PK_j). */*Set the association between T_k and PK_j */*

For each FK_j in FK where j in $[1..r]$

Call InsertAssociation (T_k, FK_j). */*Set the association between T_k and FK_j */*

End.

In order to map relational database schema to ContextMap schema by the above algorithms, we need to go through all three as detailed next. To do so we have chosen NorthWind [NSLB] database as an example to describe above algorithms.

The first algorithm is used to construct the ContextMap model from the relational database schema. It inserts sets such as: {View}, {Table}, {Primary keys Constraints}, and set roles corresponding to each set such as “V” for {view}, “F” for {Table}, and {Type}, “N” for {Primary Key Constraints}, {Field}, and {Table Space Type}, and {Table Space}, and “E” for {Foreign Key Constraints}.

In the second algorithm we perform the following steps as below:

1. Insert tables (*PRODUCTS, ORDERS, ORDER DETAILS, CUSTOMERS, CATEGORIES, SUPPLIERS, SHIPPERS, and EMPLOYEES*) as set members under {Table} set.
2. Insert the related fields as set members below {field} set.
3. Insert the related field types as set members below {type} set.
4. Insert the related primary key constraints as set members below {Primary key Constraints} set.
5. Insert the related foreign key constraints as set members below {Foreign key Constraints} set.
6. Insert set members “Types”, “Tables”, and “References” under {View} set.

In the third algorithm we do the following steps as below:

1. Associate each table to its fields by using “f” and “t” as set member roles.
2. Associate each field to its type by using “f” and “t” as set member roles.

3. Associate each Primary Constraint name to its corresponding table by using “f” and “k” set members, and also use “k” to connect the associated field to its Primary Constraint name.
4. We use “f” set member role to connect each Foreign Constraint name to its corresponding tables and we use, “f” and “t” set member to specify foreign key relationship between two tables, and we use “b” set member role to show the foreign key fields of the related tables.

4.2 Converting ContextMap model to XML Schema

In this section, we propose a forward engineering process for transferring the conceptual schema, which is presented as a ContextMap model, into an XML schema. The process in pseudocode is depicted in the algorithm below.

In order to transfer a ContextMap model to an XML schema using the algorithm below, we need to go through four steps as detailed next:

In the first step of the algorithm, each table T_i in form of $T = (T_1, \dots, T_n)$ in ContextMap is translated into an XML element E_i in form of $E = (E_1, \dots, E_n)$, having the same element name E_i in the XML schema. Under each element E_i , there will be several sub elements inside the empty element. For example, the *Orders* entity is translated into an element named “*Orders*”. The empty element is called `<complexType>`.

Input: RDB_WorkSheet */* RDB ContextMap Model */*

Output: XML Schema */* Generated XML Schema */*

Algorithm: *Generate XML Schema from ContextMap model of RDB Schema*

With the given sheet

For each T_i in S where i in $[1..n]$

- Translate each Table in the ContextMap model into a "complexType" element in XML schema (E_i).
- Find the association between each table and its fields, and Map each field in every table into a sub element within the corresponding element of the table.
- Find the association between each field and its type, and Map each field's type and length.
- Find the association between each table and its constraints, and Use "key" and "keyref" to map each relationship between two Tables

End.

Figure 4-6 illustrates the mapping of table name to XML schema:

```

...
<xs:element name="Categories">
  <xs:complexType>
    ...
  </xs:element>

<xs:element name="Customers">
  <xs:complexType>
    ...
  </xs:element>

<xs:element name="Employees">
  <xs:complexType>

```

</xs:element>

...

Figure 4-6 - Fragments of the XML schema (XSD) for NorthWind

The cardinality constraint in the ContextMap model can be explained by associating two built-in XML attributes, also called indicators, namely “minOccurs” and “maxOccurs”, with subelements under the XML <complexType>.

The “maxOccurs” indicator specifies the maximum number of times a sub-element can occur. “maxOccurs” = “unbounded” indicates the element may appear more than once. The “minOccurs” indicator specifies the minimum number of times a sub-element can occur. The default value for both the “maxOccurs” and the “minOccurs” attributes is 1. If we want to specify a value only for “minOccurs”, it must be either 0 or 1. Similarly, if we want to specify a value only for the “maxOccurs”, it should be greater than or equal to 1. If both “minOccurs” and “maxOccurs” are omitted, then the sub-element must appear exactly once.

In the second step of algorithm, we go through the ContextMap model to find the association between each attribute of the table T_i , and extract each attribute.

Each attribute of the table T_i is mapped into a sub element of the corresponding element T_i . For example inside the <complexType> of Orders table there are several sub elements such as *OrderID*, *CustomerID*, *EmployeeID*, *OrderDate*, *RequiredDate*, *ShippedDate*, *ShipVia*, *Freight*, *ShipName*, *ShipAddress*, *ShipCity*, *ShipRegion*, *ShipPostalCode*, And *ShipCountry*.

These are mapped inside of the <complexType> of Orders table.

The XML schema of the *Orders* table is shown in figure 4-7:

...

```
<xs:element name="Orders">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OrderID">
        ...
      </xs:element>
      <xs:element name="CustomerID" nillable="true">
        ...
      </xs:element>
      <xs:element name="EmployeeID" nillable="true">
        ...
      </xs:element>
      <xs:element name="OrderDate" nillable="true">
        ...
      </xs:element>
      <xs:element name="RequiredDate" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShippedDate" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipVia" nillable="true">
        ...
      </xs:element>
      <xs:element name="Freight" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipName" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipAddress" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipCity" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipRegion" nillable="true">
        ...
      </xs:element>
      <xs:element name="ShipPostalCode" nillable="true">
        ...
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



```

        <xs:element name="ShipCountry" nillable="true">
            ...
        </xs:element>
</xs:element>
...

```

Figure 4-7 - Fragments of the XML schema (XSD) for NorthWind application for viewing tables and fields

The `<sequence>` specification in the XML schema captures the sequential semantics of a set of sub elements. For instance, in the `<sequence>` given above, the sub element *OrderID* comes first, followed by *CustomerID*, and then *EmployeeID*, *OrderDate*, *RequiredDate*, *ShippedDate*, *ShipVia*, *Freight*, *ShipName*, *ShipAddress*, *ShipCity*, *ShipRegion*, *ShipPostalCode* with *ShipCountry* at the end.

These sub elements must appear in instance documents in the same sequential order as they are declared here. The XML schema also provides another constructor called `<all>` which allows elements to appear in any order, and all the elements must appear once or not at all.

In the third step of algorithm *Generating XML Schema from ContextMap RDB Schema*, we extract the related fields and their types for each table from the ContextMap model.

`<simpleType>` element is used in order to map the data type and the length of the each attribute.

For example, inside the `<complexType>` of Orders table there are several sub elements such as *ShipAddress*, *CustomerID*, etc. They are mapped inside of the

<simple Type> of Orders table, and their data type is shown by “xs:string”, “xs:integer”, and so on.

The partial XML schema of the *Orders* table’s data types is illustrated in figure 4-8:

```
....  
<xs:element name="Orders">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="OrderID">  
        <xs:simpleType>  
          <xs:restriction base="xs:integer">  
            <xs:maxInclusive value="2147483647"/>  
            <xs:minInclusive value="-2147483648"/>  
          </xs:restriction>  
        </xs:simpleType>  
      </xs:element>  
      <xs:element name="CustomerID" nillable="true">  
        <xs:simpleType>  
          <xs:restriction base="xs:string">  
            <xs:maxLength value="5"/>  
          </xs:restriction>  
        </xs:simpleType>  
      </xs:element>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>  
....  
</xs:element>
```

Figure 4-8 - Fragments of the XML Schema (XSD) for NorthWind application for
viewing field name and properties

In the fourth step of the algorithm, we traverse the map and find the association between each table and its primary key and foreign key constraints, and extract constraints of the table from the ContextMap model.

We use the elements “key”, “unique”, and “keyref” to enforce the uniqueness and referential constraints among the data. The “key” element specifies an attribute or element value as a primary key within the containing element in an instance document, and the “keyref” element specifies foreign keys, i.e., an attribute or element value corresponding to that already specified key or unique element. The “key” and “keyref” elements replace and extend the capability of “id”, “idref” and “idrefs” in DTD. They are among the valuable features introduced in XML schema. Also, we can use “key” and “keyref” to specify the uniqueness scope and multiple attributes to create the composite keys. Figure 4-9 illustrates an example:

```
<xs:element name="OrderDetails">
  <xs:complexType>
    ...
  </xs:complexType>
  <xs:key name="PK_OrderDetails">
    <xs:selector xpath="."/>
    <xs:field xpath="OrderID"/>
    <xs:field xpath="ProductID"/>
  </xs:key>
</xs:element>

<xs:element name="Orders">
  <xs:complexType>
    ...
  </xs:complexType>
  <xs:key name="PK_Orders">
    <xs:selector xpath="."/>
    <xs:field xpath="OrderID"/>
  </xs:key>
  <xs:keyref name="FK_OrderDetails1" refer="PK_Orders">
    <xs:selector xpath="OrderDetails"/>
    <xs:field xpath="OrderID"/>
  </xs:keyref>
</xs:element>
```

Figure 4-9 - Fragments of the XML Schema (XSD) for NorthWind application for
viewing table constraints

In figure 4-9, we first specify the primary key for each table in the ContextMap model. From the {Primary Key} set in ContextMap schema, we know that *OrderID* is the primary key of Orders table; *OrderID* and *ProductID* together form a composite primary key of *OrderDetails* table. From the {ForeignKey} set, *OrderID* is also a foreign key of *OrderDetails* table, so we use “keyref” to specify the foreign key relationship between *OrderDetails* and *Orders* table.

Compared to DTD, the XML schema provides a more flexible and powerful mechanism through “key” and “keyref” which share the same syntax as “unique” and also make referential constraints possible in XML documents [WLAB04].

Figure 4-10, illustrates the mapping and summarizes the steps of the above algorithms:

```

...
<xs:element name="Categories"> ← Mapping the table name
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CategoryID"> ← Mapping the attribute
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:maxInclusive value="2147483647"/>
            <xs:minInclusive value="-2147483648"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CategoryName"> ← Mapping the attribute
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="15"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  ...
  <xs:key name="PK_Categories"> ← Mapping the unique keys and the primary keys

```

```

    <xs:selector xpath="."/>
    <xs:field xpath="CategoryID"/>
  </xs:key>
  <xs:unique name="Categories_UniqueKey_0">
    <xs:selector xpath="."/>
    <xs:field xpath="CategoryName"/> Mapping the relationship between Tables
  </xs:unique>
  <xs:keyref name="FK_Products2" refer="PK_Categories">
    <xs:selector xpath="Products"/>
    <xs:field xpath="CategoryID"/>
  </xs:keyref>
</xs:element>
...

```

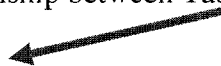


Figure 4-10 - Fragments of the XML schema (XSD) for NorthWind indicating the steps

Chapter 5

Converting XML Schema to Relational Database

In this chapter, the conversion of XML schema (XSD) into relational database schema is considered, and proposed algorithms for XML-to-Relational transformation are presented.

We propose a prototype called CODAX that maps an input XML schema to a relational database schema. The process contains the following two steps:

1. Transforming an XML schema to a ContextMap model, which will be, discussed in section 5.1. Since an XML schema can be very complex due to its hierarchical nesting capability, a ContextMap model is used to map complex XML database schema into structured ContextMap schema. On the other hand, generating relational schemas from a ContextMap model makes the process of mapping simple.
2. Generating relational schemas, which will be discussed in section 5.2. We generate a relational schema from the ContextMap representation after mapping an XML schema to a ContextMap model

5.1 Mapping XML to ContextMap Representation

In this section we propose a new process for mapping XML schemas into ContextMap notation.

	1	2																					208	209	210																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	2																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
+	341	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F</

Figure 5-1 - ContextMap schema for NorthWind XML schema

We construct the ContextMap schema shown in figure 5-1 from the NorthWind [NSLB] XML schema in figure 1-2.

The ContextMap consists of sets, namely {XML Source}, {XML Nodes}, {XML Node Markups}, {XML Node Properties}, {XML Node Property Values}, {XML Node Comments}, and {XML Node Processing Instructions}.

Set roles namely, “A”, “F”, and “N”, and set member roles, namely “f” and “t”, are used to construct the ContextMap model. Set role “S” is allocated to {XML Source}, and “F” is allocated to {XML Nodes}. “N” is allocated to {XML Node Markups},

{XML Node Properties}, {XML Node Property Values}, {XML Node Comments}, and {XML Node Processing Instructions}.

The user can also obtain useful information in ContextMap representation by applying the query tool. The terminology and symbols of ContextMaps are introduced in chapter 3 in detail.

In figure 5-1, the schema of the NorthWind XML schema is described as following:

{XML Node} (Row 341: F) has {XML Node Markups} (Row 546: N).

{XML Node Markups} (Row 546: N) have {XML Node Properties} (Row 561: F).

{XML Node Properties} (Row 561: F) have {XML Node Property Values} (Row 577: F).

{XML Node Properties} (Row 561: F) have {XML Node Comments} (Row 647: F).

Figure 5-2 - ContextMap model of NorthWind XML schema (1)

Figure 5-3 - ContextMap model of NorthWind XML schema (2)

[illegible]

Figure 5-4 - ContextMap model of NorthWind XML schema (3)

In the figures 5-2, 5-3, and 5-4, each set members is shown in right most column under the set name. In figure 5-5 we represent the relationship between database objects, set members, and set names in our context as follows:

<i>Database Object</i>	<i>Set Member</i>	<i>Set Name</i>
XML Node hierarchy	<i>n1, n2, n3, and...</i>	{XML Node}
XML Node Markups	<i>?xml, xs:schema, xs:element, xs:complexttype, xs:sequence, and ...</i>	{XML Node Markups}
XML Node Properties	<i>version, encoding, xmlns:xs, elementFormDefault, attributeFormDefault, name, base, and...</i>	{XML Node Properties}
XML Node Property Values	<i>“Categories”, “CategoryID”, “xs:integer”, “CategoryName”,...</i>	{XML Node Property Values}
XML Node Comments	<i>“Orders table primary key”</i>	{XML Node Comments}

Figure 5-5 - Summary of relationship between database objects, set members, and set names

Set member *n1*, *n2*, and ... under {XML Nodes} are used to show the hierarchy structure of the XML. In order to accomplish the hierarchy shown in figure 5-2 set member roles “f” and “t” are placed in the corresponding row.

Moving down vertically from “f” or “t” to the area of {XML Node Markups}, we have found another “f” which corresponds to the set members under {XML Node Markups}. The set members under {XML Node Markups} are XML tags such as: *?xml*, *xs:schema*, and so on.

Similarly, moving down vertically in figure 5-3 from set member role “f” in the {XML Node Markups} area, we find numbers under {XML Node Properties} area. The set members under {XML Node Properties} are XML properties such as *version*, *encoding*, and so on.

We continue to traverse the ContextMap from numbers in the {XML Node Properties} area down to find another type of numbers in the {XML Node Property Values} area as shown in figure 5-4. The set members under {XML Node Property Values} are property values such as “1.0”, “UTF-8” and so on.

For example, if we want to find the related information for Categories table, we move from XML Node (n3) vertically to find the related XML Markup, which is xs:element. For instance from set member n3 (Row: 344), we move horizontally to the left to retrieve the first “f” set member role (Row: 344, Column: 6).

To indicate the XML hierarchy in the {XML Nodes} area, we use “t” set roles in the same column and the order in which they appear in the area.

For example, n7, n9 and n13 represent a hierarchy tag structure beginning with n7 as the topmost level tag, which contains n9 as the middle level, and n13 as the lowest tag level.

From the retrieved “f” in the {XML Nodes} area, we move down in the same column to find the related “f” in the {XML Node Markups} area, and move right to find the corresponding set member which is xs:element.

We continue to traverse in the same manner to find all related {XML Node Markups}, {XML Node Properties}, {XML Node Properties Values}, and {XML Node Comments} corresponding to {XML Nodes} n3.

5.1.1 Mapping XML Schema to ContextMap Representation

Algorithm

The algorithm below retrieves the information from an XSD file, and loads XML metadata information to a ContextMap model. Let X be the input XSD file and $N = (N_1, \dots, N_j)$ be all the individual nodes in an XSD file, and $S = (S_1, \dots, S_n)$ all the sets that belongs to XSD schema.

Input: X, N, S	<i>/* XSD file, Sets and XML Nodes*/</i>
Output: $XML_WorkSheet$	<i>/* XML_WorkSheet which represents, ContextMap model of XSD file */</i>
Algorithm:	<i>Load XML (XSD file) information to Context Map Schema for Xml</i>
With given sheet	
For each S_i in S where i in $[1..n]$	
For each N_j in X where j in $[1..m]$	
<ul style="list-style-type: none">• Insert N_j as set member under {XML Nodes} set, and Insert the association between XML nodes.• Insert XML tags, as set member under {XML Node Markups} set, and Insert the association between XML nodes, and Node Markups.• Insert XML Properties for each tag as set member under {XML Node Properties} set, and Insert the association between {XML Node Markups}, and {Node Properties}.• Insert XML Property values for each tag as set member under {XML Node Property Values} set, and Insert the association between {XML Node Properties}, and {Node Property values}.	
End.	

In order to map XML (XSD) schema to ContextMap schema by the above algorithm, we need to go through all steps as detailed next:

We assume that the ContextMap XML schema already exists, and it contains {XML Nodes}, {XML Node Markups}, {XML Node Properties}, {XML Node Property Values}, {XML Node Comments}, {XML Node Processing Instructions} sets, and “F” and “N” set roles.

In the first step, we load XML nodes (N_1, N_2, \dots, N_m) as set members under {XML Nodes} set, and use “f” and “t” set member roles to associate each node to its sub nodes.

In the second step, for each node in the XSD file we load XML tags such as xs:schema, xs:element, xs:complexType, xs:sequence, xs:simpleType and , ... as set members under {XML Node Markups}, and we use “f” set member role to associate each node to its tags.

In the third step, we load each property in XML nodes under {XML Node Properties}, and we use numbers (1, 2,...) to determine the number of the properties in each node and to connect each node to its tag and properties.

In the fourth step, we load each property values in XML nodes, under {XML Node Property Values} and again we use numbers to associate each XML node property to its property value.

Finally in the last step if XSD file contains any comments, we load the comments under {XML Node Comments} section.

5.2 Converting ContextMap model to Relational

Database

In this section, we present the algorithm below to translate the conceptual schema, which is presented as ContextMap model of XML schema into relational schema. The process in pseudocode is depicted in the algorithm below.

Input: XML_WorkSheet */* XML ContextMap Model */*
Output: RDB Schema */* the corresponding relational schema */*
Algorithm: *Generate RDB Schema from ContextMap model of XML Schema*

With given sheet

For each N_i where i in $[1..n]$

1- Generate relational structure from a given ContextMap model from an XML schema by performing below steps:

- Find the “xs:element” set member under {XML Node Markups} set, and search for related set members under {XML Property Value} set.
- Translate the set member into a relational table.
- Find the inner hierarchies under “xs:complexType” set member under {XML Node Markups} and translate each sequence element into an attribute of that relational table.

2- Transform XML schema constraints into relational schema constraints by translating:

- "unique" set member under {XML Node Markups}, and related xml nodes into relational constraints.
- "key" under {XML Node Markups}, and related xml nodes into relational constraints.
- "keyref" under {XML Node Markups}, and related xml nodes into relational constraints.

End.

In order to convert the ContextMap model to relational database schema by above algorithm, we need to go through the steps as detailed next:

We use retrieved associations, in order to traverse the ContextMap and for each node N_i , $N = (N_1, \dots, N_n)$ extract XML Markups, XML Properties and XML Property Values.

Based on each XML Node Markup, we extract XML Property and Property Values for each node and generate the relational database DDL script.

In the first step of the algorithm, we construct the relational database DDL script, as follows:

1. Map the property value of `<xs:element` tag as table name.
2. Map the property value of each sub element `<xs:element` under `<xs:sequence>` as field name.
3. Map the property value of each sub element `<xs:restriction` under `<xs:simpleType>` as field type.
4. Map the property value of each sub element `<xs:maxLength` as field length.

In the second step of algorithm, we transform the XML schema constraints into relational schema constraints, as follows.

1. Map primary key constraints and related fields by extracting property values of `<xs:key` element and sub elements .
2. Map foreign key constraints and related fields by extracting property values of `<xs:keyref>` element and sub elements.

3. Map unique key constraints and related fields by retrieving property values of <xs:unique element.

Figure 5-6, shows the DDL (Data Definition language) example in Oracle which shows the mapping according to the above algorithm steps:

```
CREATE TABLE Categories      ← Mapping the table name
(
  CategoryID Number ,
  CategoryName varchar2(500) ← Mapping the attribute
  Description varchar2(4000)
  Picture varchar2(4000)
)
TABLESPACE User_Data
PCTFREE 10
PCTUSED 50
INITRANS 1
MAXTRANS 200
/

ALTER TABLE Categories ADD ( ← Mapping the unique keys and the primary keys
  CONSTRAINT PK_Categories PRIMARY KEY (CategoryID))
/

ALTER TABLE Products ADD ( ← Mapping the foreign keys
  CONSTRAINT FK_Products2 FOREIGN KEY (CategoryID)
  REFERENCES Categories (CategoryID))
/
```

Figure 5-6 - Part of NorthWind DDL example in Oracle

Chapter 6

A System Prototype

We have designed and implemented a prototype for converting the relational database schema to the XML schema and vice versa. We refer to this prototype as CODAX, which stands for **C**onverting **R**elational **D**atabase to **X**ML and vice versa. It leads to identify and understand all components of an existing database and the relationships between them. It also provides an algorithm for performing reverse engineering process and converting relational database into XML schema and vice versa. System design is discussed in section 6.1. The illustration and architecture for CODAX are discussed in section 6.2 and 6.3.

6.1 System Design

CODAX supports the schema conversion from structured into semi-structured schema and vice versa, using ContextMap map as a conceptual model.

In order to convert semi-structured to structured schema, it is necessary to extract the schema information from semi-structured metadata information. For this purpose, our prototype CODAX includes an XML parser [DHW01], which extracts the XSD

schema and a transformer, which maps the schema into ContextMap representation, and an analyzer. The analyzer process goes through the map and extracts the necessary information from the map, and performs the forward engineering process. The same mentioned steps are needed when converting structured to semi-structured schema.

There is a visual feature in CODAX, which enables users to have a list of available tables, their columns, primary keys, and foreign keys in the form of the ContextMap representation. The above metadata information has already been mapped to the ContextMap model. Users can modify the schema information, using the CONTEXT+ environment, and can also issue different query options “AND”, “XOR”, “OR”, “NOT” over this generated ContextMap model, and have the ability to merge different ContextMap schemas which belong to different database sources.

CODAX has an interface which exhibits the tables within the ContextMap model generated from the relational database, in which users can select the desired tables and use transformer functions to convert the ContextMap model to the XML schema. CODAX has also similar screens for converting the XML into the relational database schema.

Finally, CODAX has an interface to construct ContextMap representation from the relational database or XML schema. In general, CODAX provides functionalities listed below:

- Supports a visualized interface.
- Can be used for relational databases conversion into XML using any of the known RDBMS including SQL/Server, DB2, Oracle, and MS Access.

- Gives Users the option to choose between a relational database and an XML schema.
- Keeps a list of the converted and available databases, which can be used to add or delete any table or table property from the map by system developers.
- Extracts schema information from relational as well as XML database schema, and displays the result within the GUI.
- Performs reverse engineering process, and builds different ContextMap models for the given relational database or XML.
- Converts the constructed ContextMap model from relational databases schema to the corresponding XML schema and vice versa.
- Converts the constructed ContextMap model from XML to the corresponding relational database schema and vice versa.
- Provides direct view of the steps of each process.

CODAX is composed of four main modules:

1. CCTXLRR - Converting ContextMap model to relational database schema module.
2. CCTXXML - Converting ContextMap model to XML schema module.
3. CTX2XML – Constructing ContextMap model from XML schema module.
4. CTX2RDB – Constructing ContextMap model from relational database schema module.

Figure 6-1 illustrates the modules of CODAX and its flow.

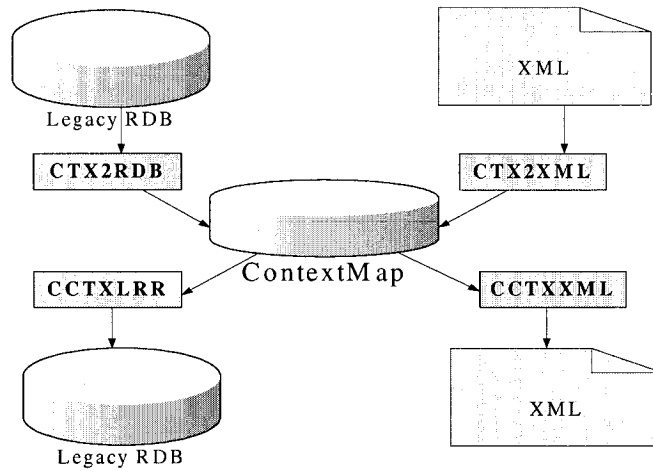


Figure 6-1 - CODAX modules

6.2 System Requirements

1. Hardware and software availability: CODAX has been implemented in two-tier network architecture. For this purpose we used different clients, and a data server, which may consist of different data base instances.
2. Development requirements: CODAX has been implemented using Oracle 9i, Microsoft Access 2000, Microsoft SQL/Server 7, DB/2

database, VBA (Visual Basic for Applications), Microsoft Excel 2003 tools, and Windows XP operating system.

3. We use ODBC Interface to connect to different databases.
4. CODAX can be run into a CPU Intel Pentium 4 with a speed of 2.40 GHz and 512 MB for the main memory. Most of the coding is based on VBA (Visual Basic for Applications) language, with more than 2000 lines of code.

6.3 System Illustration

In this section we will illustrate our developed system prototype, which includes solutions for two-way conversion, and also performs processes of mapping relational or XML schema into related ContextMap representations through reverse engineering process. Based on the schema selected by the user, CODAX generates different ContextMap representations to illustrate structured or semi-structured schemas.

In addition, the prototype has the ability to generate different database schemas from ContextMap representation using ODBC (open database connectivity). A series of CODAX functionalities have been illustrated as follow.

In figure 6-2, two entries have been added to the CONTEXT+ form in order to convert a semi-structured to structured database and vice versa.

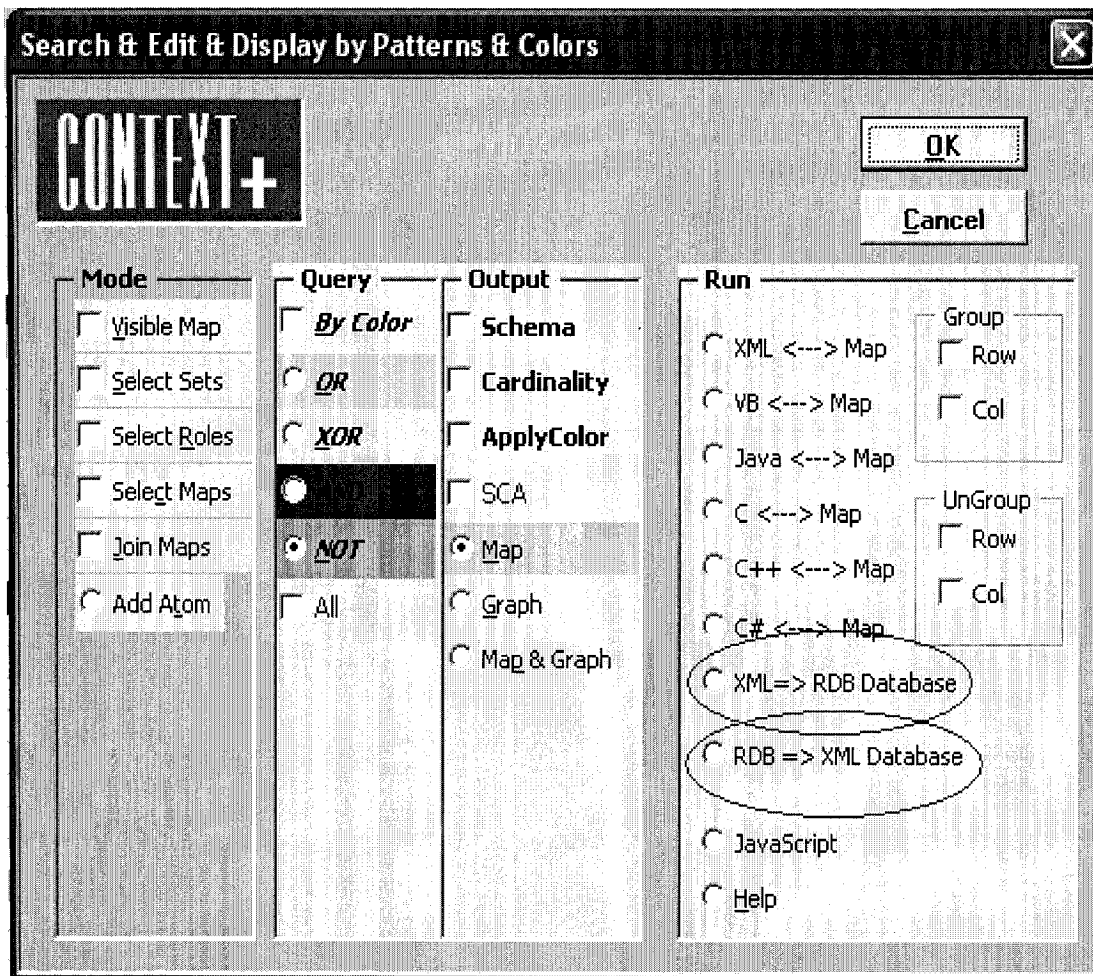


Figure 6-2 - CONTEXT+ form

6.3.1 Converting Relational Database to XML Schema

The dialog shown in figure 6-3 is used to convert relational database schema to XML. To do so, users need to follow two steps: using the first tab gives the user the ability to perform reverse engineering process and thus transform relational database schema to the ContextMap schema. For this purpose we, call a set of functions inside CTX2RDB. Users need to choose the source database and click on the *Import* push

button shown in figure 6-3, clicking on import push button will invoke the dialog shown in figure 6-4.

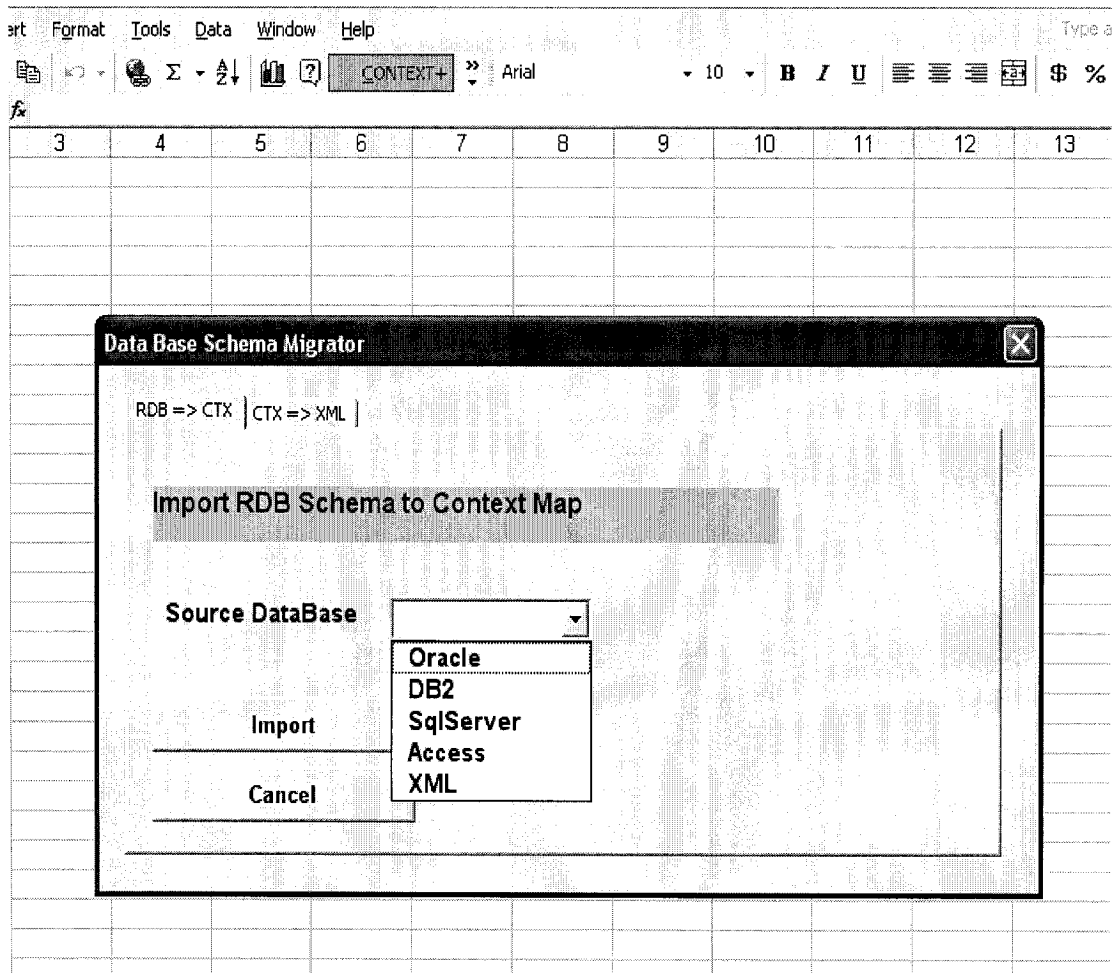


Figure 6-3 - Dialog to choose source database

As shown in figure 6-4, users have the ability to either use database DDL script, or capture the database schema directly and perform the reverse engineering process. Using either option will result the dialog shown in figure 6-5.

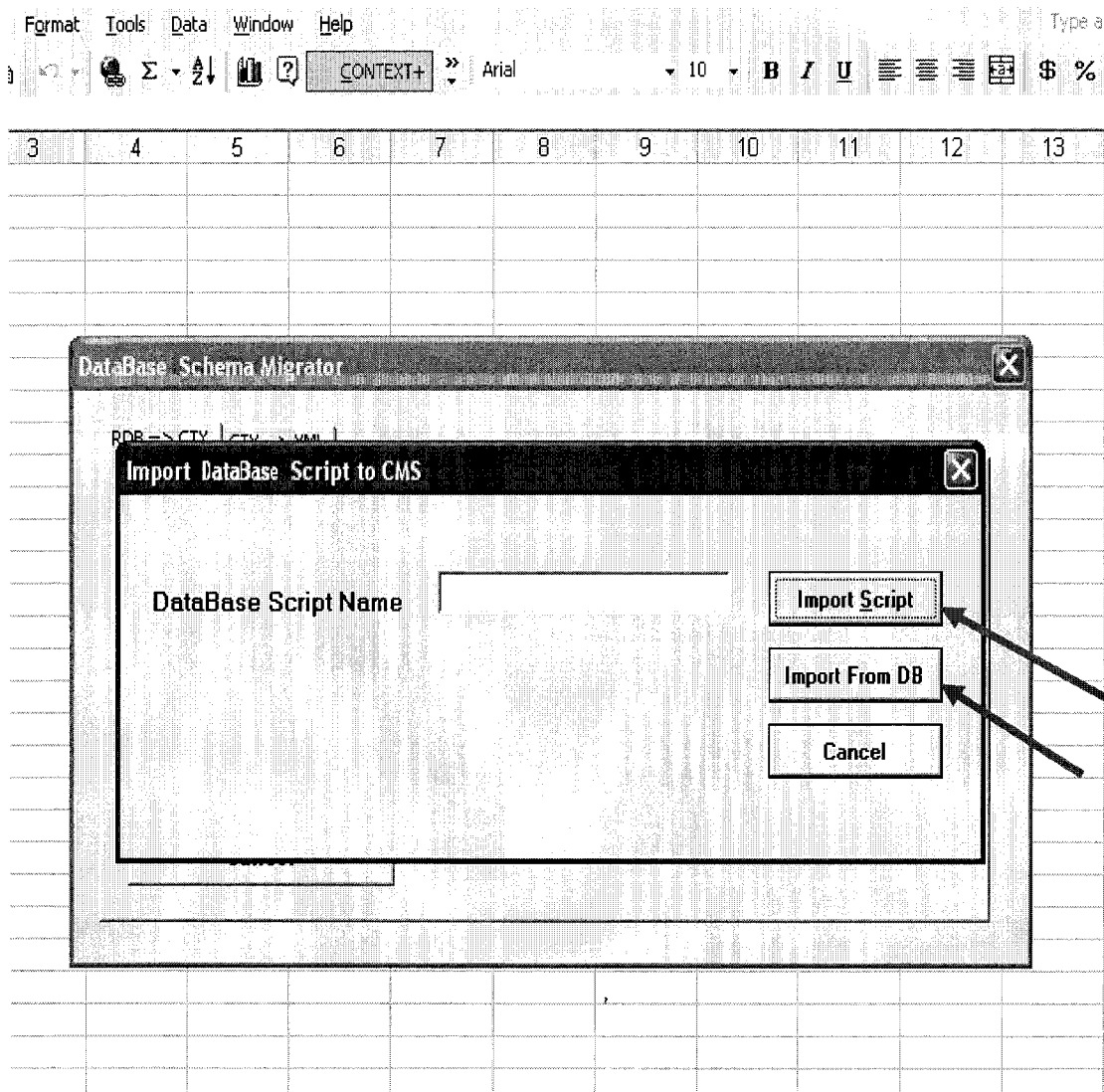


Figure 6-4 - Dialog to perform reverse engineering process

	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	V	V	V	V	V	V	V	V	V	V	V	V	V	11	3	{View}			
2														4		Types			
3	V	V	V	V	V	V								8		Tables			
4							V	V	V	V	V	V	V	7		References			
5	F	F	F	F	F	F	F	F	F	F	F	F	F	15	8	{Table}			
6							t							3		OrderDetails			
7														5		Orders			
8							t							4		Products			
9								t						2		Customers			
10												t		2		Suppliers			
11													t	2		Categories			
12										t				2		Employees			
13											t			2		Shippers			
14	N	N	N	N	N	N								8	8	{Primary Keys Constraints}			
15														1		PK_OrderDetails			
16														1		PK_Orders			
17														1		PK_Products			
18														1		PK_Customers			
19														1		PK_Suppliers			
20														1		PK_Categories			
21														1		PK_Employees			
22														1		PK_Shippers			
23							E	E	E	E	E	E	E	7	7	{Foreign Keys Constraints}			
24														1		FK_OrderDetails1			
25														1		FK_OrderDetails2			
26														1		FK_Orders1			
27														1		FK_Orders2			
28														1		FK_Orders3			
29														1		FK_Products1			
30														1		FK_Products2			
31	N	N	N	N	N	N	N	N	N	N	N	N	N	19	52	{Field}			
84	N	N	N	N	N	N								8	2	{Table Space Types}			
87														4	4	{Type}			
92	N	N	N	N	N	N								8	10	{Table Spaces}			
103	A	A	A	A	A	A	A	A	A	A	A	A	A	11	1	{AUTHOR}			
104	V	V	V	V	V	V	V	V	V	V	V	V	V	11		Syntax and Patterns © by W.M. Jaworski, 1988-2001			
105	V	V	V	V	V	V	V	V	V	V	V	V	V			Mapped by Sanaz Rahmati			
106	#	#	#	#	#	#	14	14	14	#	#	#	#	200	10				
107	8	8	8	8	5	6	6	6	6	8	5	8	6	70	52				

Figure 6-5 - ContextMap model of relational database schema

Clicking on the CTX=> XML database schema tab gives the user the option to perform forward engineering process, and will invoke the dialog shown in figure 6-6. Users can choose either XML or relational database in this section. For example, if the source database is Oracle, then after performing reverse engineering process, users can manipulate the ContextMap schema and transform it to another schema such as SQL/Server database or XML schema. For this purpose we call a set of functions inside CCTXXML module.

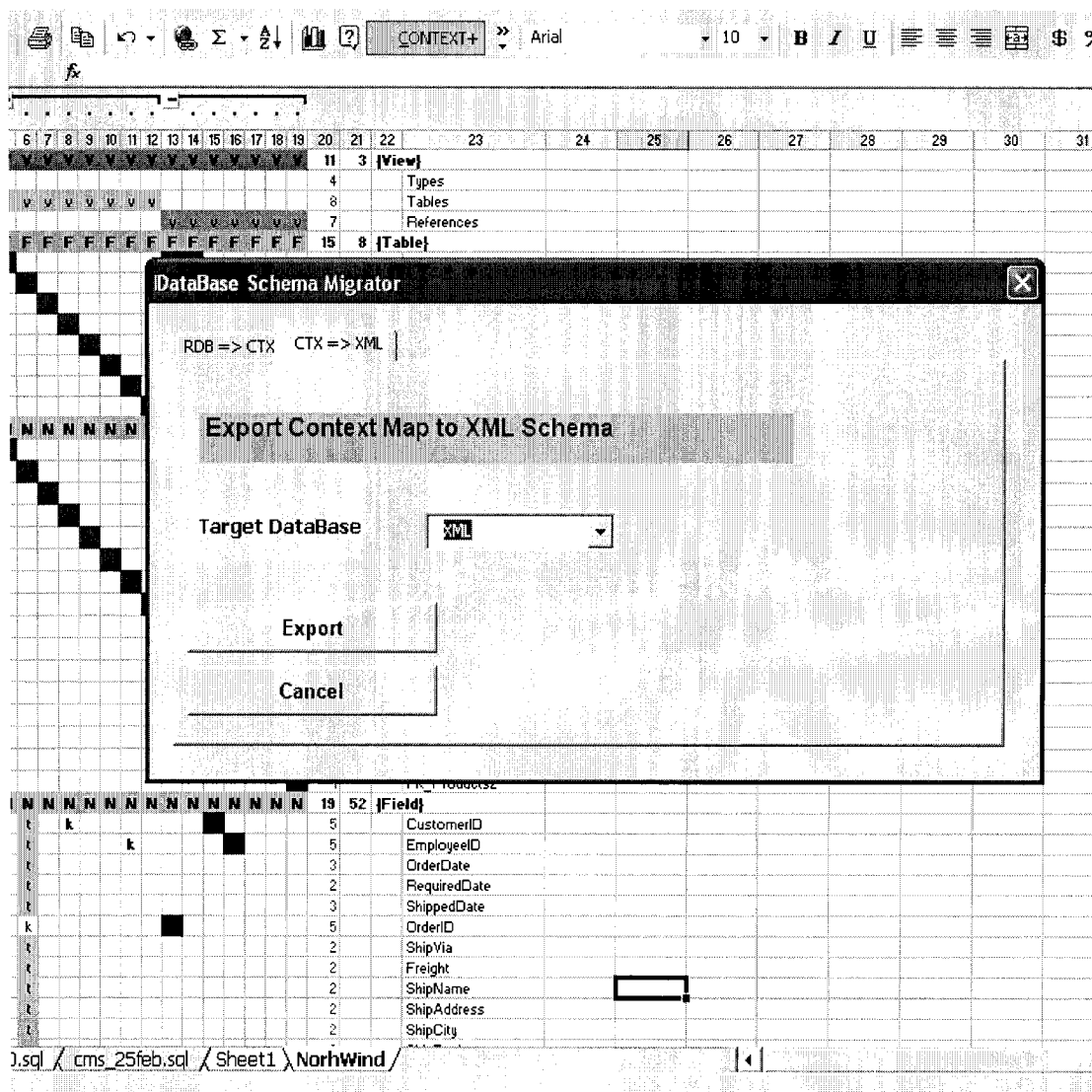


Figure 6-6 - Export ContextMap model to XML schema (1)

Clicking on the *Export* button will invoke the dialog shown in figure 6-7. Users can select the name of the tables, and also select the name of the script file shown in figure 6-8. By clicking on the *OK* button as shown in figure 6-9, the database script from ContextMap schema will be generated. By clicking on the *View Push* button the generated script from the ContextMap schema will be accessed as illustrated in figure 6-10. The *Execute* button is used to create the relational database schema from a script if users have chosen the target database as relational.

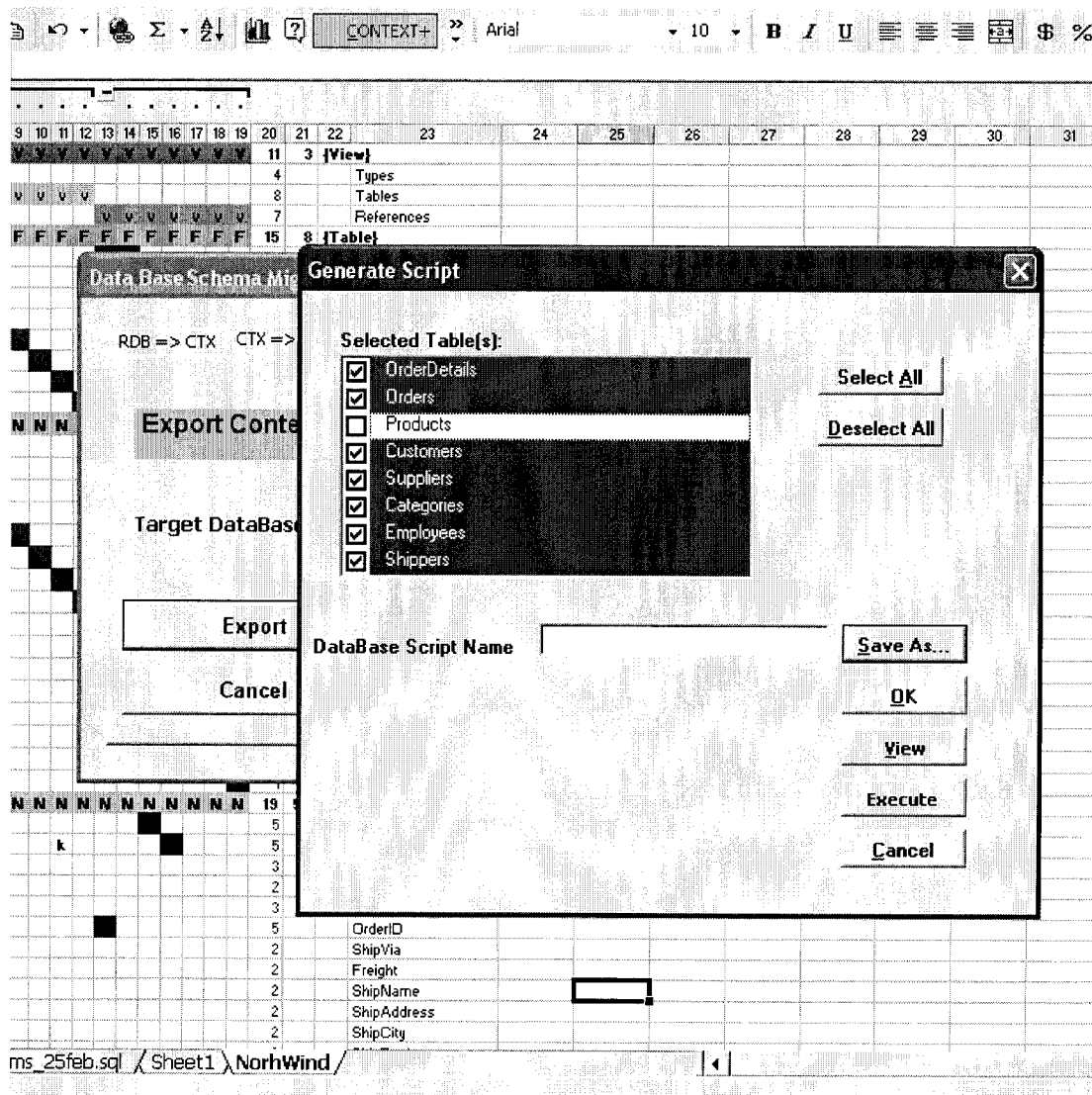


Figure 6-7 - Export ContextMap model to XML schema (2)

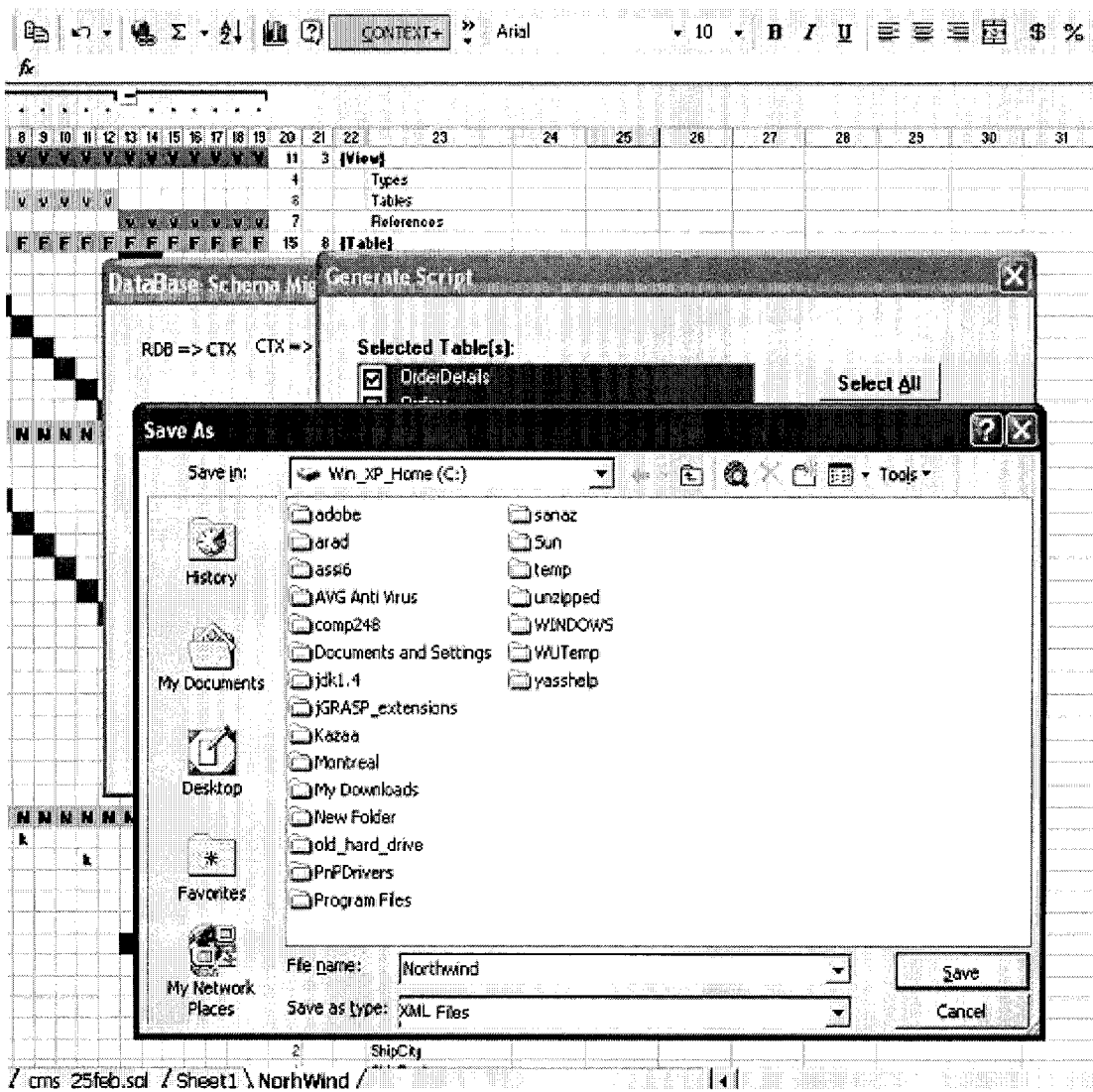


Figure 6-8 - Export ContextMap model to XML - choosing file name

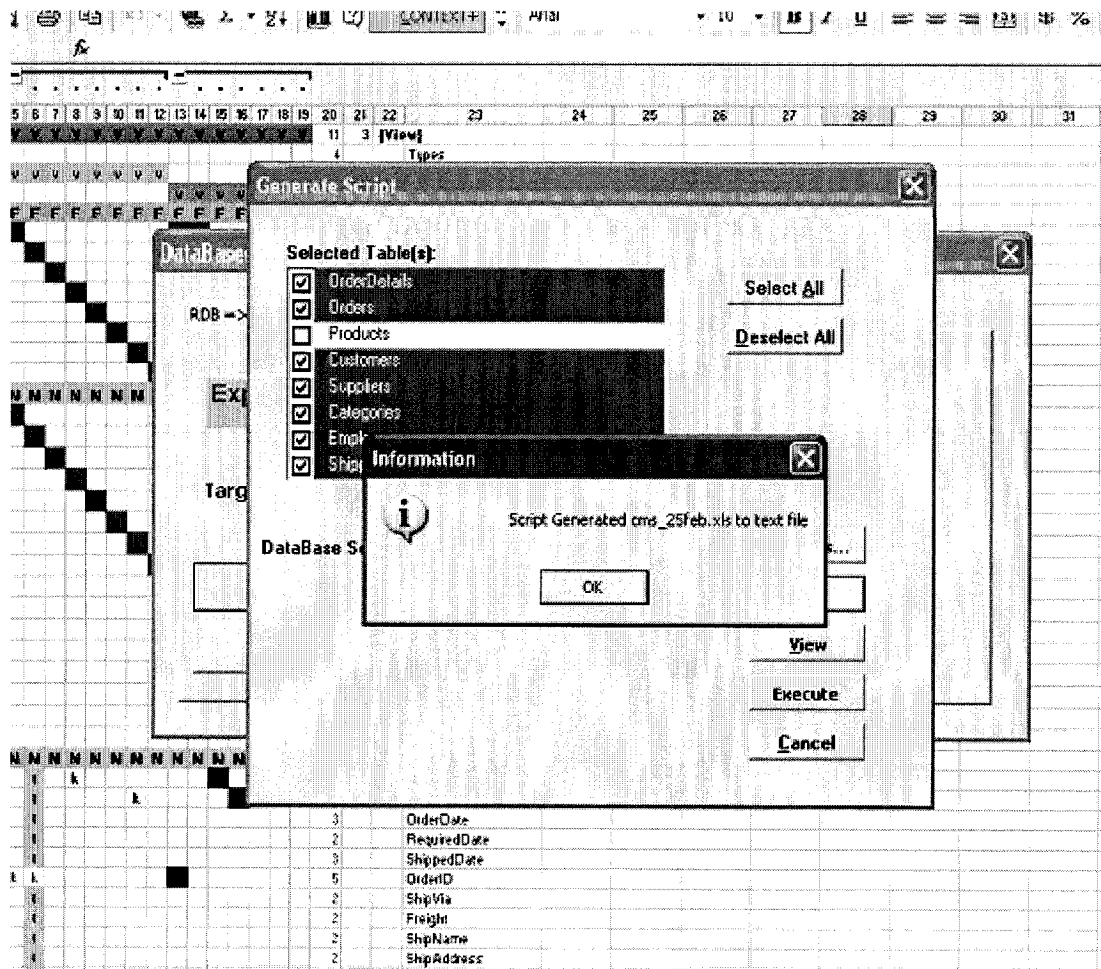


Figure 6-9 - Generate script

As shown in figure 6-10, users also have the ability to view the generated script.

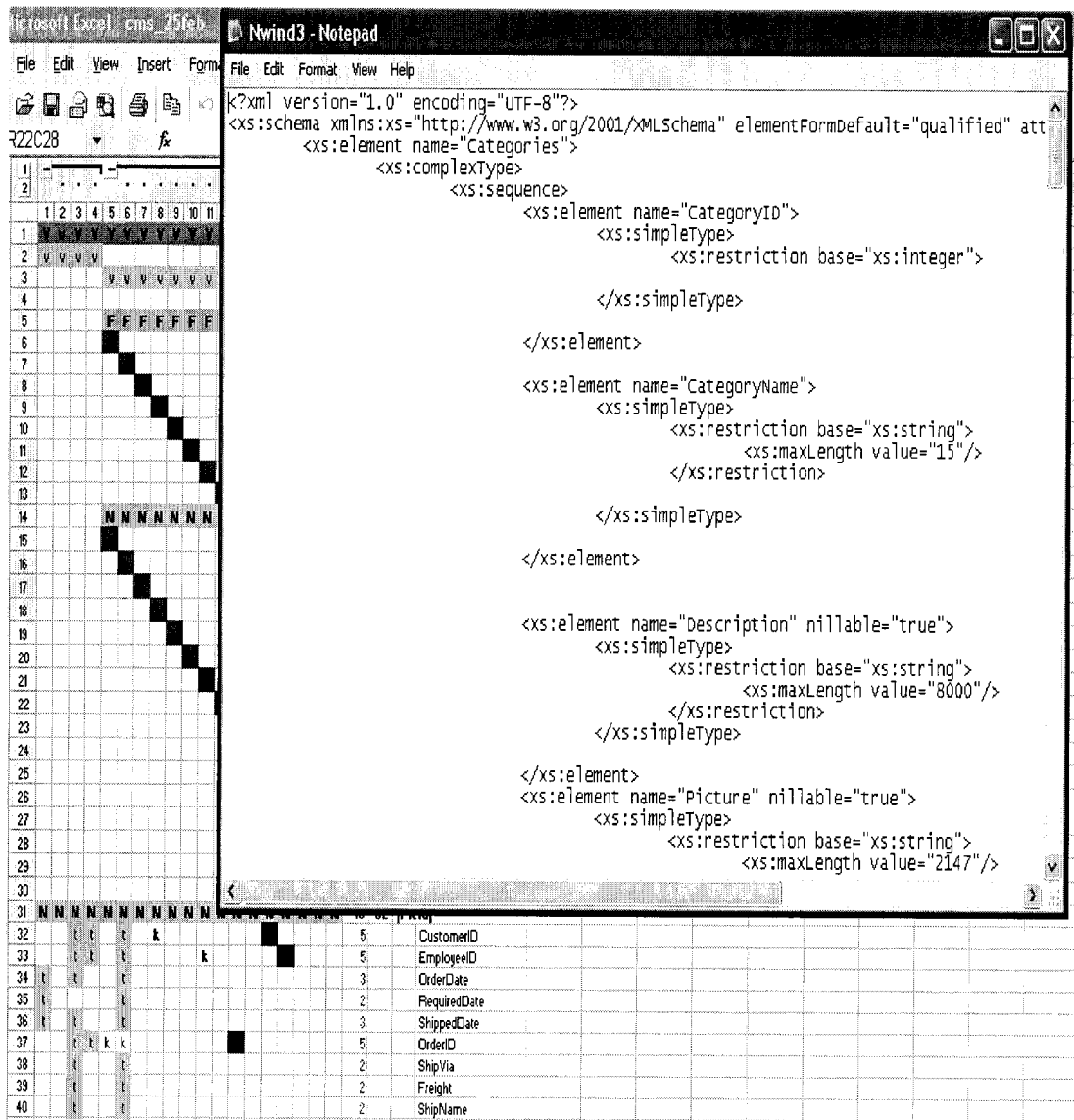


Figure 6-10 - View generated XML schema script

6.3.2 Converting XML to Relational Database Schema

The dialog shown in figure 6-11 is used to convert XML to relational database schema. The process is done in two steps. The *XML=> RDB* database schema tab

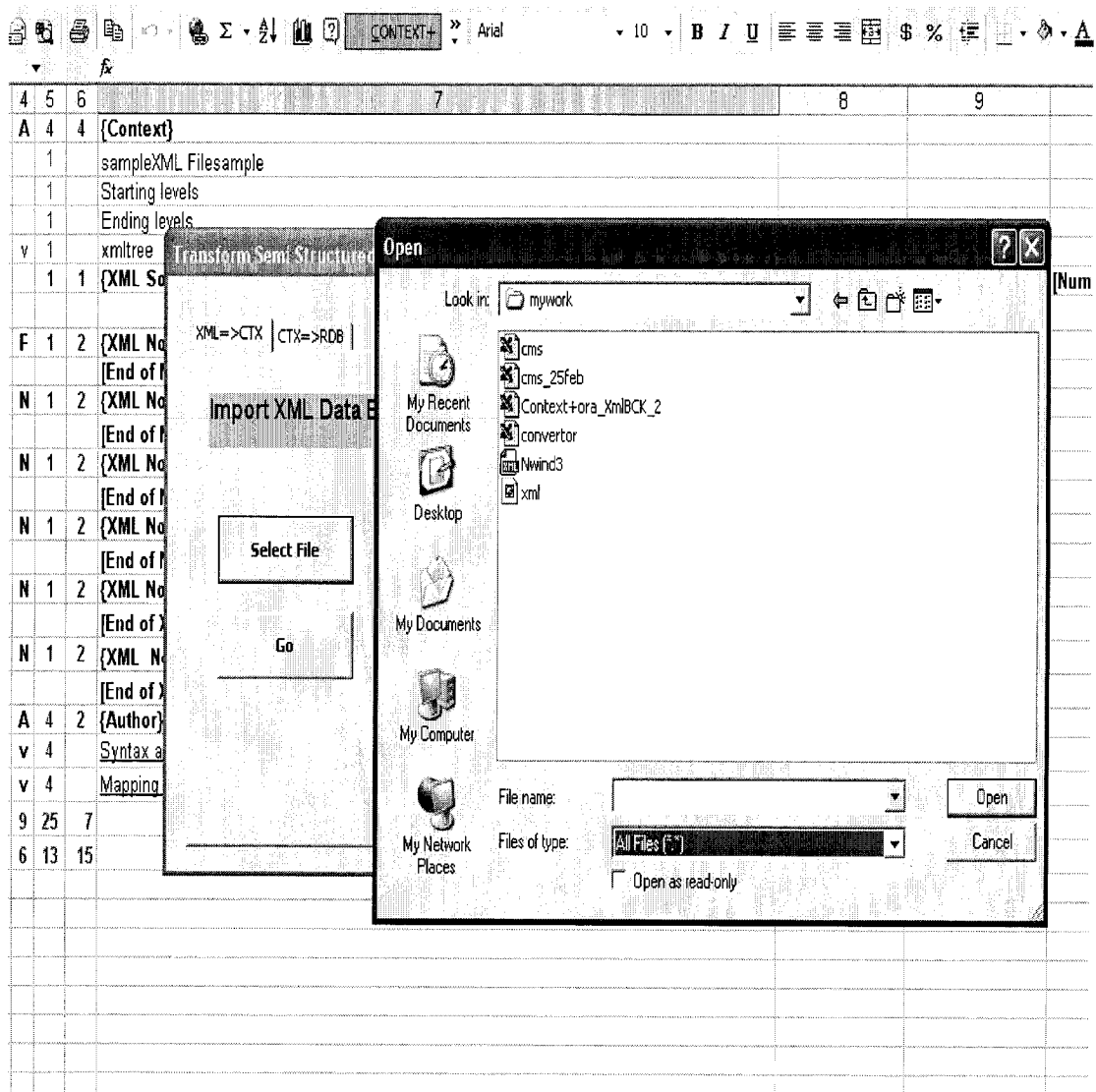


Figure 6-12 - Dialog to perform reverse engineering process, select XML file

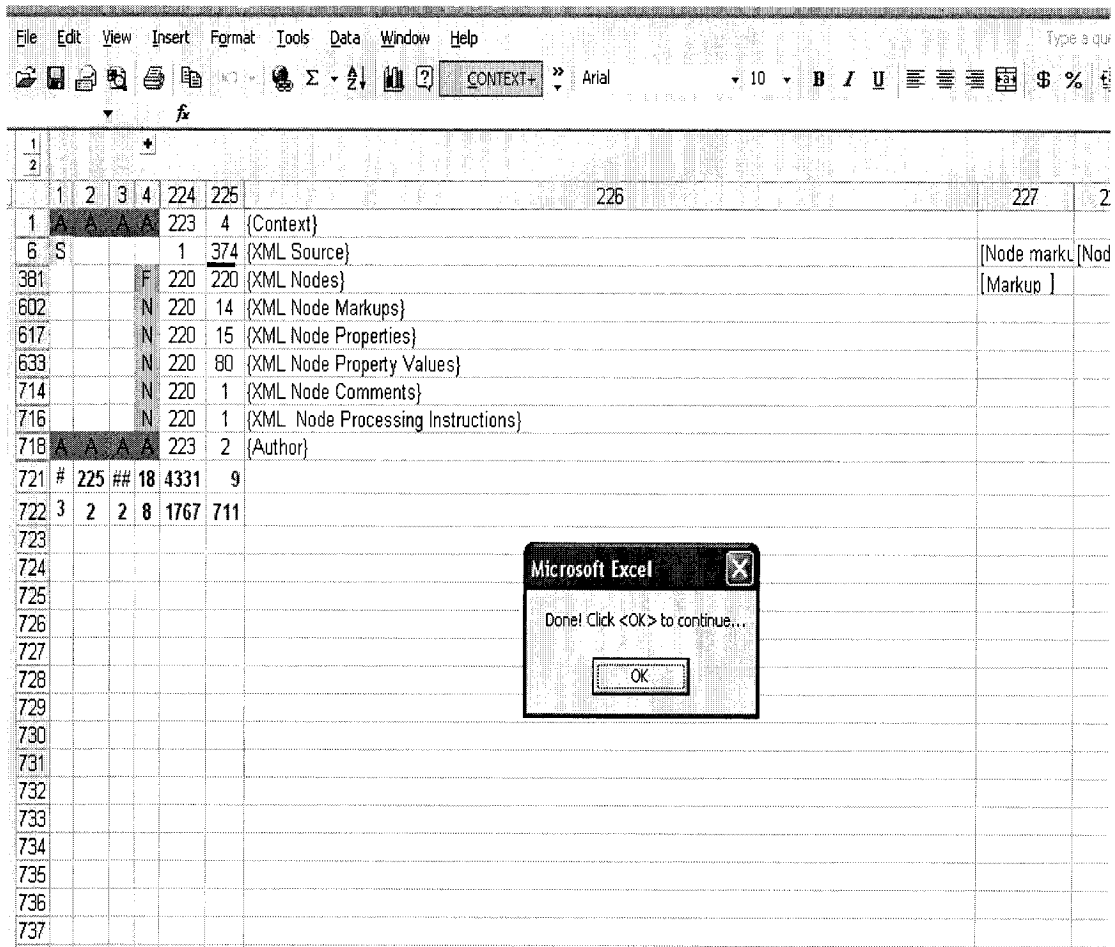


Figure 6-13 - Generated ContextMap schema from XML file

CTX=> RDB tab will invoke the dialog shown in figure 6-14. Users can choose either XML or relational database as the target database in this section and also can choose the script file or XML file name by clicking on the *Save* button as in figure 6-15. The *OK* button will generate the file based on the target database.

The *View File* button provides a view of the generated as in figure 6-16.

The *Execute* button creates the relational database schema, based on the generated script. For this purpose we call some functions inside CCTXLRR module.

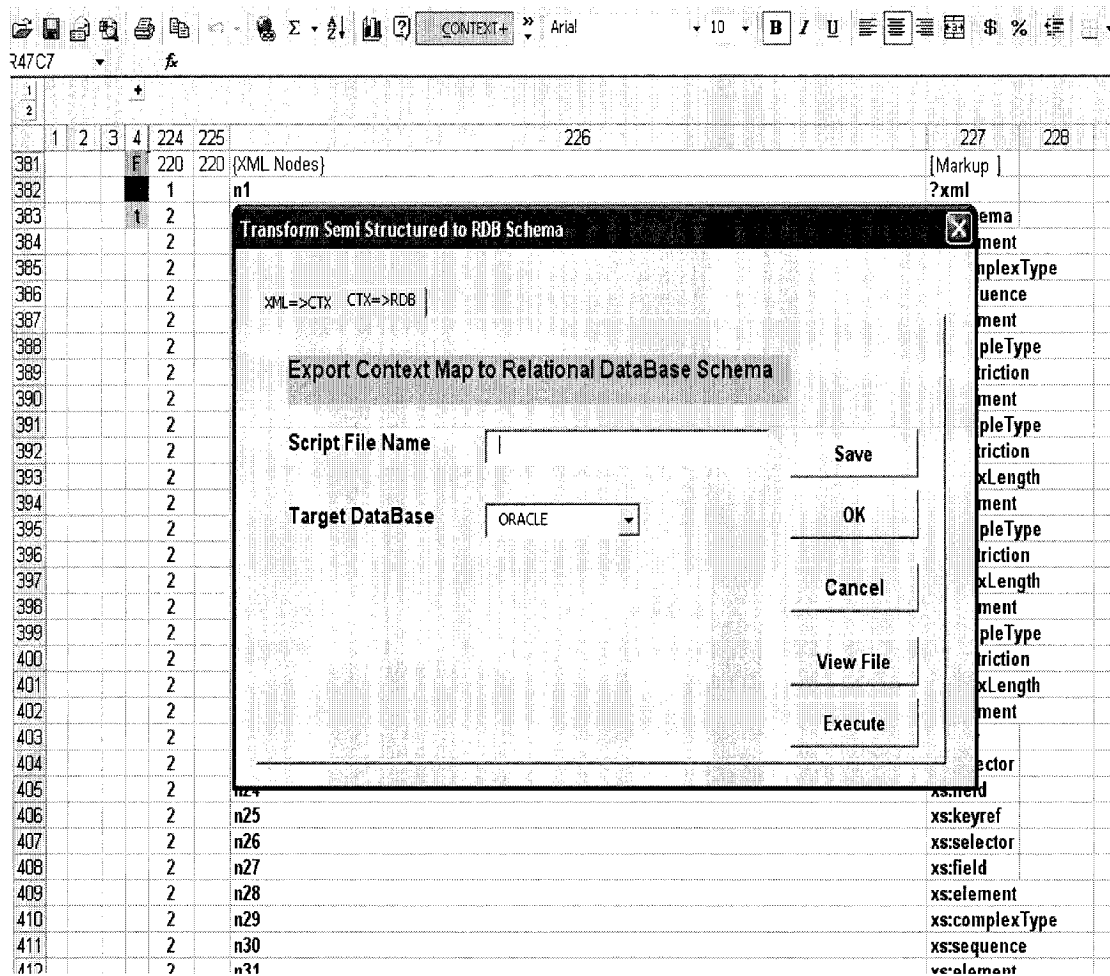


Figure 6-14 - Generate RDB schema from ContextMap

As it can be seen in figure 6-15 and 6-16 users have the ability to choose and view the generated script.

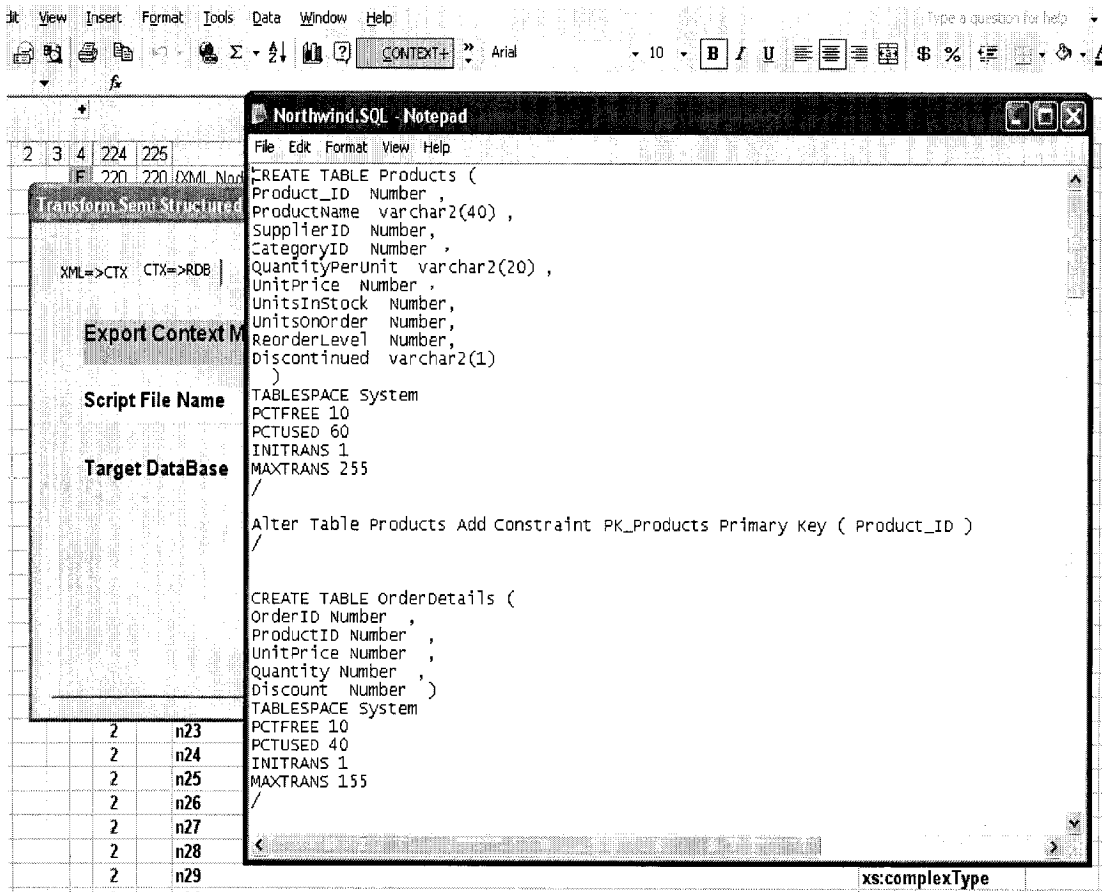


Figure 6-16 - View generated RDB schema script

6.4 Architecture

CODAX architecture model is shown in figure 6-17. As can be seen from this diagram, the source database can be either of the following types: XML, SQL/Server, Oracle, DB/2, or Access.

The Extract-Map process uses the individual source and maps its metadata information to the ContextMap. The new mapped metadata information is stored in the internal ContextMap repository. This repository can be used by the next process

(Extract-Transform), located in the target scope. The Extract-Transform process generates the desired target schema.

The CODAX processing model is illustrated in figure 6-18. It has different modules to support the schema conversion. “Extract Schema Information” and “Create ContextMap Representation” are two main modules used for reverse engineering process; they are used to extract the metadata information from the relational database or XML schema, and to generate the related “ContextMap Representation”. “Manipulate ContextMap Repository” is used to modify generated ContextMap models or perform queries. In order to perform forward engineering “Select Target Database”, “Retrieve schema from ContextMap representation” and “Transform Schema to ContextMap Representation”, are used to construct the target schema.

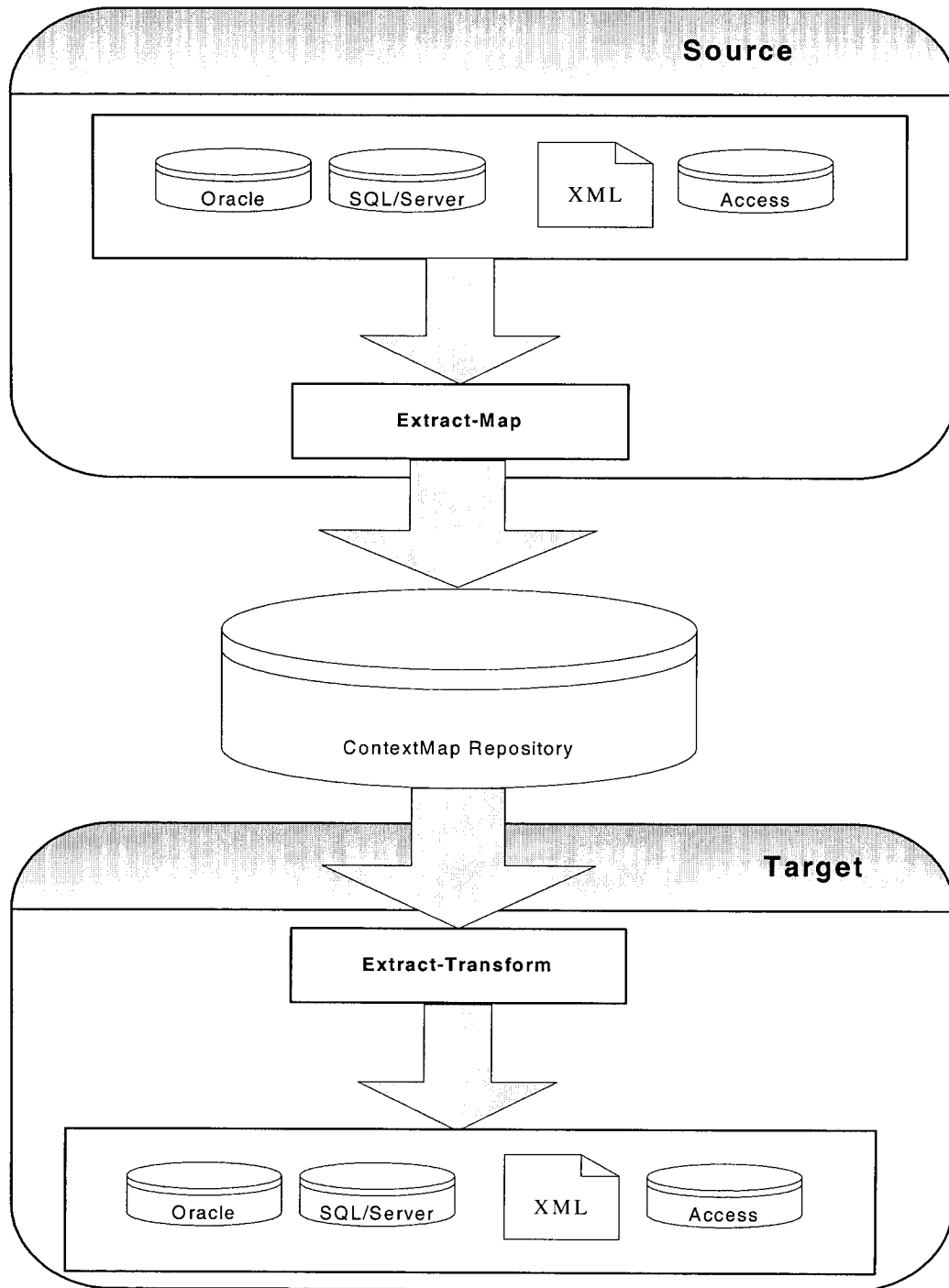


Figure 6-17 – Architecture of CODAX

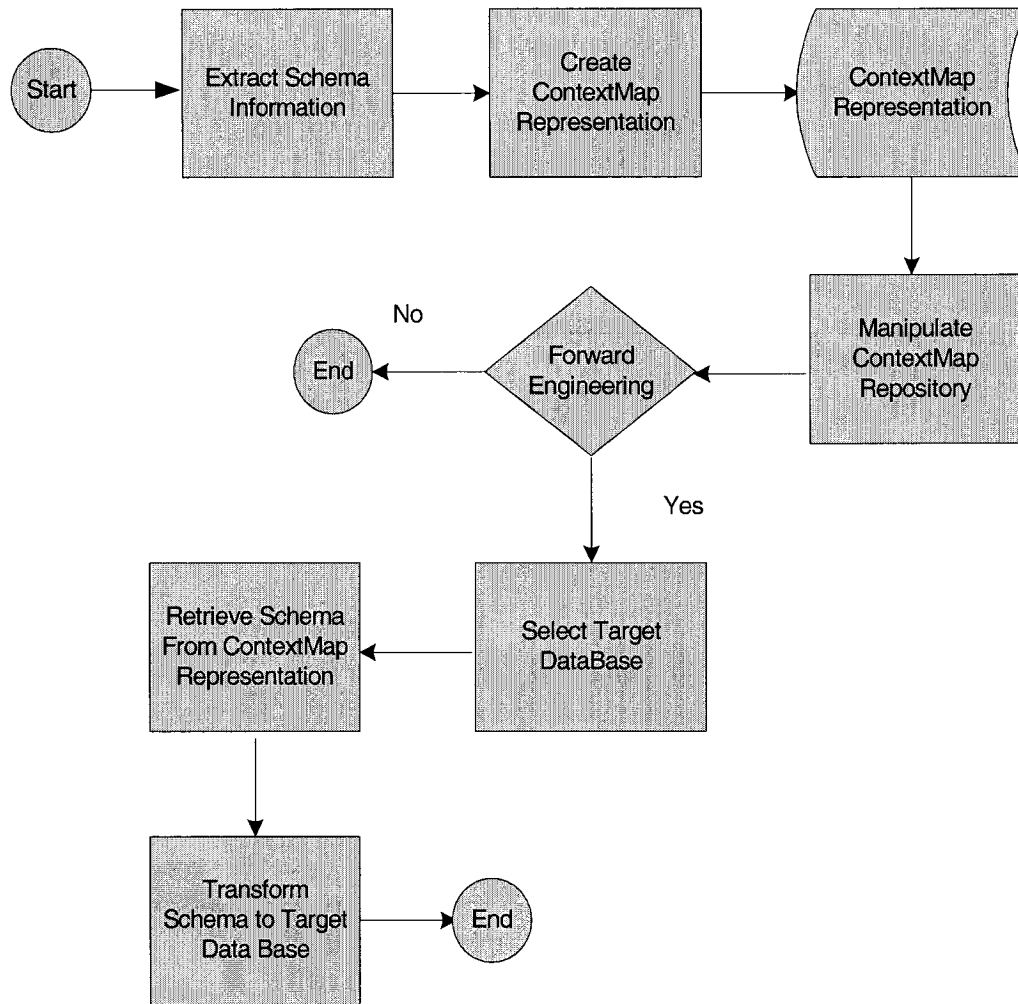


Figure 6-18 - Process model of CODAX

6.5 CODAX Performance

We assume that we have a main object such as table and objects such as columns, keys, foreign keys and candidate keys, which are related to the main object.

We have tested the proposed approach on the SAMPLE [NSLB] database in DB/2 and XML, and LIBRARY [NSLB] database in DB/2, Access, and XML, and NorthWind [NSLB] database in DB/2, MS SQL/ Server, XML, and Oracle, as well as Biblio [NSLB] database in MS SQL/ Server, XML, and Oracle.

<i>Database Name</i>	<i>Database Type</i>	<i>Number of Related Objects</i>	<i>Time to Construct ContextMap Model</i>
Biblio	XML	20	15"
	SQL/Server	20	12"
	Oracle	20	10"
NorthWind	DB/2	38	17"
	SQL/Server	38	16"
	Oracle	38	18"
	XML	38	20"
Library	DB/2	45	21"
	Access	45	24"
	XML	45	26"
SAMPLE	DB/2	54	31"
	XML	54	33"

Figure 6-19 - The comparison between running time and number of related objects

Testing these databases provided rough feedback and estimates about the relationship between the number of main objects and related objects, and the time required to convert it into XML or relational database schema. The result of the comparison between the run time required and the number of related objects is plotted in figure 6-

19. It roughly shows that, as the number of related objects to main object increases, the run time to extract the related objects and generate the ContextMap model from a database schema increases as well. Therefore, if the number of related objects increases, then the time for extracting keys, candidate keys, foreign keys and columns and constructing the ContextMap will increase.

The job of reconstructing ContextMap from a relational database is undoubtedly heavy and tedious, especially for a large real application. Particularly, deciding on candidate and foreign keys is time consuming because it requires analyzing the related tables and attributes. Users could be relieved from this heavy load by using CODAX.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we have presented a prototype called CODAX for converting relational databases into corresponding XML schema and vice versa. It includes developed algorithms for the reverse engineering and schema conversion. These algorithms handle the mapping of the semantic constraints as much as possible during the transformation process. We also construct and describe ContextMap schemas and propose new notations for that purpose.

Furthermore, CODAX can properly and equally deal with $1:1$, $1:M$, and $M:N$ relationships in the schema conversion. It provides essential functionalities that allow users to partially convert selected portions of a metadata information conversion.

We illustrated CODAX on an example NorthWind [NSLB] database. We construct the ContextMap model from the source database using reverse engineering algorithms, and use the CONTEXT+ environment to query and manipulate the sets within the map, and successfully transform the ContextMap model into the target schema.

Using the generated ContextMap model from different database schemas helps analysts and DBAs to:

- a) Deal with only one simple notation.
- b) Understand the structure of the data sources efficiently and also to apply the necessary changes accurately.
- c) Merge various ContextMap models generated from different database sources and schemas into one map.

We have tested the proposed approach on the SAMPLE [NSLB] database in DB/2 and XML, and LIBRARY [NSLB] database in DB/2, Access, and XML, and NorthWind [NSLB] database in DB/2, MS SQL/ Server, XML, and Oracle, as well as Biblio [NSLB] database in MS SQL/ Server, XML, and Oracle.

We performed the two-way transformation for above mentioned relational database schemas to XML schema and vice versa for both LIBRARY [NSLB] and NorthWind [NSLB] databases, doing so we have observed that the generated schemas are identical.

On the other hand, the users' knowledge could also be involved in this system. However, compared to reconstructing a ContextMap from scratch, it is almost impossible to manually reengineer a given relational database into XML or vice versa. As a result, the human beings mental workload is greatly reduced by using CODAX.

Further, designers, and DBAs can use CODAX to update and redesign and understand existing database systems.

The CODAX gives users a direct visualization for each phase of the process; therefore it helps users to easily view the result of each phase.

7.2 Future Work

As far as the future work is concerned, CODAX could be expanded in the following directions.

We would like to expand and develop new algorithms and interfaces to convert object oriented databases to XML, and also relational schemas. In order to do so we need to enrich our notations and define new schemas and algorithms to transform object oriented databases into ContextMap model.

Also, more research may be conducted to improve the performance of the conversion from relational database to XML schema and vice versa.

CODAX may be expanded into a Web-based application to be used over the Internet. In order to do so we first need to implement Context+ environment into a Web-based application.

Finally we would like to add new features to CODAX, to perform data migration by implementing some database store procedures.

Chapter 8

References

- [Alh03] R. Alhajj. *Extracting the extended entity-relationship model from a legacy relational database*. In *Information Systems*, volume 28, pages 591-618. Elsevier Science Ltd., 2003.
- [BCFK99] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. *UML for XML Schema Mapping Specification*. Rational White Paper, December 1999.
- [BSN92] C. Batini, S. Seri, and S. Navathe, *Conceptual Database Design: An Entity Relationship Approach*. Benjamin Cummings, Redwood City, 1992.
- [CBS96] R. Chiang, T. Barron, and V. Storey. *A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases*. In *Data & Knowledge Engineering*, volume 21, pages 57-77, 1996.
- [CFIL00] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Schanmugasundaram, E. Shekita, and S. Subramanian. *XPERATO: Publishing object relational data as XML*. In *proc. of Int'l Workshop on the Web and Databases (WebDB)*, pages 105-110, 2000.

- [C01] D. Carlson. *Modeling XML Applications with UML*. Addison-Wesley, 2001.
- [CX00] J. Cheng and J. Xu. *XML and DB2*. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering*, pages 569-573, 2000.
- [DA87] K. Davis, and A. Arora, *Converting a Relational Database Model into an Entity-Relationship Model*. In *Proc of the 6th International Conference on Entity-Relationship Approach*, pages 243-257, 1987.
- [DHW01] D. Draper, A.Y. Halevy, and D.S. Weld. *The nimble XML data integration system*. In *Proc of ICDE*, pages 155-160, 2001.
- [DW00] D. Lee, W. Chu, *Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema*, ER 2000 19th International Conference on Conceptual Modeling, volume 1920, October 2000.
- [FPB01] J. Fong, F. Pang, and C. Bloor, *Converting relational database into XML document*. In *12th International Conference on Data Engineering*, pages 61-65, IEEE Computer Society Press, 2001.
- [J94] P. Johannesson, *A Method for Transforming Relational Schemas into Conceptual Schemas*. In *Proc. of the 10th International Conference on Data Engineering*, pages 190-201, IEEE Computer Society Press, 1994.
- [KL01] C. Kleiner and U. W. Lipeck, *Automatic generation of XML DTDs from conceptual database schemas*, In *Proc. of the Annual Conference of the German and Austrian Computer Societies in Web-Databases*, 2001.

- [LS01] Eric Simon, *Le Select, a Middleware System that Eases the Publication of Scientific Data Sets and Programs. Workshop on Information Integration on the Web, 2001.*
- [LMCC01] D. Lee, M. Mani, F. Chiu, and W. W. Chu, *Nesting based relational-to-XML schema translation. In Int'l Workshop on the Web and Databases (WebDB),* pages 61-66, 2001.
- [LMCC02] D. Lee, M. Mani, F. Chiu, and W. W. Chu. , *Translating relational schemas to XML schemas using semantic constraints. In 11th ACM Int'l Conf. on Information and Knowledge Management (CIKM),* pages 282-291, 2002.
- [MFKX00] I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, and D. Olteanu. *Agora: Living with XML and relational. In The VLDB Journal,* pages 623–626, 2000.
- [MLM01b] M. Mani, D. Lee, and R. R. Muntz., *Semantic data modeling using XML schemas. In 20th Int'l Conf. on Conceptual Modeling (ER),* pages 149-163, 2001.
- [MLM01a] M. Mani, D. Lee, and R. R. Muntz., *Taxonomy of XML schema language using formal language theory. In Philip S. Yu and Arbee S. P. Chen, editors, Extreme Markup Languages. IEEE Computer Society Press,* 2001.
- [MM90] V. Markowitz, and J. Makowsky, *Identifying Extended Entity-Relationship Object Structures in Relational Schemas. In IEEE transactions on Software Engineering,* Pages 777-790, 1990.

- [NA87] S. Navathe, and A. Awong, *Abstracting Relational and Hierarchical Data with a Semantic Data Model*. In *Proc of the 6th International Conference on the Entity-Relationship Approach*, pages 277-305, 1987.
- [NSLB] Northwind, Sample, Library and Biblio are trademarks of Microsoft Corp.
- [PB94] W. Premierlani, and M. Blaha, *An Approach for Reverse Engineering of Relational Databases*, volume 37, pages 42-49, 1994.
- [PTB96] J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Koulomdjian, *Towards the Reverse Engineering of Denormalized Relational Databases*. In *Proc of the 12th International Conference on Data engineering*, pages 218-227, 1996.
- [S98] C. Soutou, *Relational Database Reverse Engineering: Algorithms to extract cardinality Constraints*, in *Data & Knowledge Engineering, Volume 28*, pages 161-207, 1998.
- [SD02] Sihem AmerYahia, Divesh Srivastava, *A mapping schema and interface for XML stores*, Pages 23 - 30, 2002.
- [SSB01] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. *Efficiently publishing relational data as XML documents*. *The VLDB Journal*, Pages 133-154, 2001.
- [WLAB04] C. Wang, A. Lo, R. Alhajj, and K. Barker. *Converting legacy relational database into XML database through reverse engineering*. In *6th International Conference on Enterprise Information Systems (ICEIS)*, pages 216-221, 2004.

[WMJ99] Wojciech M. Jaworski, et al. *Representing processes, schemata and templates with jMaps*, *Semiotica* 125(1/3), Pages 229-47, 1999.

[WMJ95] Wojciech M. Jaworski, *Conceptual Spreadsheets for Data and Knowledge Warehousing*, ATW95, University of New Hampshire, Durham, New Hampshire, June 1995.

[WMJA94] Wojciech M. Jaworski, Michailidis A. A., *Recovery and Enhancement of System Patterns: InfoSchemata and InfoMaps*, University of Massachusetts - Lowell, Lowell Massachusetts, June 1994.

[WMJK02] Wojciech M. Jaworski, Kang Zhou, *CONTEXT+: Development Environment for 3P-able Context Maps*, Major Report Concordia University, July 2002.

[XR02] X. Zhang *et al.* *Rainbow: Mapping-Driven XQuery Processing System*, In *Proc. of the ACM SIGMOD Conf. on Management of Data*, page 614, 2002.

Appendices

Appendix-A: Database Schema Examples

Figure A-1 illustrates the Flat XML Schema Output for the NorthWind Database.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="Categories">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CategoryID">
          <xs:simpleType>
            <xs:restriction base="xs:integer">

          </xs:restriction>
          </xs:simpleType>
        </xs:element>

        <xs:element name="CategoryName">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="15"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>

        <xs:element ref="Products" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="PK_Categories">
      <xs:selector xpath="."/>
      <xs:field xpath="CategoryID"/>
    </xs:key>

    <xs:keyref name="FK_Products2" refer="PK_Categories">
      <xs:selector xpath="Products"/>
      <xs:field xpath="CategoryID"/>
    </xs:keyref>
  </xs:element>
  <xs:element name="Customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CustomerID">
          <xs:simpleType>
```

```

                <xs:restriction base="xs:string">
                    <xs:maxLength value="5"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="CompanyName">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="40"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="ContactName" nillable="true">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="30"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="ContactTitle" nillable="true">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="30"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Address" nillable="true">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="60"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="City" nillable="true">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="15"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        </xs:element>
        <xs:element ref="Orders" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:key name="PK_Customers">
    <xs:selector xpath="."/>
    <xs:field xpath="CustomerID"/>
</xs:key>
<xs:keyref name="FK_Orders1" refer="PK_Customers">
    <xs:selector xpath="Orders"/>
    <xs:field xpath="CustomerID"/>
</xs:keyref>
</xs:element>
<xs:element name="Employees">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="EmployeeID">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">

```

```

</xs:element>
<xs:element name="LastName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="FirstName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="Title" nillable="true">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="30"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="BirthDate" nillable="true">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="11"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="HireDate" nillable="true">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="11"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element ref="Orders" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:key name="PK_Employees">
  <xs:selector xpath="."/>
  <xs:field xpath="EmployeeID"/>
</xs:key>
<xs:keyref name="FK_Orders2" refer="PK_Employees">
  <xs:selector xpath="Orders"/>
  <xs:field xpath="EmployeeID"/>
</xs:keyref>
</xs:element>
<xs:element name="OrderDetails">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OrderID">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      <xs:element name="ProductID">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```

        </xs:element>
        <xs:element name="UnitPrice">
            <xs:simpleType>
                <xs:restriction base="xs:integer">

                    </xs:simpleType>
                </xs:element>
            <xs:element name="Quantity">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">

                        </xs:simpleType>
                    </xs:element>
                <xs:element name="Discount">
                    <xs:simpleType>
                        <xs:restriction base="xs:integer">

                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                <xs:key name="PK_OrderDetails">
                    <xs:selector xpath="."/>
                    <xs:field xpath="OrderID"/>
                    <xs:field xpath="ProductID"/>
                </xs:key>
            </xs:element>
        <xs:element name="Orders">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="OrderID">
                        <xs:simpleType>
                            <xs:restriction base="xs:integer">

                                </xs:simpleType>
                            </xs:element>
                        <xs:element name="CustomerID" nillable="true">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:maxLength value="5"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    <xs:element name="EmployeeID" nillable="true">
                        <xs:simpleType>
                            <xs:restriction base="xs:integer">

                                </xs:simpleType>
                            </xs:element>
                        <xs:element name="OrderDate" nillable="true">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:pattern value="11"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    <xs:element name="RequiredDate" nillable="true">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:pattern value="11"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

```

        </xs:element>

        <xs:element ref="OrderDetails" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:key name="PK_Orders">
    <xs:selector xpath="."/>
    <xs:field xpath="OrderID"/>
</xs:key>
<xs:keyref name="FK_OrderDetails1" refer="PK_Orders">
    <xs:selector xpath="OrderDetails"/>
    <xs:field xpath="OrderID"/>
</xs:keyref>
</xs:element>
<xs:element name="Products">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ProductID">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">

                        </xs:simpleType>
                    </xs:element>
                    <xs:element name="ProductName">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:maxLength value="40"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:element>
                    <xs:element name="SupplierID" nillable="true">
                        <xs:simpleType>
                            <xs:restriction base="xs:integer">

                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="CategoryID" nillable="true">
                                <xs:simpleType>
                                    <xs:restriction base="xs:integer">

                                        </xs:simpleType>
                                    </xs:element>
                                    <xs:element name="QuantityPerUnit" nillable="true">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:maxLength value="20"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:element>
                                    <xs:element name="UnitPrice" nillable="true">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:integer">

                                                </xs:simpleType>
                                            </xs:element>
                                            <xs:element name="UnitsInStock" nillable="true">
                                                <xs:simpleType>
                                                    <xs:restriction base="xs:integer">

                                                        </xs:simpleType>
                                                    </xs:element>
                                                    </xs:element>
                                                </xs:element>
                                            </xs:element>
                                        </xs:element>
                                    </xs:element>
                                </xs:element>
                            </xs:element>
                        </xs:element>
                    </xs:element>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

```



```

                                <xs:element ref="OrderDetails" minOccurs="0"
maxOccurs="unbounded"/>
                                </xs:sequence>
                                </xs:complexType>
                                <xs:key name="PK_Products">
                                    <xs:selector xpath="."/>
                                    <xs:field xpath="ProductID"/>
                                </xs:key>
                                <xs:keyref name="FK_OrderDetails2" refer="PK_Products">
                                    <xs:selector xpath="OrderDetails"/>
                                    <xs:field xpath="ProductID"/>
                                </xs:keyref>
                                </xs:element>
                                <xs:element name="Shippers">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="ShipperID">
                                                <xs:simpleType>
                                                    <xs:restriction base="xs:integer">
                                                        </xs:restriction>
                                                    </xs:simpleType>
                                                </xs:element>
                                            <xs:element name="CompanyName">
                                                <xs:simpleType>
                                                    <xs:restriction base="xs:string">
                                                        <xs:maxLength value="40"/>
                                                    </xs:restriction>
                                                </xs:simpleType>
                                            </xs:element>
                                            <xs:element ref="Orders" minOccurs="0" maxOccurs="unbounded"/>
                                        </xs:sequence>
                                    </xs:complexType>
                                    <xs:key name="PK_Shippers">
                                        <xs:selector xpath="."/>
                                        <xs:field xpath="ShipperID"/>
                                    </xs:key>
                                    <xs:keyref name="FK_Orders3" refer="PK_Shippers">
                                        <xs:selector xpath="Orders"/>
                                        <xs:field xpath="ShipperID"/>
                                    </xs:keyref>
                                </xs:element>
                                <xs:element name="Suppliers">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="SupplierID">
                                                <xs:simpleType>
                                                    <xs:restriction base="xs:integer">
                                                        </xs:restriction>
                                                    </xs:simpleType>
                                                </xs:element>
                                            <xs:element name="CompanyName">
                                                <xs:simpleType>
                                                    <xs:restriction base="xs:string">
                                                        <xs:maxLength value="40"/>
                                                    </xs:restriction>
                                                </xs:simpleType>
                                            </xs:element>
                                            <xs:element ref="Products" minOccurs="0" maxOccurs="unbounded"/>
                                        </xs:sequence>
                                    </xs:complexType>

```

```

        <xs:key name="PK_Suppliers">
            <xs:selector xpath="."/>
            <xs:field xpath="SupplierID"/>
        </xs:key>
        <xs:keyref name="FK_Products1" refer="PK_Suppliers">
            <xs:selector xpath="Products"/>
            <xs:field xpath="SupplierID"/>
        </xs:keyref>
    </xs:element>
</xs:schema>

```

Figure A-1 - XML Schema for the NorthWind Database

Figure A-2 illustrates the Oracle Relational Schema for the NorthWind [NSLB] database.

```

CREATE TABLE Products (
Product_ID Number ,
ProductName varchar2(40) ,
SupplierID Number ,
CategoryID Number ,
QuantityPerUnit varchar2(20) ,
UnitPrice Number ,
UnitsInStock Number ,
UnitsOnOrder Number ,
ReorderLevel Number ,
Discontinued varchar2(1)
)
TABLESPACE System
PCTFREE 10
PCTUSED 60
INITRANS 1
MAXTRANS 255
/

```

```

Alter Table Products Add Constraint PK_Products Primary Key (
Product_ID )
/

```

```

CREATE TABLE OrderDetails (
OrderID Number ,
ProductID Number ,
UnitPrice Number ,
Quantity Number ,
Discount Number
)
TABLESPACE System
PCTFREE 10
PCTUSED 40
INITRANS 1
MAXTRANS 155
/

```

```

Alter Table OrderDetails Add Constraint PK_OrderDetails Primary Key (
OrderID , ProductID )
/

```

```

CREATE TABLE Orders (

```

```

OrderID Number ,
CustomerID Number ,
EmployeeID Number ,
OrderDate Date ,
RequiredDate Date ,
ShippedDate Date ,
RequiredDate Date ,
ShipVia Number ,
Freight Number ,
ShipName varchar2(40) ,
ShipAddress varchar2(60) ,
ShipCity varchar2(15) ,
ShipRegion varchar2(15) ,
ShipPostalCode varchar2(10) ,
ShipCountry varchar2(15)
)
TABLESPACE User_Data
PCTFREE 10
PCTUSED 50
INITRANS 1
MAXTRANS 200
/

```

```

Alter Table Orders Add Constraint PK_Orders Primary Key ( OrderID )
/

```

```

CREATE TABLE Customers (
CustomerID Number ,
CompanyName varchar2(40) ,
ContactName varchar2(30) ,
ContactTitle varchar2(30) ,
Address varchar2(60) ,
City varchar2(15) ,
Region varchar2(15) ,
PostalCode varchar2(10) ,
Country varchar2(15) ,
Phone varchar2(24) ,
Fax varchar2(24)
)
TABLESPACE User_Data
PCTFREE 10
PCTUSED 70
INITRANS 1
MAXTRANS 355
/

```

```

Alter Table Customers Add Constraint PK_Customers Primary Key (
CustomerID )
/

```

```

CREATE TABLE Suppliers (
SupplierID Number ,
CompanyName varchar2(40) ,
ContactName varchar2(30) ,
ContactTitle varchar2(30) ,
Address varchar2(60) ,
City varchar2(15) ,
Region varchar2(15) ,
PostalCode varchar2(10) ,
Country varchar2(15) ,
Phone varchar2(24) ,
Fax varchar2(24) ,

```

```

HomePage varchar2(500)
)
TABLESPACE User_Data
PCTFREE 10
PCTUSED 50
INITRANS 1
MAXTRANS 200
/

```

```

Alter Table Suppliers Add Constraint PK_Suppliers Primary Key (
SupplierID )
/

```

```

CREATE TABLE Categories (
CategoryID Number ,
CategoryName varchar2(500) ,
Description varchar2(4000) ,
Picture varchar2(4000)
)
TABLESPACE User_Data
PCTFREE 10
PCTUSED 50
INITRANS 1
MAXTRANS 200
/

```

```

Alter Table Categories Add Constraint PK_Categories Primary Key (
CategoryID )

```

```

CREATE TABLE Employees (
EmployeeID Number ,
LastName varchar2(20) ,
FirstName varchar2(10) ,
Title varchar2(30) ,
TitleOfCourtesy varchar2(25) ,
BirthDate Date ,
HireDate Date ,
Address varchar2(60) ,
City varchar2(15) ,
Region varchar2(15) ,
PostalCode varchar2(10) ,
Country varchar2(15) ,
HomePhone varchar2(24) ,
Extension varchar2(4) ,
Photo varchar2(4000) ,
Notes varchar2(4000) ,
ReportsTo Number
)
TABLESPACE System
PCTFREE 10
PCTUSED 60
INITRANS 1
MAXTRANS 255
/

```

```

Alter Table Employees Add Constraint PK_Employees Primary Key (
EmployeeID )

```

```

CREATE TABLE Shippers (
ShipperID Number ,
CompanyName varchar2(40) ,
Phone varchar2(24)
)

```

```

)
TABLESPACE System
PCTFREE 10
PCTUSED 60
INITRANS 1
MAXTRANS 255
/

Alter Table Shippers Add Constraint PK_Shippers Primary Key (
ShipperID)

Alter Table OrderDetails Add Constraint FK_OrderDetails1 Foreign Key
( OrderID ) References Orders (OrderID)
/
Alter Table OrderDetails Add Constraint FK_OrderDetails2 Foreign Key
(ProductID ) References Products ( ProductID)
/
Alter Table Orders Add Constraint FK_Orders1 Foreign Key (
CustomerID ) References Customers ( CustomerID )
/

Alter Table Orders Add Constraint FK_Orders2 Foreign Key (
EmployeeID ) References Employees ( EmployeeID )
/

Alter Table Orders Add Constraint FK_Orders3 Foreign Key ( ShipperID
) References Shippers ( ShipperID )
/

Alter Table Products Add Constraint FK_Products1 Foreign Key (
SupplierID ) References Suppliers ( SupplierID )
/

Alter Table Products Add Constraint FK_Products2 Foreign Key (
CategoryID ) References Categories ( CategoryID )
/

```

Figure A-2 - Oracle relational schema for the NorthWind database

Appendix-B: CODAX Implementation Specification

The following figure lists all forms and their corresponding classes.

<i>Form Name</i>	<i>Related Classes</i>	<i>Calling Forms</i>
FrmMainRDBtoXML	Mstartup	FrmCCTXLRR
	Mshow	FrmCTX2XML
FrmCCTXLRR	Mshow	
	MimportDBWKS	
	MGeneralFunctions	
	MGeneralExcelFunctions	
	MApplyColor	
	MCardinality	
	MGeneralCTXFunctions	
	MGlobals	
	MGroupMap	
FrmCTX2XML	Mshow	
	SqlTable	
	MexportDB	
	MGeneralFunctions	
FrmMainXMLtoRDB	MStartup	FrmCCTXXML
	Mshow	FrmCTX2RDB
FrmCCTXXML	Mshow	
	MxmlToCtxMap	
	MGeneralFunctions	
	MGeneralExcelFunctions	
	MApplyColor	

	MCardinality	
	MGeneralCTXFunctions	
	MGlobals	
	MGroupMap	
FrmCTX2RDB	Mshow	
	SqlTable	
	MexportRDB	
	MGeneralFunctions	

Figure A-3 - User interface and class specification

Appendix-C: ContextMap Terminology and Notation

C-1: ContextMap Notation

ContextMaps notation is illustrated in the following figure [WMJA94]:

<i>Category</i>	<i>Name</i>	<i>Description</i>
Notation of Set Roles	A	(A)ggregation of columns - context tuples
	E	- (E)dge properties
	F	- (F)low graph nodes
	L	- (L)flow graph with cycles
	N	- (N)ode properties
	V	- (V)alue
	S	- (S)equence
	G	- (G)uard
	R	- (R)esource
	O	- (O)bject
	I	- (I)dentifier
	X	- Cartesian Product
	?	- unknown
Notation of Set Member Roles	v	- marker
	?	- unknown
	m	- (m)iddle of 'arrow'
	f	- tail of 'arrow'
	t	- head of 'arrow'
	b	- both f/t
	F	- (f)rom node

<i>Category</i>	<i>Name</i>	<i>Description</i>
	t	- (t)o node
	l	- (l)oop
	b	- f/t - both nodes component
	f	- (f)rom node component
	t	- (t)o node component
	l	- (l)oop node component
	y	- yes
	o	- otherwise
	r	- (r)ead
	u	- (u)pdate
	d	- (d)elele
	x	- component of Cartesian Product
	c	- concurrence
	j	- (J)oin from fork
	k	- (K)ey

Figure A-4 - ContextMaps notation

C-2: 4P-able Capability

4P-able notation is a technology for representing enterprise architectures, static and dynamic structures, templates, and schemas for system processes and artifacts. The library of this notation acts as a knowledge repository. The most important character of *ContextMaps* is 4P-able notation, can be illustrated as the following formula [WMJK02]:

4P-able = Plug-able + Process-able + Pattern search-able + Perform-able

- **Plug-able:** Merge-able

ContextMap is a collection of different pieces of knowledge connected together in a logical way. Many scattered elements and relationships among the concepts in software engineering process can be integrated to one *ContextMap*, so that a formatted and clear view is presented for users. Based on *ContextMap* technology, separate knowledge can be merged into one view horizontally and vertically. The “Join Maps” function of the CONTEXT+ was developed to meet this requirement.

- **Process-able:** Create-able, Read-able, Update-able, Delete-able

By using different syntax in the spreadsheet, it is feasible and convenient to describe and process conceptual knowledge. The letters “c”, “r”, “u”, and “d” stand for “create”, “read”, “update”, and “delete” in software engineering process correspondingly.

- **Pattern able:** Search-able, Navigate-able

The *ContextMap* is a kind of notational technology. This notational pattern strongly supports search and navigation. With the use of spreadsheet structure, a large amount of data can be organized logically. One can simplify the procedure in processing ContextMaps. by using the custom-make Query of the CONTEXT+; it is convenient to get the specific knowledge that users expect to search from the map.

- **Perform-able:** Execute-able, Run-able

A *ContextMap* consist of a notational map on the left side and the State, Precondition, Action, Data Object etc. on the right side. Using the map, the performance tools of *ContextMap* execute the source code in the same way as a traditional program is executed, and displays the output.

C-3: Context+ Functionalities

The main functionalities of the Context+ are listed below:

Mode Section Functionality

- **“Visible Map”** - is used to display the map without hidden parts.
- **“Select Sets”** - is used to select the specific sets for viewing, and it will display the map only with selected sets.
- **“Select Maps”** - is used to display all the available worksheets in the current workbook and have some of these worksheets perform “By Color” the operation simultaneously. The result of selected maps will be opened in the new sheet with the sheet name “original sheet name + Result”.
- **“Join Maps”** - is used to merge one or more selected maps together in one map. The criteria: comparing the values of concept column within each selected worksheet, join the columns having the same values of concept. Or add additional rows with different values of concept. Then save the result into a separate sheet named “Merge Result”. This function can be implemented independently.
- **“Add Atom”** - is used to insert a Row into all spreadsheets of the Active Workbook.

Query Section Functionality

- **“By Color”** - this button will be implemented with one of ‘AND’, ‘XOR’, ‘OR’, ‘NOT’ operations based on the selected predefined - query. The predefined - query should be marked in three places in the Context Maps.
- **“AND”** - in order to perform an “AND” query, select at least two cells from different rows to generate corresponding map, which remains all the not empty columns within these rows.
- **“XOR”** - in order to perform an “XOR” query, select at least two or multiple cells from different rows to generate corresponding map, which remains the exclusive not empty columns within these rows.
- **“OR”** - in order to perform an “OR” query, select at least one or multiple cells from different rows to generate corresponding map, which retains the each of the not empty columns from within these rows.
- **“NOT”** - in order to perform a “NOT” query, select only one cell to retrieve negation columns and generate corresponding map.

Output Section Functionality

- **“Schema”** - is used to extract the schema from the map.
- **“Cardinality”** - is used to display the number of non empty roles of each row and the number of sub-concept values of each concept set. It helps us to see the complexity of each set.
- **“ApplyColor”** - is used to apply a different color in each cell of the spreadsheet based on the following criteria.

- **“Map”, “Graph”,and ”Map & Graph”** - is used to present views, all outputs are in ContextMap formats. But graph format is recommended for future work.

Output Section Functionality

- **“Group (Row and Col)”** - is used to view the whole map in a grouped way, each range of group is based on each range of set.
- **“Ungroup (Row and Col)”**- is used to view the whole map in an ungrouped way.
- **“Help”**- is used to display help.