XML DATA TRANSFER BETWEEN HETEROGENEOUS DATABASE SYSTEMS

QINGHONG LIN

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

SEPTEMBER 2003

0-612-85278-4

ABSTRACT

XML Data Transfer Between Heterogeneous Database Systems

Qinghong Lin

This major report presents the design and implementation of XML data transfer, a tool for transferring data between heterogeneous data sources by using XML standards and the related technology.

Relational database design focuses on decomposing a flat data structure to a normalized relational model. In reality, the real world objects can be modeled through many different ways as long as they represent the objects and relationship correctly.

In today's software development world, multiple vendors have been implementing their own data model through practising the relational theory. Therefore, the data model dealing with the same real world objects are often different. The data exchange between these different software systems is extremely difficult due to the difference of the data models. To resolve the data exchange problem in integration and information sharing among different software systems, XML (Extensible Markup Language) becomes a natural selection as a means to relay the data in the information exchange process.

In this report, data is transferred between two sample school administration database systems, which are semantically similar, but have different data models (i.e. schema representations). The transferring is carried out through an intermediate data model as the bridge between these two data sources. The intermediate data model is in XML data format. The mappings are via two XML schemas, one from the source school database to the intermediate data file, another is from the intermediate to the destination school database. The programs are developed using Microsoft .NET and C# programming language. The databases are built on Microsoft SQL Server.

Acknowledgements

I would like to express my special gratitude to all the people who gave me great help during this major report.

- I am greatly indebted to my supervisor, Professor Stan Klasa, for acceptance to be my supervisor. He gave me kindly understanding, support and encouragement during this major report. With his advice and continuous help, I was able to complete this major report successfully.

- My sincere thanks are extended to Professor Adam Krzyzak, who is an examiner for the report, for his valuable time of reviewing the report, and making suggestions.

- My special thanks are also due to Ms. Halina Monkiewicz, the Graduate Program Secretary, for her collaboration and kindly help.

Finally, I would like to express my appreciation to the faculty and staff in the Computer Science Department at Concordia University, who provided large support during my master program study.

Table of Contents

List of Figures vii

List of Tables viii

1. Introduction 1

1.1 Background 1

1.2 Outline of Report 6

1.3 Objective of Study 7

1.4 Procedure of Research 8

2. Design Rationale and Architecture 10

2.1 Design Rationale 10

2.2 System Architecture 14

2.3 Data Flow 15

3. Detailed Design 17

3.1 Database Design 17

3.2 XML to Present the Middle-tier Data Model 20

3.3 XML Schema to validate and translate the data model 24

4. Program Specification 32

4.1 Program Design Overview 32

4.2 Specification of Schema Module 32

4.3 Specification of ExportToXml Module 44

4.4 Specification of XmlToDatabase Module 45

5. Conclusions and Recommendations 48

5.1 Conclusions 48

5.2 Recommendations for Future Works 48

Bibliography 49

Appendices 50

A. Database schemas 50

B. XML Schemas 53

C. Program Source Code 60

List of Figures

Figure 1 Denormalized Structure Serves as the Middle-tier Data Model 12

Figure 2 System Architecture of the Project 14

Figure 3 Data Flow of the Transferring Process 15

Figure 4 School_A in Relational Data Model 18

Figure 5 School_B in Relational Data Model 19

Figure 6 Middle-tier Object Model 20

Figure 7 Mapping Between Source and Middle-tier Object Model 23

List of Tables

Table 1 Objects and Attributes ... 21

Table 2 Data Represented in XML Format ... 22

Table 3 Represent Relational Table in XML Schema ... 24

Table 4 Represent Foreign Key Reference in XML Schema ... 25

Table 5 Represent reference to tables in XML Schema ... 26

Table 6 Use of Directive in XML Schema ... 27

Table 7 Represent Inheritance in XML Schema ... 28

Table 8 Represent Optional Data in XML Schema ... 29

Table 9 Application-defined attributes in XML Schema ... 30

Chapter 1

1. Introduction

1.1 Background

The problem

Today, with ever growing data and applications, sharing, integration, and redistribution data and information is an important issue. Thus, it is often required to transfer data between different data sources. Some common scenarios are, we try to integrate multiple applications and database systems, or an application gathers information from diverse data sources, or data needs to be migrated from an old database to a newer version after schema reconstruction or application upgrade, or reuse data from some legacy databases, etc. In these enumerated situations, it might be in the case of sharing same data models, or in the case of using different ones. This project will focus on the later case, when we are in the situation of dealing with different data models, when we need to share and transfer data between diverse data sources.

A schema is the organization or structure for a database. It represents a data model, which reflects real world objects and their relationship with each other. A real world problem is analysed and modeled by different groups of people, it is likely to produce different results, yields different data models. So, it normally results in schemas organized in various ways, which means schema implementations can be semantically similar, but have different physical presentations. For example, two school databases, school A and school B, both of them are dealing with school administration information, such as students and faculty records, registration and tuition payments, class schedules, and other related information. But each schema can name their tables, columns differently; can have relations, key constraints differently; same attributes can have different data types, default values, and constraints; can have composition of two or more attributes to one attribute conflicts, and so on. This brings the problem when we want to share and reuse information. It is very useful and important to find a way to match between the schemas, yet is efficient, and easy to process.

This project addresses the experiment of finding an efficient matching method to bridge two heterogenous schemas, so to be able to transfer data and share information between them. To demonstrate the approach, two heterogenous sample schemas for managing school information are designed. The project discusses the rationale of the design of the method, illustrates the matching model, and implements the data transferring between the two school databases by using the matching model.

The middle-tier data model

In order to solve this data transferring between heterogenous schemas problem, a possible solution is to build a middle-tier data model as a bridge between the transferring and receiving side. The middle-tier data model complies with the semantic meaning of this specific application. It is responsible for the mapping from the source to the destination schema, providing a representation of the relevant data relationship on the transportation pipeline. It also serves as the storage place for the data to be transferred.

Relational database design focuses on decomposing a flat data structure to a normalized relational model. Serving as a bridge between two data sources, the middle-tier data model first assembles a denormalized view of the information from the source database, normally a relational database, and then decomposes to re-assemble a normalized schema for the transferring to the target data source, usually a relational database as well.

The middle-tier data model organizes the data in an object centric view. It describes the structure of the related objects according to the semantic meaning of the application, as well as the relationships among these objects. This middle-tier data model is represented by XML (Extensible Markup Language) standards, in particular, an XML data file, and two XML Schemas, each of which responsible for matching and extracting data from the source data source, and mapping and transferring the data to the destined data source.

XML and XML Schema

XML, a W3C (World Wide Web Consortium) endorsed standard, is about describing and formatting data. It defines a generic syntax used to mark up data with simple, human-readable tags. It can contain other elements, vary the order, and the number of attributes, and contain multiple elements with the same element type, etc. XML is reliable and flexible, and extremely broad-structuring standard. With this nature, XML can be completely customized to fit each different type of data being represented, its format can be customized for different domains. It is widely adopted in different applications.

The middle-tier data model denormalizes information from the source database; XML data format is hierarchical in nature, lends itself well to modeling denormalized views of information it conveys. Also, the middle-tier data model is object oriented; XML has some characters that are similar to objects in OOP, such as element content, subtypes, named attributes, and can represent hierarchical structures. Besides these powerful and expressive features, it is light, easy to process, platform-independent. Hence, XML is a suitable means to represent this middle-tier data model, to store and relay the data in the information exchange process.

XML clearly separates data and schema. XML is data, XML Schema describes the content, defines the structure, formalizes the constraints, gives semantics of XML data. It is the annotation to the XML data; it makes the data understood by humans and applications. By using schema components, XML Schema defines the rule, and relationships of all the constituent parts including datatypes, elements and their content and attributes and their values. The schema restricts the presence and sequence of elements, decides the cardinality of elements, specifies if the element contains data or is empty, and describes the hierachical relationships among elements. It also provides other information, such as normalization and defaulting of attribute and element values, and so on.

Among lots of other advantages, one important character of XML Schema is its extensibility. Schemas can be reused in other schemas. It can create new vocabularies, and is extensible to datatypes by deriving from the predefined datatypes, or creating new simple and complex datatypes. Also, several schemas can be easily combined to use for one XML data file. XML schema is a schema, it servers both roles of definition and validation, so the schema has to be carefully designed and written.

In this project, XML Schema along with XML data file is used to build the middle-tier data model. The schemas define the rules to denormalize the information from the source database, and also give the rules to re-assemble the information to be relayed to the destination database. For the nature of this database transferring application, it conveys information such as, names of tables, names of columns, dependant relationships among tables and columns, key constraints, data types, and cardinalities, default values, etc. The schemas also depict the mapping between these two data models.

XML and XML Schema serve good for representing the middle-tier data model, and storing data, but it requires a programming language to connect to different data sources, to access and corrperate between each transferring process and models, to control and monitor the data transferring procedures, to manipulate and match the data. Further more, a good programming language is powerful to develop other features, such as a user-friendly interface to be able to make configurations for the parameters related to the transferring, such as changing data locations, changing schemas and mapping rules, and maybe automating the transferring process, etc. In this Major Report project, this task is carried out on the platform of Microsoft .NET and implemented by C# programming language.

Microsoft .NET and C#

Microsoft .NET provides an integrated, standard-based, multi-language development environment. The .NET base class libraries provide strong support for XML within the namespace of System.Xml. The vast sets of classes cover range from XML data manipulation, query, navigation, schema validation, presentation style transformation, to lots of other XML related functionalities.

Microsoft .NET's new generation language C# inherited many features from both Java and C++ language. It accomplishes design goals such as, provides a unified type system and simplifying the way that value and reference types are used, establishes a component-based design, implements features like safe pointer manipulation, overflow checking, robust exception handling, and lots more.

Microsoft SQL SERVER

XML data transferring tool is suitable to transfer data stored in all sorts of database management systems. This Major Report uses Microsoft SQL SERVER as the data storage place for the sample databases is only for demonstration purpose.

1.2 Outline of Report

This report designs and implements a tool for data transferring between two heterogenous schemas using XML standards.

- Chapter 1 – gives the background introduction of this project, describes the objective of study, the choice of the technology and development environment, and the research and implementation procedure of the project.

- Chapter 2 – explains the design rationale behind this project, and describes the overall architecture design of the system, and shows the data flow inside the transferring process.
- Chapter 3 – presents the design of the project in detail, including the schemas of the data sources, the middle-tier data model, the XML schemas for describing and structuring different data sources, and also shows the steps to decide the semantics of the schemas and make the proper mapping.
- Chapter 4 – gives the specification of the program, describes the relationship and functionalities of each module, explains the classes inside each module and its major data members, methods, and properties.
- Chapter 5 – gives some discussion of this project, and makes conclusion and recommendations for the future work.

1.3 Objective of Study

The main objective of this Major Report is to study one of the important issues in the application integration domain, data transferring between heterogenous schemas. This project tries to design and implement a tool for this problem by utilizing the widely adopted XML technologies and the recently popular software suite, Microsoft .NET development platform.

The goal is to find a reliable and flexible, yet efficient method to carry out the data transferring task between heterogeous schemas. To build a middle-tier data model in between the transferring two sides is proposed, and is implemented by utilizing XML and XML Schema standards. The rationale and benefits of the design are examined during this development process.

The programs of this project are developed in Microsoft Visual Studio .NET development environment, and written in C# programming language. It opens an opportunity to explore some .NET technology, and along with their new generation programming language C#, providing a lot of hands-on experience.

1.4 Procedure of Research

The research work of this Major Report was supervised by Professor Stan Klasa. It was started in the winter of 2002.

The procedure of this Major Report is progressed in the following steps:

1) Decide the goal of this project, analyze the problem, research and read some related articles in this domain.
2) Learn the syntax rules of XML and XML Schema, try out some examples to help better understanding the meaning and usage of this data format standards.
3) Start to design the architecture of the project, make a draft of the middle-tier data model, decide what objects should be included in the XML data file, and how should the XML Schema files be organized.
4) Decide the data flow of the application, make up two heterogenous data models of school administration systems as the examples, and generate the schemas and data for these two databases.
5) Get familiar with Microsoft .NET development environment – Visual Studio .NET, learn C# programming language, and some of the classes from the .NET class library.
6) Install Microsoft SQL Server, create two school databases, and insert some data.

7) Work on the implementation of the actual data transfer runtime process.

8) Testing, and bug fixing of the programs.

9) Make adjustment, implementation, and testing and fixing problems in iterative cycles.

10) Draw conclusions for this research work, and write the report for the project.

Chapter 2

2. Design Rationale and Architecture

2.1 Design Rationale

There are many ways to implement data transferring between applications or database systems. In some cases, transferring tools are specially designed and built for the application, so the data transferring is limited to the specific data schemas; in some other cases, the tool is more generic, capable of carrying out the transferring tasks among a veriaty of applications or data sources, and can be configured to meet new requirements, even to reuse in different application domains.

The principle of steps needed for data transferring are extracting data from the source, and then importing data into the destination. If the schemas are same, then the process is straightforward. But in reality, the problem could be complicated when in the situation of transferring data between heterogenous schemas from different applications. The challenge is to find a matching mechanism from the source to the target schema, to establish a proper way of data mapping, and possibly change, merge and coalesce the data. It can be imagined, the matching cases between some schemas could be quite complex, and the mapping rules could be numerous. Therefore, it requires a good understanding of the semantical meaning of the systems and schemas involved, and also a tool for modeling and finding their logical relationship, and the semantical connection inside the data.

The important design issue needs to be considered is the flexibility and reusability of the tool. To achieve the goal, the design could be to save the SQL commands to a file or write stored procedures (if the DBMS where the data resides has the feature). When the requirements or schemas are changed, then the commands and stored procedures need to be re-written, or updated accordingly. If the SQL commands or calls to the stored procedures are embedded inside the application programs, then rewriting or making modifications to the programs is inevitable. The tool needs to have a module to access and configure the parameters related to the data transferring process, for example, to change of data sources, data format or type, constraints; to decide which tables or relations are involved in the source database, and how data should be organized or tailored to match the schema of the receiving data side, etc. Usually it provides the users to access, configure the parameters through an user interface. The tool will get complicated when the number and the diversity of the applications of sharing the tool increases.

Another possible solution, which is addressed and implemented in this project, is to use an intermediate data model as the middle data storage in the transferring process, making the exporting and importing data two independent steps. The intermediate data model not only stores data, but also makes the schema transformation. There are some advantages of this design of the architecture. The procedure of the transferring process is more clearly separated, first is exporting data from the source to the middle storage, and then importing from the middle to the destination data source. The intermediate model is a bridge of two different schemas. The proper design of the intermediate data model brings flexibility, making the schema transformation smoother, making the whole process easy to manage, and making it possible to reuse and share with other systems. During the exporting process, normalized schema is denormalized to a flat structure, and when the importing process starts, the data is reassembled according to the schema of the target database, which normally is a relational schema, so the flat data structure is normalized based on the structure of the target database. Figure 1 illustrates this normalize-denormalize transforming processes.

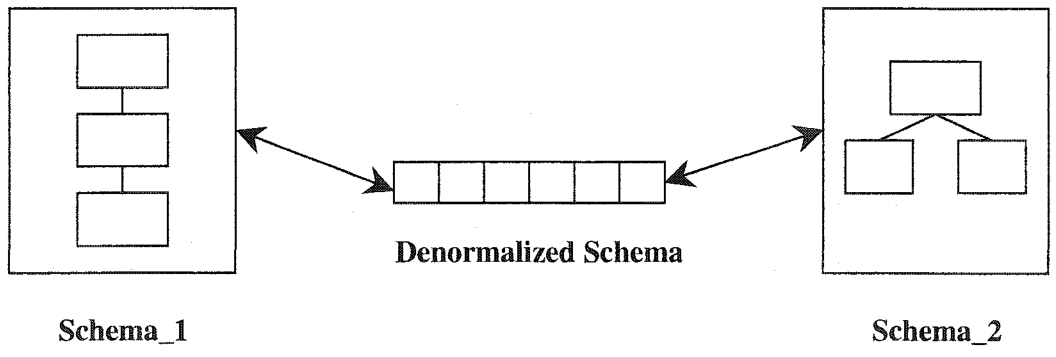


Figure 1 Denormalized Structure Serves as the Middle-tier Data Model

In Figure 1, Schema_1 and Schema_2 are relational schemas, representing logical table views, as the Denormalized Schema is the middle-tier data model, which consists of flat structured data from the relational tables.

There are different decisions to make for the choice of the intermediate data model. Depending on the nature of the application where the data transferring happens, the intermediate data model could be implemented with a database system, or simply to use a data file to store the data.

Database system is powerful to store and manipulate data, but on the other side, it is cumbersome. Data file is simple, but data is organized in sequence, which lacks of flexibility, and expressiveness. The becoming widely accepted and supported standard XML data format comes into the picture. It is light as a data file, but powerful in structuring and describing data. It is a platform-neutral data format file, easy to create, to maintain and share among internal and external systems.

XML organizes data in hierarchical orders, which lends well to model denormalized views of data. It is well suited for storing and caching data for the middle-tier data model of this tool. When data from more than one source is aggregated into a denormalized view, XML becomes especially attractive – XML provides a single data model and type system for data coming from heterogeneous systems.

XML document contains data, XML Schema describes XML document. It defines vocabularies and structure. The vocabulary is extensible, components can be easily reused to create other schemas. The language is extensible and flexible, yet it also serves a role to restrict and validate XML documents by using predefined data types or creating new data types, restricting the sequence of the data etc. This project demonstrates some of the features and benefits of this powerful data formatting and structuring standards.

2.2 System Architecture

The figure 2 below illustrates the system architecture of the project. Assume there are two heterogenous database systems: School_A and School_B, representatively, and data needs to be transferred from one to the other.

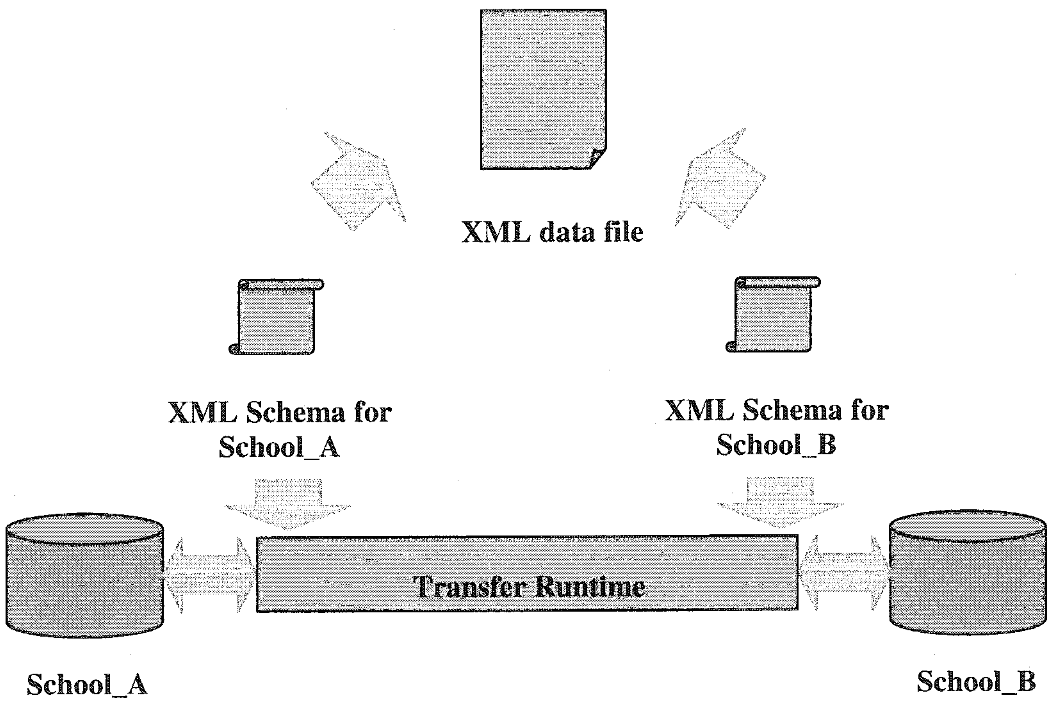


Figure 2 System Architecture of the Project

As illustrated in Figure 2 above, the system includes following components:

- XML data file serves as intermediate data storage for the transferring process. Data is denormalized, and reorganized into hierachical view.
- XML Schema (XSD), which defines the structure of the middle-tier object model, provides mapping between database and middle-tier data model. There

are two XML Schema files, one is used for extracting data from the source database, and the other is for importing data into the target database.

- The transfer runtime implements the actual transferring process. The main functionalities it provides include: 1) expose the schema from relational tables, 2) extract data from a database into a XML document, 3) import data into the target database, and 4) validate the XML data file and XML schemas, map relational schema to/from the middle-tier data model.

2.3 Data Flow

The figure 3 here shows the data flow in the transferring process:

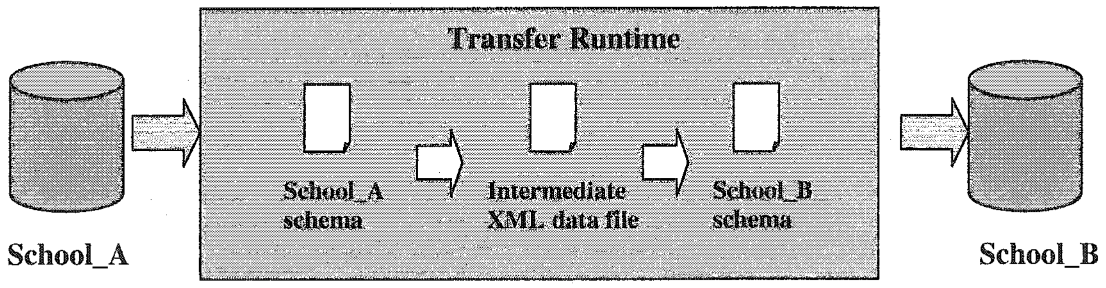


Figure 3 Data Flow of the Transferring Process

First of all, the data is extracted from the source database, School_A, and transported into an XML document whose schema is a mapping from relational schema of School_A to a hierachical-structured data model. This process denormalizes the relational tables according to the XML Schema for School_A database. The objects in the intermediate data file are based on the semantical meaning of the application, which in this project, are students, faculty members, courses, transcripts, organizations, salary payments, and addresses information. Data is temporarily stored in the XML data file. Then, the transfer runtime can easily import the data from this XML data file into the destination - School_B database with the mapping reference to the XML Schema for School_B. In this step, the mapping is from the middle-tier denormalized data model to the relational model.

Chapter 3

3. Detailed Design

3.1 Database Design

In this project, it is assumed data needs to be transferred between two school administration databases, School_A and School_B. Assume, two school database schemas are semantically similar - both of them have stored information such as students, faculty, departments, courses and transcripts, salary payments, etc., but their data models are different as can see from their data models presented in figures below (To reduce the problem scope, two school data models are only the partial and simplified versions of the real-world data models). The data model of School_A is showed in figure 4, and School_B showed in figure 5 (Column names in bold font are the primary keys, and in italic font are the foreign keys).

This project designs the data transferring tool working in the batch mode, so in the target database, School_B, each table has a row_id column to uniquely identify each row of the table. This design is to avoid conflict when inserting rows into tables, otherwise the verification before insertion should be performed, and the resolution routine has to be provided to handle the conflict. This tool enforces the data constraints before the data importing actually starts by using the power of XML Schema syntax. XML Schema file not only maps data between intermediate data model and target database, but also imposes the data constraints. During the transferring runtime process, the validation is performed before the XML data is loaded in.

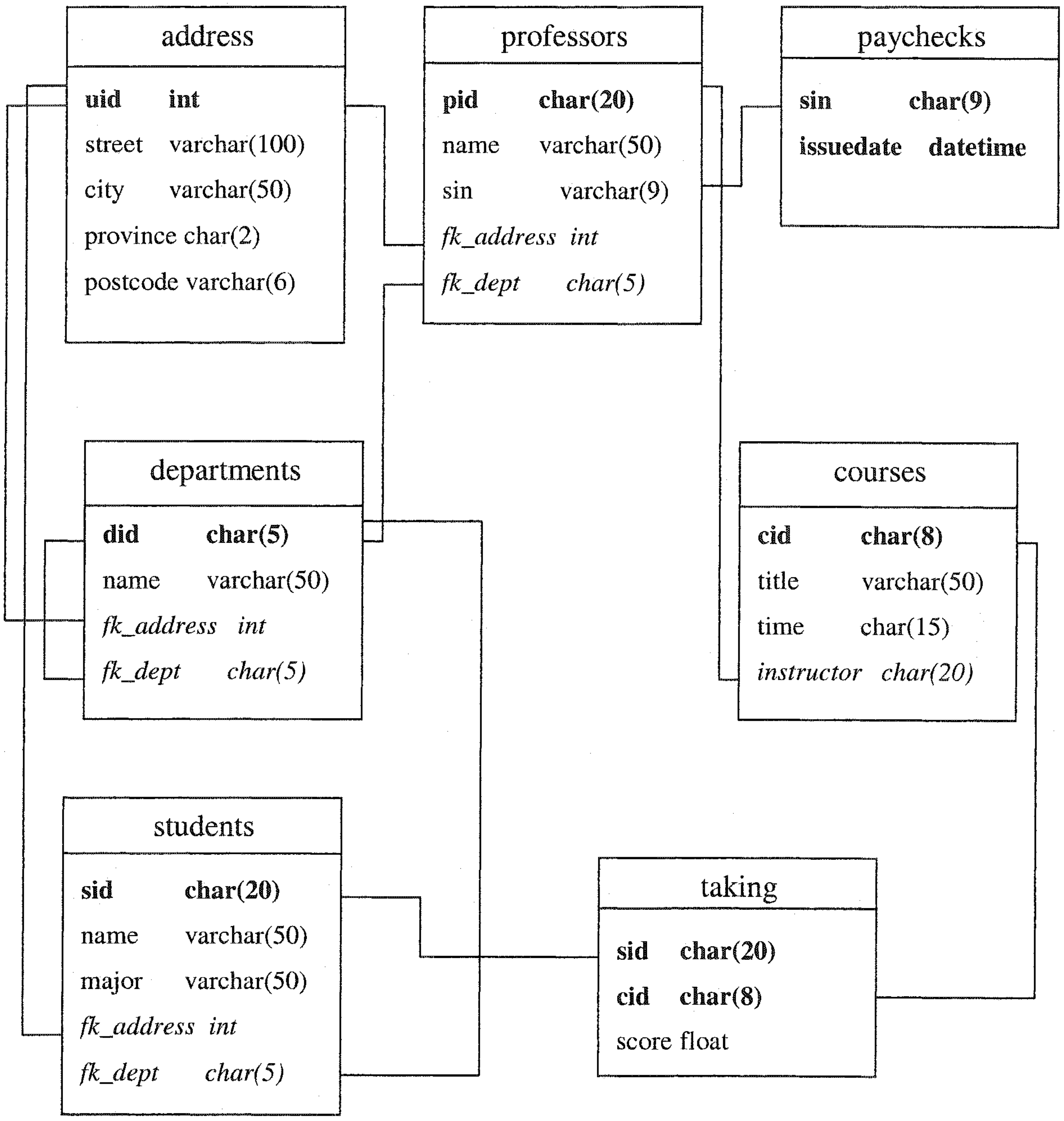


Figure 4 School_A in Relational Data Model

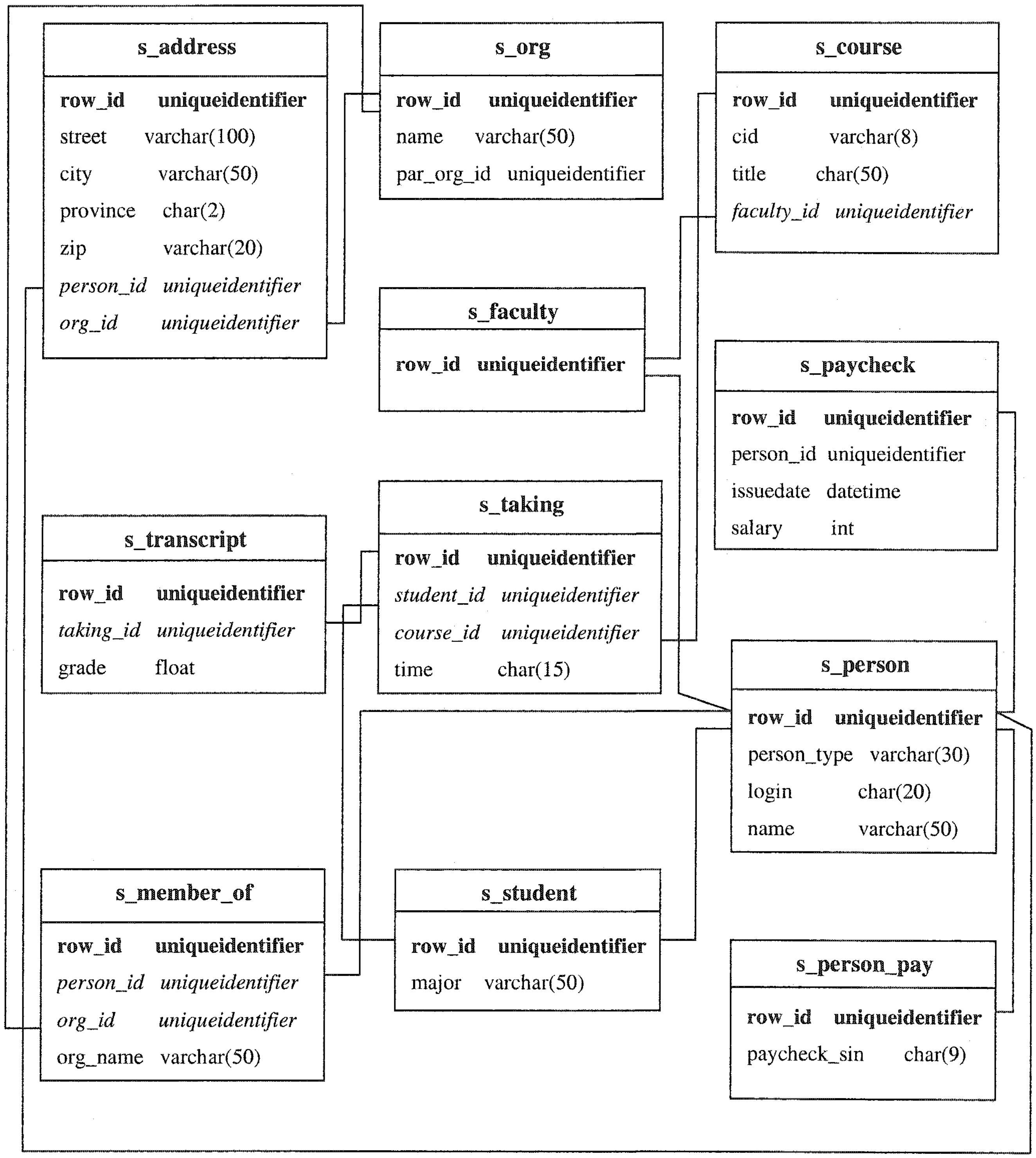


Figure 5 School_B in Relational Data Model

3.2 XML to Present the Middle-tier Data Model

According to the semantical meaning of the database schema and the transferring requirement, the middle-tier data model consists of five main objects, each object may have sub-objects. The object hierarchical data model is showed in figure 6.

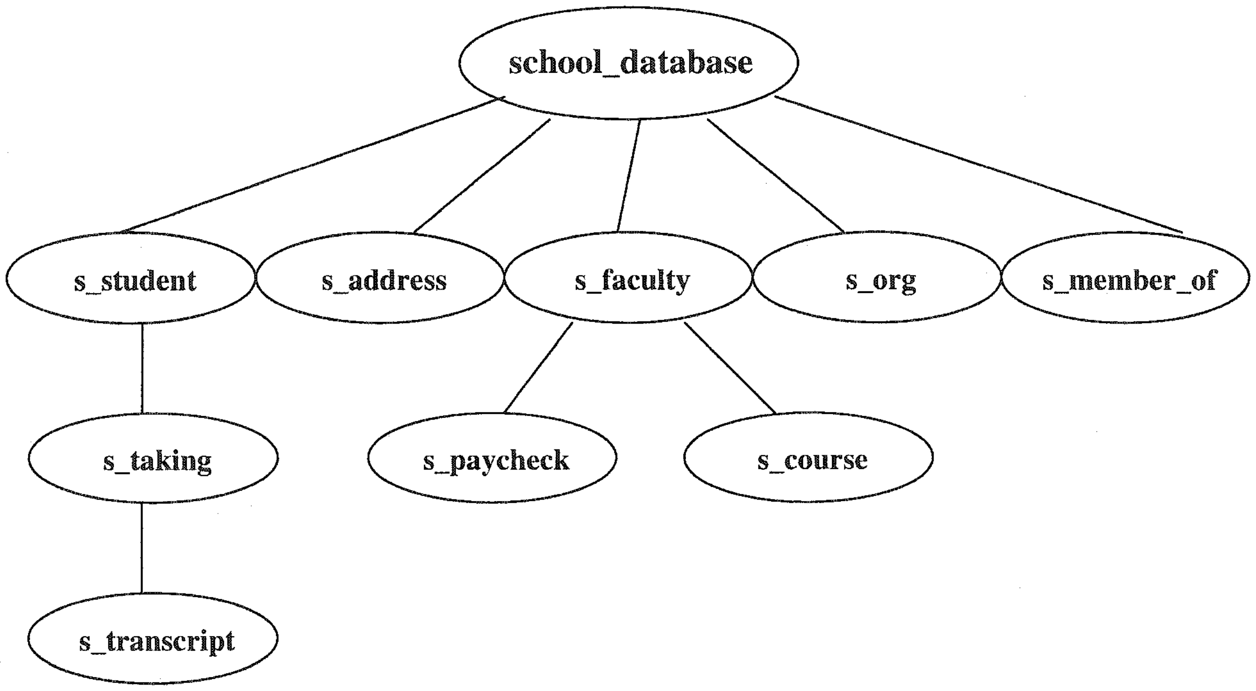


Figure 6 Middle-tier Object Model

Objects and their attributes are listed in Table 1 below. The objects and attributes can be designed differently, if so, the XML schema for mapping needs to be changed accordingly. Table 1 shows the objects and attributes are based on the schema of target database – School_B.

Table 1 Objects and Attributes

Object Name	Attribute Name
s_student	login, name, major
s_taking	course_id, time
s_transcript	grade
s_faculty	login, name, paycheck_sin
s_paycheck	salary, issuedate
s_course	cid, title
s_address	street, city, province, zip, person_id
s_org	name, par_org_id
s_member_of	person_id, org_id, org_name

XML uses matching start tag and end tag to markup elements - the primary unit of XML data format. Tags are self-defined. XML elements can have attributes in the start tag. Attributes are used to provide additional information about elements. Data is stored in elements or in attributes. Elements can be nested, which represents the parent and child relationships. XML elements are extensible. Table 2 gives examples of some objects of the middle-tier data model presented in XML data format.

Table 2 Data Represented in XML Format

Object Name	XML Data Format
student	<s_student> <login>bcosby</login> <name>Bill Cosby</name> <major>Mathematics of Computation</major> <s_taking> <course_id>CS244A</course_id> <time>TTh 14:45-17:00</time> <s_transcript /> </s_taking> <s_taking> <course_id>CS255</course_id> <time>MW 14:15-16:30</time> <s_transcript> <grade>83</grade> </s_transcript> </s_taking> </s_student>
faculty	<s_faculty> <login>sspielberg</login> <name>Steven Spielberg</name> <paycheck_sin>333333333</paycheck_sin> <s_paycheck> <salary>5244</salary> <issuedate>1/1/2003 12:00:00 AM</issuedate> </s_paycheck> <s_paycheck> <salary>5244</salary> <issuedate>1/15/2003 12:00:00 AM</issuedate> </s_paycheck> <s_paycheck> <salary>5244</salary> <issuedate>2/1/2003 12:00:00 AM</issuedate> </s_paycheck> <s_course> <cid>CS140</cid> <title>Data Structures and Algorithms</title> </s_course > </s_faculty>
organization	<s_org id="CONC"> <name>Concordia University</name> </s_org> <s_org id="CS"> <name>Computer Science</name> <par_org_id>SOE</par_org_id> </s_org> <s_org id="EE"> <name>Electrical Engineering</name>

	<par_org_id>SOE</par_org_id> </s_org>

As can see from Table 2, *s_student* has child element *s_taking*, which further nested child element *s_transcript*. Element can be empty. *s_org* with attribute *id = "CONC"*, is an example of data represented by using element attributes. *Id* is to identify the uniqueness, which is an example of enforcing the data constraints.

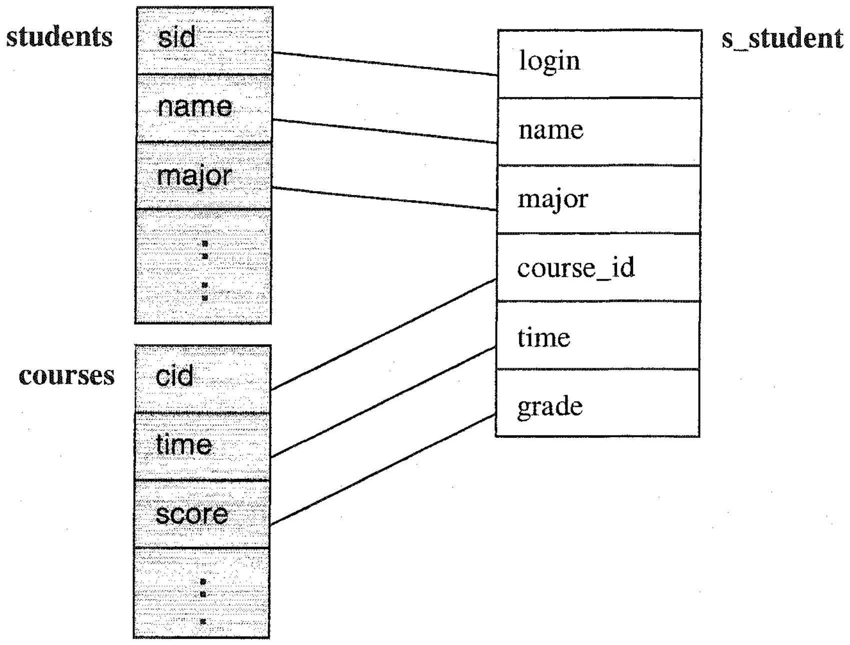


Figure 7 Mapping Between Source and Middle-tier Object Model

Figure 7 shows the data mapping from the relational tables (greyed out) from the source database School_A to the object s_student in the Middle-tier object model (on the right). The mapping schema will be discussed in section 3.3.

3.3 XML Schema to validate and translate the data model

XML standards use schema to describe the vocabularies and structure of the data model. XML Schema decides what elements and attributes available, in which order, the relationship and number of elements, in what data type, etc. There are two primary approaches for creating schemas: DTDs and XSDs. This project uses XSD approach.

Table 3 shows an example of modeling relational schema in XML schema.

Table 3 Represent Relational Table in XML Schema

Table Name	XML Schema
students(**sid**,name,major)	<xsd:element name=”students” concordia:table=”s_student”> <xsd:complexType> <xsd:sequence> <xsd:element name=”sid” concordia:column=”student_id”/> <xsd:element name=”name” type=”xsd:string”/> <xsd:element name=”major” type=”xsd:string” minOccurs=”0”/> </xsd:sequence> </xsd:complexType> </xsd:element>

The primary unit in XML Schema is *element*, it has name, type, value and attributes. Its datatype can be simple type (*simpleType)* and complex type (*complexType).* A simple type is a restriction on the value of an element or an attribute, while a complex type is a definition of a content model. The elements in the example above are of type *<xsd:string>*, which is a predefined simple type in XML Schema to match a set of characters. Both of simple type and complex type can be extended, and also allowed to create new data types. In Table 3, *element* name maps to the table name *students.* The *sequence* enforces the order of the child elements. Inside the *sequence*, child elements map to the columns of the table. Child element name maps to the column name. The attribute *type* defines the data type, as student *name* is of type *string*. The attributes *minOccurs* and *MaxOccurs* indicate the occurance of the data. Attribute has default values, for example, default value of *minOccurs* and *maxOccurs* is 1. If both *minOccurs* and *MaxOccurs* is 0, then the data should not appear in the XML data file. There are some predefined attributes, as well, the Schema syntax is flexible for users to define their own. Here, "*concordia*" is the namespace to uniquely identify a set of attributes associated to this specific application. *concordia:table* is defined to map to the tables in the target database, while *concordia:column* is to map to the columns of tables in the target database.

Table 4 and Table 5 give examples of representing relations in XML Schema.

Table 4 Represent Foreign Key Reference in XML Schema

Relations	XML Schema

The referenced column is the primary key
 	
professors(**pid**,sin,name)
courses(**cid**,title,instructor)
courses.instructor -->
professors.pid
	<xsd:element name="courses">
	  <xsd:annotation>
	    <xsd:appinfo>
	      <concordia:relation parent="professors" parentkey="pid" child="courses" childkey="cid"/>
	    </xsd:appinfo>
	  </xsd:annotation>
	...
	</xsd:element>

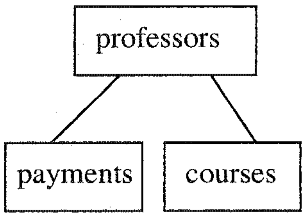
The referenced column is not the primary key
 	
professors(**pid**,sin,name)
paychecks(**sin,issuedate**,salary)
paychecks.sin -> professors.sin
	<xsd:element name="paychecks">
	  <xsd:annotation>
	    <xsd:appinfo>
	      <concordia:relation parent="professors" parentkey ="sin" child="paychecks" childkey ="sin"/>
	    </xsd:appinfo>
	  </xsd:annotation>
	...
	</xsd:element>

In the first example of Table 4, the *instructor* is a foreign key in the table *courses*, it references to the primary key *pid* in the table *professors*. The key referential constraint is included in the *<xsd:appinfo>* of *<xsd:annotation>* section. It is represented as *<concordia:relation>*. The *parent* and *child* attributes give the name of parent and child table, and *parentkey* and *childkey* attributes indicate the referenced columns. The second example of Table 4 shows the situation when the referenced column *sin* is not the primary key in the parent table *professors*. The schema mapping in both cases is the same.

Table 4 shows the foreign key reference in the element represented for child table, the parent to reference child tables is presented in Table 5 below.

Table 5 Represent reference to tables in XML Schema

Reference Tables	XML Schema

	`<xsd:element name="professors">`
	`<xsd:complexType>`
	`<xsd:sequence>`
	`<xsd:element name="pid" type="xsd:string"/>`
	`<xsd:element name="name" type="xsd:string"/>`
	`<xsd:element name="paycheck_sin" type="xsd:string" minOccurs="0"/>`
	`<xsd:element` **ref="payments"** `minOccurs="0" maxOccurs="unbounded"/>`
	`<xsd:element` **ref="courses"** `minOccurs="0" maxOccurs="unbounded"/>`
	`</xsd:sequence>`
	`</xsd:complexType>`
	`</xsd:element>`

In the Table 5 example, *professors* element has a *ref* attribute to indicate the names of the reference tables, *payments* and *courses*. The attributes *minOccurs* and *maxOccurs* decide that the number of payments and courses can be none or unlimited.

For extracting data, the tool also provides two special attributes, *directive* and *join*, under this application namespace *concordia*. Table 6 illustrates an example of use *directive* in XML Schema.

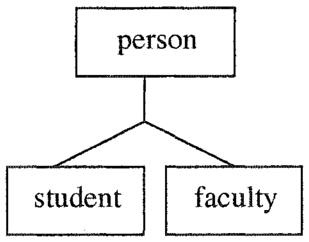
Table 6 Use of Directive in XML Schema

Table Name	**XML Schema**
address(**uid**, street, city, province, postcode) students(**sid**, name, *fk_address*) students.fk_address →address.uid professors(**pid**, name, *fk_address*) professors.fk_address →address.uid departments(**did**,name,*fk_address*) departments.fk_address →address.uid	<xsd:element name="s_address" concordia:table="address"> <xsd:annotation> <xsd:appinfo> **<concordia:directive sql="SELECT street,city,province,postcode,sid,NULL FROM address,students WHERE address.uid=students.fk_address;"/>** **<concordia:directive sql="SELECT street,city,province,postcode,pid,NULL FROM address,professors WHERE address.uid=professors.fk_address;"/>** **<concordia:directive sql="SELECT street,city,province,postcode,NULL,name FROM address,departments WHERE address.uid=departments.fk_address;"/>** </xsd:appinfo> </xsd:annotation> ... </xsd:element>

In the schema, there is also a mapping of inheritance relationship. Both *Student* and *faculty* are extended from *person* object. Table 7 gives the modeling of object extension in XML Schema.

Table 7 Represent Inheritance in XML Schema

Inheritance Relation	XML Schema

	`<xsd:complexType name="type_person">`
	`        <xsd:sequence>`
	`            <xsd:element name="login" type="xsd:string"/>`
	`            <xsd:element name="name" type="xsd:string"/>`
	`            <xsd:element name="paycheck_sin" type="xsd:string"`
	`minOccurs="0"/>`
	`            <xsd:element ref="s_paycheck" minOccurs="0"`
	`maxOccurs="unbounded"/>`
	`        </xsd:sequence>`
	`</xsd:complexType>`
	`<xsd:element name="person" type="type_person"/>`
	`<xsd:element name="student" type="type_student"/>`
	`<xsd:complexType name="type_student">`
	`    <xsd:complexContent>`
	`        <xsd:extension base="type_person">`
	`            <xsd:sequence>`
	`                <xsd:element name="major" type="xsd:string"`
	`minOccurs="0"/>`
	`                <xsd:element ref="s_taking" minOccurs="0"`
	`maxOccurs="unbounded"/>`
	`            </xsd:sequence>`
	`        </xsd:extension>`
	`    </xsd:complexContent>`
	`</xsd:complexType>`
	`<xsd:element name="faculty" type="type_faculty"/>`
	`<xsd:complexType name="type_faculty">`
	`    <xsd:complexContent>`
	`        <xsd:extension base="type_person">`
	`            <xsd:sequence>`
	`                <xsd:element ref="s_course" minOccurs="0"`
	`maxOccurs="unbounded"/>`
	`            </xsd:sequence>`
	`        </xsd:extension>`
	`    </xsd:complexContent>`
	`</xsd:complexType>`

Illustrated in Table 7, it defines a base type named *type_person*, both *student* and *faculty* element type, *type_student* and *type_faculty*, are extended from the base type *type_person*, using *<xsd:extension base="type_person">* in complex content model.

Student, faculty and department all have addresses. Both *student* and *faculty* are derived from *person* type, and *person* has *person_id* as its unique identification. The *address* table is referenced by *person* and *department* tables through their key elements, *person_id* and *org_id*, respectively. In the rows of *student* or *faculty* information, the columns reference to *department* is null, likewise, in the records of *department* address, the column reference to *person* is null. To represent the optional situation, *<xsd:choice>* is used as illustrated in Table 8 below.

Table 8 Represent Optional Data in XML Schema

Table Name	XML Schema

address(**uid**, street, city, province, postcode, person_id, org_id)	<xsd:element name="address">
	  <xsd:complexType>
	    <xsd:sequence>
	      <xsd:element name="street" type="xsd:string"/>
	      <xsd:element name="city" type="xsd:string"/>
	      <xsd:element name="province" type="xsd:string"/>
	      <xsd:element name="postcode" type="xsd:string"/>
	      <xsd:choice>
	        <xsd:element name="person_id"
	type="xsd:string"/>
	        <xsd:element name="org_id" type="xsd:string"/>
	      </xsd:choice>
	    </xsd:sequence>
	  </xsd:complexType>
	</xsd:element>

This project not only uses the predefined attributes in XML Schema, but also has defined a set of attributes in the namespace of this project, named of "concordia". Table 9 lists a summary of the attributes specially defined for this project.

Table 9 Application-defined attributes in XML Schema

Application Defined Attribute	**Usage**
concordia:table	To map the table name in the source database schema
concordia:column	To map the column name in the source database schema
concordia:relation	To describe the table relations in the target database schema
concordia:directive	To give directive sql command for extracting data from source database
concordia:join	To join tables when extracting data from source database
concordia:xmlOnly	To indicate the element or attibute is only for schema mapping purpose, not involving in making SQL commands
concordia:abstract	To indicate the relationship is in the source database schema

This gives the design of the middle-tier data model, and the mapping rules between relational database schemas to XML schemas. To understand and interpret the rules and to actually accomplish the task of transferring data, is implemented by programs written in C# programming language, the organization and specification of the program is given in the next chapter.

Chapter 4

4. Program Specification

4.1 Program Design Overview

This program is developed by using C# language in the Microsoft Visual Studio .NET environment. The implementation consists of three modules, Schema, ExportToXml, and XmlToDatabase. The main functions of three modules are:

- Schema is a module to read and parse the schema file, and build a category for each database objects, including tables, columns, data types, relationships between tables, join and directive information etc., so as to be used in the other two modules to extract or import data from/to the database. It is actually a class library.
- ExportToXml module is to connect to the source database, and to extract the data by using the category information built by Schema module, then to transfer to the intermediate data file, which is a XML document.
- XmlToDatabase module is to connect to the target database, to validate and import data from the intermediate data file, in XML data format, to the target database according to the specified schema mapping.

4.2 Specification of Schema Module

This module reads XML Schema file, assembles elements, attributes, datatypes and other application related information to database objects, such as tables, columns, relations, and special mapping objects designed for this application, for example, joins, directives.

Class TableInfo

Represents database table related information.

Data Member

XmlSchemaElement m_xmlSchemaElement

to store each element from the XML Schema file

Hashtable m_tblColumns

a hash table object to store columns of the table, it uses name of column as index key, and ColumnInfo object as its value

ArrayList m_lstColumns

to store a list of ColumnInfo objects

ArrayList m_lstInnerTables

to store a list of tables it relates with

ArrayList m_lstHeirs

to store a list of heir tables

TableInfo m_superTableInfo

to hold the TableInfo object of its super table

RelationInfo m_relationInfo

to hold the RelationInfo object which represents the referential relationship with other table

JoinInfo m_joinInfo

to hold the JoinInfo object

DirectiveInfo m_directiveInfo

to hold the DirectiveInfo object

string m_strTableName

to store the name of the table

static bool m_bSucceeded

flag to set the result of the validation is successful or not

Method

TableInfo()

TableInfo(XmlSchemaElement schemaElement, XmlSchemaObjectTable typeTable, Hashtable tblCatalog)

constructor, initializes data members

Parse(XmlSchemaElement element, XmlSchemaObjectTable typeTable, Hashtable tblCatalog)

parses the XML Schema *element*, and stores the result in the hash table *tblcatalog*

static Hashtable Compile(FileStream fs, ref XmlSchema schema)

reads file *fs* to *schema*, validates *schema*, and parses through each element of the *schema*

ColumnInfo Column(object key)

retrieves the *columnInfo* object with the given *key* index from the hash table *m_tblColumns*

static CompileAppInfo(Hashtable tblCatalog, TableInfo ti)

compiles the info in the *<xsd:appinfo>* section of the schema file

ValidationCallback(object sender, ValidationEventArgs args)

invokes when error occurs during the XML schema validation

Property

string ElementName

to get/set the name of the element, *m_xmlSchemaElement.Name*

string TableName

to get/set the name of table, *m_strTableName*

ArrayList LocalColumns

returns the list of columns of the table, *m_lstColumns*

ColumnInfo LocalColumn(int i)

returns the ColumnInfo object with index *i* from the column list, *m_lstColumns*

ColumnInfo IDREF

returns the ColumnInfo object which is of type *xsd:IDREF*

ArrayList InnerList

returns the ColumnInfo object with index *i* from the column list, *m_lstColumns*

TableInfo Outter

returns *parent* TableInfo object

RelationInfo RelationInfo

returns RelationInfo object *m_relationInfo*

JoinInfo JoinInfo

returns JoinInfo object *m_joinInfo*

DirectiveInfo DirectiveInfo

returns DirectiveInfo object *m_directiveInfo*

TableInfo Super

returns TableInfo object *m_superTableInfo*

ArrayList HeirList

returns the heirs of the table, *m_lstHeirs*

Class ColumnInfo

Represents table column related information.

Data Member

TableInfo m_tableInfo

to store table information

bool m_bForeignKey

flag to indicate if the column is foreign key

SqlDbType m_dbType

to store SQL Server data type

string m_strColumnName

to store name of the column

bool m_bXmlOnly

flag to indicate if the attribute is only for XML Schema mapping usage

Method

ColumnInfo(TableInfo ti)

constructor, initializes data members

SqlDbType ConvertXmlType(string strXmlType)

convert XML data type to SQL Server type

Property

abstract string ElementName

abstract property to be overridden in the derived class for the name of the XML element

abstract decimal MinOccurs

abstract property to be overridden in the derived class for the attribute of minimal occurence abstract decimal MaxOccurs

abstract property to be overridden in the derived class for the attribute of maximum occurence

abstract string XmlType

abstract property to be overridden in the derived class for the name of the XML data type

virtual bool IsElement

to indicate whether is from the schema element or attribute

virtual bool IsAttribute

to indicate whether is from the schema element or attribute

virtual bool IsAbstract

returns whether is abstract element for the schema mapping purpose

TableInfo TableInfo

returns the table object of this column belong to, *m_tableInfo*

string ColumnName

to get/set name of the column, *m_strColumnName*

bool IsXmlOnly

returns flag of XML Schema use only, *m_bXmlOnly*

SqlDbType DbType

returns SQL Server data type, *m_dbType*

bool IsFK

to get/set whether the column is a foreign key, *m_bForeignKey*

Class ElementCI

Derived from class ColumnInfo, stores column information extract from the Schema element, <*xsd:element*>

Data Member

XmlSchemaElement m_xmlSchemaElement

stores corresponding XML schema element

Method

ElementCI(XmlSchemaElement element, TableInfo ti)

constructor, initializes data members

Property

override string ElementName

returns element name, *m_xmlSchemaElement.name*

override decimal MinOccurs

returns attribute of minimal occurence, *m_xmlSchemaElement.minOccurs*

override decimal MaxOccurs

returns attribute of maximum occurence, *m_xmlSchemaElement.maxOccurs*

override bool IsElement

overrides to indicate whether is from the schema element or attribute

override string XmlType

returns XML data type, *m_xmlSchemaElement.SchemaTypeName.Name*

Class AttributeCI

Derived from class ColumnInfo, store column information extract from the attribute element, *<xsd:attribute>*

Data Member

XmlSchemaAttribute m_xmlSchemaAttribute

stores corresponding XML schema attribute

Method

AttributeCI(XmlSchemaAttribute attribute, TableInfo ti) constructor, initializes data members

Property

override string ElementName

returns attribute name, *m_xmlSchemaAttribute.Name*

override decimal MinOccurs

it is an attribute, always returns 1

override decimal MaxOccurs

it is an attribute, always returns 1

override bool IsAttribute

overrides to indicate whether is from the schema element or attribute

override string XmlType

returns XML data type, *m_xmlSchemaAttribute.SchemaTypeName.Name*

Class DirectiveInfo

Represents the directive information of the schema element from the XML Schema file, *<concordia:directive>*

Data Member

ArrayList m_lstDirectives

holds a list of directive objects

Method

DirectiveInfo()

constructor, initializes data members

int Add(Directive d)

adds directive object *d* to the list *m_lstDirectives*

Property

int Count

returns the number of directive objects in the list, *m_lstDirectives*

Directive this[int index]

returns the directive object with the given *index* from the list, *m_lstDirectives*

Class Directive

Represents directive object, which provides SQL command directly

Data Member

string m_strSql

holds sql command string

Method

Directive(XmlNode node)

constructor, initializes data members

Property

string Sql

to get/set string *m_strSql*

Class JoinInfo

Represents the join information of the schema element from the XML Schema file, *<concordia:join>*

Data Member

ArrayList m_lstJoins

holds a list of join objects

Method

JoinInfo()

constructor, initializes data members

int Add(Join j)

adds join relation object *j* to the list *m_lstJoins*

Property

int Count

returns the number of join objects in the list, *m_lstJoins*

Join this[int index]

returns the join object with the given *index* from the list, *m_lstJoins*

Class Join

Represents the join relation object of the list, *m_lstJoins*, in the JoinInfo class

Data Member

string parent

gives the name of the parent table

string[] parentKeys

gives the array of keys of the parent table

string child

gives the name of the child table

string[] childKeys

gives the array of keys of the child table

Method

Join(XmlNode node)

constructor, initializes data members

Property

string Parent

to get/set the parent table name

string[] ParentKeys

to get/set the keys of the parent table

string Child

to get/set the child table name

string[] ChildKeys

to get/set the keys of the child table

Class RelationInfo

Represents the relationship information of the schema element from the XML Schema file, <*concordia:relation*>

Data Member

ArrayList m_lstRelations

holds a list of relations for the table

Method

RelationInfo()

constructor, initializes data members

int Add(Relation r)

adds relation *r* to the list *m_lstRelations*

Property

int Count

returns the number of relations in the list, *m_lstRelations*

Relation this[int index]

returns the relation with the given *index* from the list, *m_lstRelations*

Relation AbstractRelation

returns the abstract relation from the list, *m_lstRelations*

Class Relation

Represents the relation object of the list, *m_lstRelations*, in the class RelationInfo

Data Member

TableInfo m_parentTableInfo

stores the parent table information of the relation

ColumnInfo[] m_parentKeys

stores parent keys information of the relation

TableInfo m_childTableInfo

stores the child table information of the relation

ColumnInfo[] m_childKeys

stores child keys information of the relation

bool m_bIsAbstract

flag of whether the relation is abstract or not

Method

Relation(Hashtable catalog, XmlNode node)

constructor, initializes data members

XmlAttribute GetAttribute(XmlAttributeCollection attributes, string strName)

to get the attribute matching with the *strName*

Property

TableInfo Parent

to get/set the parent table information

ColumnInfo[] ParentKeys

to get/set the keys of the parent table

TableInfo Child

to get/set the child table information

ColumnInfo[] ChildKeys

to get/set the keys of the child table

bool IsAbstract

returns whether is abstract relation, *m_bIsAbstract*

4.3 Specification of ExportToXml Module

Class ExportToXml

Connects to the source database, and export data to the XML data file according to the corresponding XML schema

Data Member

const string strDOCROOT

constant string to hold the root name of XML document

SqlConnection m_conn

holds the connection object to SQL Server database

XmlTextWriter m_xmlTextWriter

holds the XML text writer object which is used to write the XML data file

Hashtable m_tblCatalog

holds the hash table object which stores all the tableInfo objects

Hashtable m_tblAdapters

holds the hash table object with all the adapter objects, which store the data from the database

Method

ExportToXml(string strSource, string strDB)

constructor, initializes data members, and generates the database connection string

Execute(string schemaFile, string xmlFile)

controls the actions of selecting the data from the database and writing the data to a XML file, including compiling the XML schema, building a catalog of table information, connecting to the database, selecting the data from the tables in the catalog object, and disconnecting from the database when the actions finish

ExecuteOneTable(TableInfo ti, TableInfo directParent, Object[] passValues)

selects the data from the table of tableInfo object *ti* holding for

string FormatSqlSelect(TableInfo ti, TableInfo directParent)

builds SQL command to select from the table of tableInfo object *ti* holding for

string FormatSqlColumn(string tn, string cn)

builds SQL command with table name *tn* and column name *cn*

4.4 Specification of XmlToDatabase Module

Class ExportToXml

Connects to the target database, and imports data from the XML data file to the destined database according to the corresponding XML schema

Data Member

const string strDOCROOT

constant string to hold the root name of XML document

SqlConnection m_conn

holds the connection object to SQL Server database

XmlDocument m_xmlDoc

holds the XML document object

Hashtable m_tblCatalog

holds the hash table object which stores all the tableInfo objects

Method

XmlToDatabase(string strSource, string strDB)

constructor, initializes data members, and generates the database connection string

Execute(string schemafile, string xmlfile)

controls the actions to insert data into the database, including connects to the database, calls to functions like, readXml, buildDependency, executeOneTable and disconnects to the database when actions finish

ReadXml(XmlSchema schema, string xmlfile)

validates the XML data file *xmlfile* against XML Schema *schema*, if succeeds, loads into *m_xmlDoc* object

ValidationCallback(object sender, ValidationEventArgs args)

callback function to give error messages when XML validation fails

BuildDependency(TableInfo ti, Hashtable dependents)

checks relationInfo list and reference table list of *ti* to build a hash table of *dependents*

ExecuteOneTable(TableInfo ti, XmlNode outter, object passvalue)

inserts the data into the table of tableInfo object *ti* holding for

Guid ExecuteOneRow(SqlCommand command, TableInfo ti, XmlNode row, object passvalue)

inserts the data into the rows of the table

SqlCommand MakeInsertCommand(TableInfo ti)

builds the SQL command for inserting data into the database

SqlCommand MakeSelectCommand(Relation r)

builds the SQL command for selecting data from the database

Chapter 5

5. Conclusions and Recommendations

5.1 Conclusions

This project has presented a tool for transferring data between heterogeneous data sources by using XML standards and the related technology. The tool has introduced a middle-tier data model, as a bridge between the source and target databases. The middle-tier data model is an object-oriented, hierachical structured representation of the semantical meaning of the source and target data models. The middle-tier data model is in XML data format, and the mapping between source to middle-tier and middle-tier to target is via two XML schemas. The mapping scenarios and rules are presented, and the translation to the XML schemas and data mapping is provided in this report. The project also implements the program to coorperate these three data models to successfully transfer data from the source to the target database.

5.2 Recommendations for Future Works

This project experients a way to use XML standards as the middle-tier data model to solving the data transferring problem between heterogeous data sources. There are further works to do to extend and fulfill the tool, such as, formalizing the data and schema mapping rules; automating the process, for example, automatically generating the mapping schemas; providing tools to configure the transferring and integration related parameters, and to manipulate middle-tier data model to meet the different schemas of target data sources.

Bibliography

1) **W3C**, Extensible Markup Language (XML) http://www.w3.org/XML

2) **W3C,** XML Schema http://www.w3.org/XML/Schema

3) **Michael Morrison**, "Teach Yourself XML in 24 Hours"**,** SAMS Publishing, 2001

4) **Harold, Elliotte Rusty**, "XML in a nutshell", O'Reilly & Associates, Inc., 2002

5) **Eric van der Vlist**, "XML Schema", O'Reilly & Associates, Inc., 2002

6) **Thuan Thai & Hoang Q. Lam**, ".NET Framework Essentials", O'Reilly & Associates, Inc., 2001

7) **MSDN Library Help**, ".NET development", Microsoft Co., 2003

8) **MSDN Library Help**, "Visual C# Language", Microsoft Co., 2003

9) **Karli Watson**, "Beginning C#", Wrox Press Ltd., 2001

10) **John Hunt**, "Guide to C# and Object Orientation", Springer-Verlag London Ltd., 2002

11) **James W. Cooper**, "C# Design Patterns", Addison-Wesley, 2003

12) **Microsoft SQL Server Technical Resources**, http://www.microsoft.com/sql/techinfo/default.asp

13) **Robert Stevens**, A Definition of Semantic Heterogeneity, http://www.cs.man.ac.uk/~stevensr/rash/heterogeneity.html

14) **Joachim Hammer**, The information integration wizard (Iwiz) project, http://www.cise.ufl.edu/~jhammer/publications.html

Appendices

A.Database schemas

School_A:

```
CREATE TABLE address
(
   uid        int     NOT NULL,
   street         varchar(100),
   city           varchar(50),
   province       char(2),
   postcode       varchar(6),
   CONSTRAINT pk_address PRIMARY KEY (uid)
)

CREATE TABLE departments
(
   did            char(5)       NOT NULL,
   name           varchar(50) NOT NULL,
   fk_address int          FOREIGN KEY REFERENCES address(uid),
   fk_dept        char(5)   FOREIGN KEY REFERENCES departments(did),
   CONSTRAINT pk_departments PRIMARY KEY(did)
)

CREATE TABLE students
(
   sid            char(20)      NOT NULL,
   name           varchar(50) NOT NULL,
   major          varchar(50),
   fk_address int          FOREIGN KEY REFERENCES address(uid),
   fk_dept        char(5)   FOREIGN KEY REFERENCES departments(did),
   CONSTRAINT pk_students PRIMARY KEY (sid)
)

CREATE TABLE professors
(
   pid            char(20)          NOT NULL,
   name           varchar(50)       NOT NULL,
   sin            char(9)           NOT NULL UNIQUE,
   fk_address int          FOREIGN KEY REFERENCES address(uid),
   fk_dept        char(5)   FOREIGN KEY REFERENCES departments(did),
   CONSTRAINT pk_professors PRIMARY KEY (pid)
)

CREATE TABLE paychecks
(
   sin        char(9)      NOT NULL FOREIGN KEY REFERENCES professors(sin),
   issuedate  datetime NOT NULL,
   salary     decimal  NOT NULL,
   CONSTRAINT pk_paychecks PRIMARY KEY (sin,issuedate)
)

CREATE TABLE courses
(
   cid        char(8)      NOT NULL,
   title      varchar(50) NOT NULL,
   time       char(15) NOT NULL,
   instructor char(20)     FOREIGN KEY REFERENCES professors(pid),
   CONSTRAINT pk_courses PRIMARY KEY (cid)
)

CREATE TABLE taking
(
      sid     char(20) NOT NULL FOREIGN KEY REFERENCES students(sid),
      cid     char(8)  NOT NULL FOREIGN KEY REFERENCES courses(cid),
   score float,
   CONSTRAINT pk_taking PRIMARY KEY (sid, cid)
)
```

School_B:

```
CREATE TABLE s_person
(
   row_id        uniqueidentifier   NOT NULL,
   person_type   varchar(30)     NOT NULL,
   login         char(20)     NOT NULL,
   name          varchar(50)     NOT NULL,
   CONSTRAINT s_person_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_person_pay
(
      row_id         uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                     s_person(row_id),
   paycheck_sin     char(9)      NOT NULL
)

CREATE TABLE s_student
(
      row_id         uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                     s_person(row_id),
   major    varchar(50),
   CONSTRAINT s_student_pk PRIMARY KEY (row_id)
)
```

```
CREATE TABLE s_faculty
(
        row_id          uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                        s_person(row_id),
   CONSTRAINT s_faculty_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_org
(
   row_id        uniqueidentifier   NOT NULL,
   name          varchar(50)        NOT NULL,
   par_org_id    uniqueidentifier,
   CONSTRAINT s_org_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_member_of
(
        row_id          uniqueidentifierNOT NULL,
        person_id       uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                        s_person(row_id),
        org_id          uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                        s_org(row_id),
        org_name        varchar(50)      NOT NULL
)

CREATE TABLE s_address
(
   row_id      uniqueidentifier   NOT NULL,
   street      varchar(100),
   city        varchar(50),
   province    char(2),
   zip         varchar(20),
        person_id       uniqueidentifier FOREIGN KEY REFERENCES
                        s_person(row_id),
   org_id      uniqueidentifier FOREIGN KEY REFERENCES s_org(row_id),
   CONSTRAINT s_address_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_paycheck
(
   row_id      uniqueidentifier   NOT NULL,
        person_id       uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                        s_person(row_id),
   issuedate   datetime     NOT NULL,
   salary      int          NOT NULL,
   CONSTRAINT s_paycheck_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_course
(
   row_id  uniqueidentifier       NOT NULL,
   cid     char(8)                NOT NULL,
   title   varchar(50)            NOT NULL,
        faculty_id     uniqueidentifier FOREIGN KEY REFERENCES
                       s_faculty(row_id),
   CONSTRAINT s_course_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_taking
(
   row_id       uniqueidentifier   NOT NULL,
        student_id     uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                       s_student(row_id),
        course_id      uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                       s_course(row_id),
        time           char(15)       NOT NULL,
   CONSTRAINT s_taking_pk PRIMARY KEY (row_id)
)

CREATE TABLE s_transcript
(
   row_id       uniqueidentifier   NOT NULL,
        taking_id      uniqueidentifier NOT NULL FOREIGN KEY REFERENCES
                       s_taking(row_id),
   grade     float,
   CONSTRAINT s_transcript_pk PRIMARY KEY (row_id)
)
```

B. XML Schemas

From School_A database to Middle-tier data model

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:concordia="http://concordia/qlin">
   <xsd:element name="school_database">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="s_student" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="s_faculty" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="s_address" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="s_org" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="s_member_of" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_transcript" concordia:table="taking">
      <xsd:annotation>
```

```
        <xsd:appinfo>
          <concordia:relation parent="s_taking" parentKey="student_id
course_id" child="s_transcript" childKey="taking_id taking_id_1"
abstract="true"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="taking_id" type="xsd:string"
concordia:column="sid" minOccurs="0" maxOccurs="0"/>
          <xsd:element name="taking_id_1" type="xsd:string"
concordia:xmlOnly="true" concordia:column="cid" minOccurs="0"
maxOccurs="0"/>
          <xsd:element name="grade" type="xsd:float" minOccurs="0"
concordia:column="score"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_taking" concordia:table="taking">
      <xsd:annotation>
        <xsd:appinfo>
          <concordia:relation parent="s_course" parentKey="cid"
child="s_taking" childKey="course_id"/>
          <concordia:relation parent="s_student" parentKey="login"
child="s_taking" childKey="student_id" abstract="true"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="student_id" type="xsd:string"
minOccurs="0" maxOccurs="0" concordia:column="sid"/>
          <xsd:element name="course_id" type="xsd:string"
concordia:column="cid"/>
          <xsd:element name="time" type="xsd:string"
concordia:column="courses.time"/>
          <xsd:element ref="s_transcript" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_course" concordia:table="courses">
      <xsd:annotation>
        <xsd:appinfo>
          <concordia:relation parent="s_faculty" parentKey="login"
child="s_course" childKey="faculty_id" abstract="true"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="cid" type="xsd:string"/>
          <xsd:element name="title" type="xsd:string"/>
          <xsd:element name="faculty_id" type="xsd:string"
concordia:column="instructor" maxOccurs="0" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_person" type="t_s_person"/>
    <xsd:complexType name="t_s_person">
      <xsd:sequence>
        <xsd:element name="login" type="xsd:string"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="paycheck_sin" type="xsd:string"
minOccurs="0"/>
        <xsd:element ref="s_paycheck" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="s_student" type="t_s_student"
concordia:table="students"/>
   <xsd:complexType name="t_s_student">
      <xsd:complexContent>
         <xsd:extension base="t_s_person">
            <xsd:sequence>
               <xsd:element name="login" type="xsd:string"
concordia:column="sid"/>
               <xsd:element name="name" type="xsd:string"/>
               <xsd:element name="paycheck_sin" type="xsd:string"
minOccurs="0" concordia:column="NULL"/>
               <xsd:element ref="s_paycheck" minOccurs="0"
maxOccurs="unbounded"/>
               <xsd:element name="major" type="xsd:string"
minOccurs="0"/>
               <xsd:element ref="s_taking" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="s_faculty" type="t_s_faculty"
concordia:table="professors">
   </xsd:element>
   <xsd:complexType name="t_s_faculty">
      <xsd:complexContent>
         <xsd:extension base="t_s_person">
            <xsd:sequence>
               <xsd:element name="login" type="xsd:string"
concordia:column="pid"/>
               <xsd:element name="name" type="xsd:string"/>
               <xsd:element name="paycheck_sin" type="xsd:string"
minOccurs="0" concordia:column="sin"/>
               <xsd:element ref="s_paycheck" minOccurs="0"
maxOccurs="unbounded"/>
               <xsd:element ref="s_course" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="s_paycheck" concordia:table="paychecks">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_person"
parentKey="paycheck_sin" child="s_paycheck" childKey="person_id"
abstract="true"/>
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="salary" type="xsd:integer"/>
            <xsd:element name="issuedate" type="xsd:dateTime"/>
            <xsd:element name="person_id" type="xsd:string"
concordia:column="sin" minOccurs="0" maxOccurs="0"/>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_address" concordia:table="address">
      <xsd:annotation>
        <xsd:appinfo>
          <concordia:relation parent="s_person" parentKey="login"
child="s_address" childKey="person_id"/>
          <concordia:relation parent="s_org" parentKey="name"
child="s_address" childKey="org_id"/>
          <concordia:directive sql="SELECT
street,city,province,postcode,sid,NULL FROM address,students WHERE
address.uid=students.fk_address;"/>
          <concordia:directive sql="SELECT
street,city,province,postcode,pid,NULL FROM address,professors WHERE
address.uid=professors.fk_address;"/>
          <concordia:directive sql="SELECT
street,city,province,postcode,NULL,name FROM address,departments
WHERE address.uid=departments.fk_address;"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="street" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="province" type="xsd:string"/>
          <xsd:element name="zip" type="xsd:string"/>
          <xsd:choice>
            <xsd:element name="person_id" type="xsd:string"/>
            <xsd:element name="org_id" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_org" concordia:table="departments">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="par_org_id" type="xsd:IDREF"
concordia:column="fk_dept"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID"
concordia:xmlOnly="true" concordia:column="did"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="s_member_of">
      <xsd:annotation>
        <xsd:appinfo>
          <concordia:relation parent="s_org" parentKey="name"
child="s_member_of" childKey="org_id"/>
          <concordia:relation parent="s_person" parentKey="login"
child="s_member_of" childKey="person_id"/>
          <concordia:directive sql="SELECT
pid,departments.name,departments.name FROM departments,professors
WHERE departments.did=professors.fk_dept;"/>
          <concordia:directive sql="SELECT
sid,departments.name,departments.name FROM departments,students WHERE
departments.did=students.fk_dept;"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
```

```
        <xsd:sequence>
          <xsd:element name="person_id" type="xsd:string"/>
          <xsd:element name="org_id" type="xsd:string"/>
          <xsd:element name="org_name" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

From middle-tier data model to School_B database

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:concordia="http://concordia/qlin">
   <xsd:element name="school_database">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="s_student" minOccurs="0"
maxOccurs="unbounded" />
            <xsd:element ref="s_faculty" minOccurs="0"
maxOccurs="unbounded" />
            <xsd:element ref="s_address" minOccurs="0"
maxOccurs="unbounded" />
            <xsd:element ref="s_org" minOccurs="0"
maxOccurs="unbounded" />
            <xsd:element ref="s_member_of" minOccurs="0"
maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_transcript">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_taking"
parentKey="student_id course_id" child="s_transcript"
childKey="taking_id taking_id_1" abstract="true" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="taking_id" type="xsd:string"
minOccurs="0" maxOccurs="0" />
            <xsd:element name="taking_id_1" type="xsd:string"
minOccurs="0" maxOccurs="0" concordia:xmlOnly="true" />
            <xsd:element name="grade" type="xsd:float" minOccurs="0"
/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_taking">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_course" parentKey="cid"
child="s_taking" childKey="course_id" />
            <concordia:relation parent="s_student" parentKey="login"
child="s_taking" childKey="student_id" abstract="true" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
```

```
         <xsd:sequence>
            <xsd:element name="student_id" type="xsd:string"
minOccurs="0" maxOccurs="0" />
            <xsd:element name="course_id" type="xsd:string" />
            <xsd:element name="time" type="xsd:string" />
            <xsd:element ref="s_transcript" minOccurs="0" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_course">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_faculty" parentKey="login"
child="s_course" childKey="faculty_id" abstract="true" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="cid" type="xsd:string" />
            <xsd:element name="title" type="xsd:string" />
            <xsd:element name="faculty_id" type="xsd:string"
minOccurs="0" maxOccurs="0" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_person" type="t_s_person"></xsd:element>
   <xsd:complexType name="t_s_person">
      <xsd:sequence>
         <xsd:element name="login" type="xsd:string" />
         <xsd:element name="name" type="xsd:string" />
         <xsd:element name="paycheck_sin" type="xsd:string"
minOccurs="0" concordia:xmlOnly="true"/>
         <xsd:element ref="s_paycheck" minOccurs="0"
maxOccurs="unbounded" />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="s_student" type="t_s_student"></xsd:element>
   <xsd:complexType name="t_s_student">
      <xsd:complexContent>
         <xsd:extension base="t_s_person">
            <xsd:sequence>
               <xsd:element name="major" type="xsd:string"
minOccurs="0" />
               <xsd:element ref="s_taking" minOccurs="0"
maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="s_faculty" type="t_s_faculty"></xsd:element>
   <xsd:complexType name="t_s_faculty">
      <xsd:complexContent>
         <xsd:extension base="t_s_person">
            <xsd:sequence>
               <xsd:element ref="s_course" minOccurs="0"
maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="s_paycheck">
```

```
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_person" parentKey="login"
child="s_paycheck" childKey="person_id" abstract="true" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="salary" type="xsd:integer" />
            <xsd:element name="issuedate" type="xsd:dateTime" />
            <xsd:element name="person_id" type="xsd:string"
minOccurs="0" maxOccurs="0" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_address">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_person" parentKey="login"
child="s_address" childKey="person_id" />
            <concordia:relation parent="s_org" parentKey="name"
child="s_address" childKey="org_id" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="street" type="xsd:string" />
            <xsd:element name="city" type="xsd:string" />
            <xsd:element name="province" type="xsd:string" />
            <xsd:element name="zip" type="xsd:string" />
            <xsd:choice>
               <xsd:element name="person_id" type="xsd:string" />
               <xsd:element name="org_id" type="xsd:string" />
            </xsd:choice>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_org">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="name" type="xsd:string" />
            <xsd:element name="par_org_id" type="xsd:IDREF"
minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="id" type="xsd:ID"
concordia:xmlOnly="true"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="s_member_of">
      <xsd:annotation>
         <xsd:appinfo>
            <concordia:relation parent="s_org" parentKey="name"
child="s_member_of" childKey="org_id" />
            <concordia:relation parent="s_person" parentKey="login"
child="s_member_of" childKey="person_id" />
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="person_id" type="xsd:string" />
            <xsd:element name="org_id" type="xsd:string" />
            <xsd:element name="org_name" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

C. Program Source Code

The program was coded by using Microsoft C# language, the project consists of three modules: Schema, ExportToXml, and XmlToDatabase modules.

Module Schema:

TableInfo.cs

```
using System;
using System.IO;
using System.Collections;
using System.Xml;
using System.Xml.Schema;

namespace DataTransferProject.Schema
{

    /// class for table related information

    public class TableInfo
    {
        private XmlSchemaElement m_xmlSchemaElement;

        private Hashtable     m_tblColumns;
        private ArrayList     m_lstColumns;
        private ArrayList     m_lstInnerTables = null;
        private ArrayList     m_lstHeirs = null;

        private TableInfo     m_superTableInfo = null;
        private RelationInfo  m_relationInfo = null;
        private JoinInfo      m_joinInfo = null;
        private DirectiveInfo m_directiveInfo = null;

        private string        m_strTableName = null;

        public TableInfo()
        {
            m_lstColumns = new ArrayList();
            m_tblColumns = new Hashtable();
        }
```

```
public TableInfo(XmlSchemaElement schemaElement, XmlSchemaObjectTable
typeTable, Hashtable tblCatalog)
{
    m_lstColumns = new ArrayList();
    m_tblColumns = new Hashtable();
    Parse(schemaElement, typeTable, tblCatalog);
    tblCatalog.Add(ElementName, this);
}

public void Parse(XmlSchemaElement element, XmlSchemaObjectTable typeTable,
Hashtable tblCatalog)
{
    m_xmlSchemaElement = element;

    if (element.UnhandledAttributes != null)
    {
        // Handle "Concordia" namespace attributes
        foreach (XmlAttribute attr in element.UnhandledAttributes)
        {
            if (attr.Name.Equals("concordia:table"))
            {
                // Extracts table name
                m_strTableName = attr.Value;
            }
            else
            {
                // Throw exception
                throw new NotSupportedException("Unrecognized attribute " +
attr.Name);
            }
        }
    }

    // Check schema type, can be anonymous type or named type
    XmlSchemaType schemaType;
    if ((schemaType = element.SchemaType) == null)
    {
        schemaType = (XmlSchemaType)typeTable[element.SchemaTypeName];
    }

    // xsd:complexType
    if (schemaType is XmlSchemaComplexType)
    {
        XmlSchemaComplexType complexType =
                                (XmlSchemaComplexType)schemaType;

        // attributes of element type
        for (int i = 0; i < complexType.Attributes.Count; i++)
        {
            if (complexType.Attributes[i] is XmlSchemaAttribute)
            {
                ColumnInfo ci = new
                    AttributeCI((XmlSchemaAttribute)complexType.Attributes[i],
                    this);
                m_lstColumns.Add(ci);
        m_tblColumns.Add(ci.ElementName, ci);
    }
    else
    {
        throw new NotSupportedException();
    }

}

// elements
XmlSchemaParticle schemaParticle; // Partical is mutually exclusive with the
                                            ContentModel
if ((schemaParticle = complexType.Particle) == null) // ContentModel
{
    // xsd:complexContent/xsd:extension
    if (complexType.ContentModel.Content is
                                XmlSchemaComplexContentExtension)
    {
        XmlSchemaComplexContentExtension ext =
    (XmlSchemaComplexContentExtension)complexType.ContentModel.Conten
    t;
        // assume the prefix of type name is t_
        string superName = ext.BaseTypeName.Name.Substring(2);
        if ((m_superTableInfo = (TableInfo)tblCatalog[superName]) == null)
        {
            m_superTableInfo = new TableInfo();
            tblCatalog.Add(superName, m_superTableInfo);
        }

        // wire super's heir list
        if (m_superTableInfo.HeirList == null)
        {
            m_superTableInfo.HeirList = new ArrayList();
        }
        m_superTableInfo.HeirList.Add(this);

        schemaParticle = ext.Particle;
    }
    else
    {
        throw new NotSupportedException();
    }
}

// xsd:sequence
if (schemaParticle is XmlSchemaSequence)
{
    XmlSchemaSequence schemaSequence =
                    (XmlSchemaSequence)schemaParticle;
    foreach (XmlSchemaObject schemaObject in schemaSequence.Items)
    {
        // xsd:element
        if (schemaObject is XmlSchemaElement)
        {
```

```
                    XmlSchemaElement schemaElement =
                                          (XmlSchemaElement)schemaObject;
                    if (schemaElement.RefName.IsEmpty)
                    {
                        // This element describes the column information
                        ColumnInfo ci = new ElementCI(schemaElement, this);
                        m_lstColumns.Add(ci);
                        m_tblColumns.Add(ci.ElementName, ci);
                    }
                    else
                    {
                        // This element represents the reference of other tables
                        if (m_lstInnerTables == null)
                        {
                            m_lstInnerTables = new ArrayList();
                        }
                        m_lstInnerTables.Add(schemaElement.RefName.Name);
                    }
                }
                else if (schemaObject is XmlSchemaChoice)
                {
                    // xsd:choice
                    foreach (XmlSchemaElement e in
                                          ((XmlSchemaChoice)schemaObject).Items)
                    {
                        ColumnInfo ci = new ElementCI(e, this);
                        m_lstColumns.Add(ci);
                        m_tblColumns.Add(ci.ElementName, ci);
                    }
                }
            }
        }
    }
    else
    {
        // xsd:simpleType
        throw new NotSupportedException();
    }
}

public static Hashtable Compile(FileStream fs)
{
    XmlSchema schema = null;
    return Compile(fs, ref schema);
}

public static Hashtable Compile(FileStream fs, ref XmlSchema schema)
{
    // read the schema
    schema = XmlSchema.Read(fs, new ValidationEventHandler(ValidationCallback));
    // compile to load schema elements
    schema.Compile(new ValidationEventHandler(ValidationCallback));

    // if pass the schema validation, go on to build the tables, columns, relations, etc.
    if (m_bSucceeded)
```

```
            {
                Hashtable tblCatalog = new Hashtable();

                // first pass to build table info from schema elements
                foreach (XmlSchemaElement element in schema.Elements.Values)
                {
                    if (tblCatalog.Contains(element.Name))
                    {
                        ((TableInfo)tblCatalog[element.Name]).Parse(element,
                                        schema.SchemaTypes, tblCatalog);
                    }
                    else
                    {
                        TableInfo ti = new TableInfo(element, schema.SchemaTypes, tblCatalog);
                    }
                }

                // second pass to rewire relations
                foreach (TableInfo ti in tblCatalog.Values)
                {
                    CompileAppInfo(tblCatalog, ti);
                }

                return tblCatalog;
            }
            else
            {
                return null;
            }

        }

        private static void CompileAppInfo(Hashtable tblCatalog, TableInfo ti)
        {
            if (ti.m_xmlSchemaElement.Annotation != null)
            {
                foreach (XmlSchemaObject schemaObject in
                                ti.m_xmlSchemaElement.Annotation.Items)
                {
                    if (schemaObject is XmlSchemaAppInfo)
                    {
                        foreach (XmlNode node in
    ((XmlSchemaAppInfo)schemaObject).Markup)
                        {
                            if (node.Name.Equals("concordia:relation"))
                            {
                                if (ti.m_relationInfo == null)
                                    ti.m_relationInfo = new RelationInfo();

                                ti.m_relationInfo.Add(new Relation(tblCatalog, node));
                            }
                            else if (node.Name.Equals("concordia:join"))
                            {
                                if (ti.m_joinInfo == null)
                                    ti.m_joinInfo = new JoinInfo();
                    ti.m_joinInfo.Add(new Join(node));
                }
                else if (node.Name.Equals("concordia:directive"))
                {
                    if (ti.m_directiveInfo == null)
                        ti.m_directiveInfo = new DirectiveInfo();
                    ti.m_directiveInfo.Add(new Directive(node));
                }
            }
        }
    }
}
}

private static bool m_bSucceeded = true;
public static void ValidationCallback(object sender, ValidationEventArgs args)
{
    m_bSucceeded = false;
    Console.WriteLine(args.Message);
}

public string ElementName
{
    get
    {
        return m_xmlSchemaElement.Name;
    }
    set
    {
        m_xmlSchemaElement.Name = value;
    }
}

public string TableName
{
    get
    {
        if (m_strTableName == null)
            return this.ElementName; // default value
        else
            return m_strTableName;
    }

    set { m_strTableName = value; }
}

public ArrayList LocalColumns { get { return m_lstColumns; } }

public ColumnInfo LocalColumn(int i) { return (ColumnInfo)m_lstColumns[i]; }

public ColumnInfo Column(object key)
{
    ColumnInfo ci;
    if ((ci = (ColumnInfo)m_tblColumns[key]) == null && m_superTableInfo != null)
    {
        ci = m_superTableInfo.Column(key);
    }
    return ci;
}

public ColumnInfo IDREF
{
    get
    {
        foreach (ColumnInfo ci in m_lstColumns)
        {
            if (ci.XmlType.Equals("IDREF"))
                return ci;
        }
        return null;
    }
}

public ArrayList InnerList { get { return m_lstInnerTables; } }

public TableInfo Outter
{
    get
    {
        if (m_relationInfo != null && m_relationInfo.AbstractRelation != null)
        {

            return m_relationInfo.AbstractRelation.Parent;
        }
        return null;
    }
}

public RelationInfo RelationInfo { get { return m_relationInfo; } }

public JoinInfo JoinInfo { get { return m_joinInfo; } }

public DirectiveInfo DirectiveInfo { get { return m_directiveInfo; } }

public TableInfo Super { get { return m_superTableInfo; } }

public ArrayList HeirList
{
    get
    {
        return m_lstHeirs;
    }
    set
    {
        m_lstHeirs = value;
    }
}

public override string ToString()
{
```

```
            String str = String.Format("TAB(EN:{0}|TN:{1})", ElementName, TableName);
            if (m_superTableInfo != null)
                str += String.Format(" : {0}", m_superTableInfo.ElementName);
            str += "\n";
            if (m_lstHeirs != null)
            {
                str += "SUCCESSORS: ";
                foreach (TableInfo ti in m_lstHeirs)
                    str += ti.ElementName + " ";
                str += "\n";
            }
            if (m_relationInfo != null)
                str += m_relationInfo;
            if (m_directiveInfo != null)
                str += m_directiveInfo;
            foreach (ColumnInfo ci in m_lstColumns)
            {
                str += "    " + ci + "\n";
            }
            return str;
        }
    }
}
```

ColumnInfo.cs

```
using System;
using System.Collections;
using System.Data;
using System.Xml;
using System.Xml.Schema;

namespace DataTransferProject.Schema
{
    /// <summary>
    /// class for column information.
    /// </summary>
    public abstract class ColumnInfo
    {
        protected TableInfo     m_tableInfo;
        protected bool          m_bForeignKey = false;
        protected SqlDbType  m_dbType;

        protected string      m_strColumnName = null;
        protected bool        m_bXmlOnly = false;

        public ColumnInfo(TableInfo ti)
        {
            m_tableInfo = ti;
        }

        public abstract string  ElementName { get;}
        public abstract decimal MinOccurs { get; }
public abstract decimal MaxOccurs { get; }
public abstract string  XmlType { get; }

public virtual bool IsElement { get { return false; } }
public virtual bool IsAttribute { get { return false; } }

// Gets the TableInfo to which the column belongs to.
public TableInfo TableInfo { get { return m_tableInfo; } }

public string ColumnName
{
    get
    {
        if (m_strColumnName == null)
        {
            return ElementName; // default value
        }
        else
        {
            return m_strColumnName;
        }
    }

    set { m_strColumnName = value; }
}

public bool IsXmlOnly { get { return m_bXmlOnly; } }
public SqlDbType DbType { get { return m_dbType; } }

public bool IsFK
{
    get { return (m_bForeignKey || XmlType.Equals("IDREF")); }
    set { m_bForeignKey = value; }
}

public virtual bool IsAbstract
{
    get
    {
        return (MaxOccurs == 0 && MinOccurs == 0);
    }
}

protected SqlDbType ConvertXmlType(string strXmlType)
{
    if (strXmlType.Equals("string"))
        return SqlDbType.VarChar;
    else if (strXmlType.Equals("dateTime"))
        return SqlDbType.DateTime;
    else if (strXmlType.Equals("integer"))
        return SqlDbType.Int;
    else if (strXmlType.Equals("float"))
        return SqlDbType.Float;
    else if (strXmlType.Equals("ID") || strXmlType.Equals("IDREF"))
        return SqlDbType.VarChar;
```

```
            else
            {
                throw new NotSupportedException("Unrecognized data type " + strXmlType);
            }
        }

        public override string ToString()
        {
            return String.Format("COL(EN:{0}|CN:{1}@{2})", ElementName, ColumnName,
                            m_tableInfo.ElementName);
        }
    }

    class ElementCI : ColumnInfo
    {
        private XmlSchemaElement m_xmlSchemaElement;

        public ElementCI(XmlSchemaElement element, TableInfo ti) : base(ti)
        {
            m_xmlSchemaElement = element;
            m_dbType = ConvertXmlType(element.SchemaTypeName.Name);

            // Handle "Concordia" namespace attributes
            if (element.UnhandledAttributes != null)
            {
                foreach (XmlAttribute attr in element.UnhandledAttributes)
                {
                    if (attr.Name.Equals("concordia:column"))
                    {
                        m_strColumnName = attr.Value;
                    }
                    else if (attr.Name.Equals("concordia:xmlOnly"))
                    {
                        m_bXmlOnly = !(attr.Value.Equals("0") || attr.Value.Equals("false"));
                    }
                    else
                    {
                        throw new NotSupportedException("Unrecognized attribute " +
attr.Name);
                    }
                }
            }
        }

        public override string ElementName { get { return m_xmlSchemaElement.Name; } }
        public override decimal MinOccurs { get { return m_xmlSchemaElement.MinOccurs;
}}
        public override decimal MaxOccurs { get { return
m_xmlSchemaElement.MaxOccurs;}}
        public override bool IsElement { get { return true; } }
        public override string XmlType { get { return
                m_xmlSchemaElement.SchemaTypeName.Name; } }
    }

    class AttributeCI : ColumnInfo
    {
        private XmlSchemaAttribute m_xmlSchemaAttribute;

        public AttributeCI(XmlSchemaAttribute attribute, TableInfo ti) : base(ti)
        {
            m_xmlSchemaAttribute = attribute;
            m_dbType = ConvertXmlType(attribute.SchemaTypeName.Name);

            // Handle Concordia namespace attributes
            if (attribute.UnhandledAttributes != null)
            {
                foreach (XmlAttribute attr in attribute.UnhandledAttributes)
                {
                    if (attr.Name.Equals("concordia:column"))
                    {
                        m_strColumnName = attr.Value;
                    }
                    else if (attr.Name.Equals("concordia:xmlOnly"))
                    {
                        m_bXmlOnly = !(attr.Value.Equals("0") || attr.Value.Equals("false"));
                    }
                    else
                    {
                        throw new NotSupportedException("Unrecognized attribute " +
attr.Name);
                    }
                }
            }
        }

        public override string ElementName { get { return m_xmlSchemaAttribute.Name; } }
        public override decimal MinOccurs { get { return 1; } }
        public override decimal MaxOccurs { get { return 1; } }
        public override bool IsAttribute { get { return true; } }
        public override string XmlType { get { return
                        m_xmlSchemaAttribute.SchemaTypeName.Name; } }
    }
}
```

RelationInfo.cs

```
using System;
using System.Collections;
using System.Xml;
using System.Xml.Schema;

namespace DataTransferProject.Schema
{
    /// <summary>
    /// Class for relations between tables
    /// </summary>
    public class RelationInfo
    {
```

```
    private ArrayList m_lstRelations;

    internal RelationInfo()
    {
        m_lstRelations = new ArrayList();
    }

    public int Count
    {
        get { return m_lstRelations.Count; }
    }

    public Relation this[int index]
    {
        get { return (Relation)m_lstRelations[index]; }
    }

    public Relation AbstractRelation
    {
        get
        {
            foreach (Relation r in m_lstRelations)
            {
                if (r.IsAbstract)
                    return r;
            }
            return null;
        }
    }

    public int Add(Relation r)
    {
        return m_lstRelations.Add(r);
    }

    public override string ToString()
    {
        String str = "RELATIONS:\n";
        for (int i = 0; i < this.Count; i++)
        {
            str += this[i].ToString() + "\n";
        }
        return str;
    }
}

public class Relation
{
    private TableInfo    m_parentTableInfo = null;
    private ColumnInfo[] m_parentKeys = null;

    private TableInfo    m_childTableInfo = null;
    private ColumnInfo[] m_childKeys = null;

    private bool         m_bIsAbstract = false;
```

```
public Relation(Hashtable tblCatalog, XmlNode xmlNode)
{
    // get parent table
    XmlAttribute attr = GetAttribute(xmlNode.Attributes, "parent");
    m_parentTableInfo = (TableInfo)tblCatalog[attr.Value];

    // get parent key
    attr = GetAttribute(xmlNode.Attributes, "parentKey");
    string[] keys = attr.Value.Split(new char[] {' '});
    m_parentKeys = new ColumnInfo[keys.Length];
    for (int i = 0; i < keys.Length; i++)
    {
        m_parentKeys[i] = m_parentTableInfo.Column(keys[i]);
    }

    // get child table
    attr = GetAttribute(xmlNode.Attributes, "child");
    m_childTableInfo = (TableInfo)tblCatalog[attr.Value];

    // get child key
    attr = GetAttribute(xmlNode.Attributes, "childKey");
    keys = attr.Value.Split(new char[] {' '});
    m_childKeys = new ColumnInfo[keys.Length];
    for (int i = 0; i < keys.Length; i++)
    {
        m_childKeys[i] = m_childTableInfo.Column(keys[i]);
        m_childKeys[i].IsFK = true;
    }

    // if relation is abstract
    if ((attr = GetAttribute(xmlNode.Attributes, "abstract")) != null)
    {
        m_bIsAbstract = !(attr.Value.Equals("0") || attr.Value.Equals("false"));
    }
}

public TableInfo Parent
{
    get { return m_parentTableInfo; }
    set { m_parentTableInfo = value; }
}

public ColumnInfo[] ParentKeys
{
    get { return m_parentKeys; }
    set { m_parentKeys = value; }
}

public TableInfo Child
{
    get { return m_childTableInfo; }
    set { m_childTableInfo = value; }
}
```

```
        public ColumnInfo[] ChildKeys
        {
            get { return m_childKeys; }
            set { m_childKeys = value; }
        }

        public bool IsAbstract
        {
            get { return m_bIsAbstract; }
        }

        private XmlAttribute GetAttribute(XmlAttributeCollection attributes, string strName)
        {
            foreach (XmlAttribute attr in attributes)
            {
                if (attr.Name.Equals(strName))
                    return attr;
            }
            return null;
        }

        public override string ToString()
        {
            return String.Format("{0}.{1} -> {2}.{3}", m_childTableInfo.ElementName,
m_childKeys[0], m_parentTableInfo.ElementName, m_parentKeys[0]);
        }
    }
}
```

DirectiveInfo.cs

```
using System;
using System.Collections;
using System.Xml;
using System.Xml.Schema;

namespace DataTransferProject.Schema
{
    /// <summary>
    /// class for directive information
    /// </summary>
    public class DirectiveInfo
    {
        ArrayList m_lstDirectives;

        internal DirectiveInfo()
        {
            m_lstDirectives = new ArrayList();
        }

        public int Count
        {
            get { return m_lstDirectives.Count; }
        }

        public Directive this[int index]
        {
            get { return (Directive)m_lstDirectives[index]; }
        }

        public int Add(Directive d)
        {
            return m_lstDirectives.Add(d);
        }

        public override string ToString()
        {
            String str = "Directives:\n";
            for (int i = 0; i < this.Count; i++)
            {
                str += this[i].ToString() + "\n";
            }
            return str;
        }
    }


    public class Directive
    {
        private string m_strSql = null;

        public Directive(XmlNode node)
        {
            foreach (XmlAttribute attr in node.Attributes)
            {
                if (attr.Name.Equals("sql"))
                    m_strSql = attr.Value;
            }
        }

        public string Sql
        {
            get { return m_strSql; }
            set { m_strSql = value; }
        }

        public override string ToString()
        {
            return String.Format("sql: {0}", m_strSql);
        }

    }
}
```

JoinInfo.cs

```
using System;
using System.Collections;
using System.Xml;
using System.Xml.Schema;

namespace DataTransferProject.Schema
{
    /// <summary>
    /// Class for join information
    /// </summary>
    public class JoinInfo
    {
        ArrayList m_lstJoins;

        internal JoinInfo()
        {
            m_lstJoins = new ArrayList();
        }

        public int Count
        {
            get { return m_lstJoins.Count; }
        }

        public Join this[int index]
        {
            get { return (Join)m_lstJoins[index]; }
        }

        public int Add(Join j)
        {
            return m_lstJoins.Add(j);
        }
    }

    public class Join
    {
        private string        parent = null;
        private string[]      parentKeys = null;
        private string        child = null;
        private string[]      childKeys = null;

        public Join(XmlNode xmlNode)
        {
            foreach (XmlAttribute attr in xmlNode.Attributes)
            {
                if (attr.Name.Equals("parent"))
                    parent = attr.Value;
                else if (attr.Name.Equals("parentKey"))
                    parentKeys = attr.Value.Split(new char[] {' '});
                else if (attr.Name.Equals("child"))
                    child = attr.Value;
                else if (attr.Name.Equals("childKey"))
                    childKeys = attr.Value.Split(new char[] {' '});
            }
        }
        public string Parent
        {
            get { return parent; }
            set { parent = value; }
        }

        public string[] ParentKeys
        {
            get { return parentKeys; }
            set { parentKeys = value; }
        }

        public string Child
        {
            get { return child; }
            set { child = value; }
        }

        public string[] ChildKeys
        {
            get { return childKeys; }
            set { childKeys = value; }
        }
    }
}
```

ExportToXML.cs

```
using System;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Xml;
using DataTransferProject.Schema;

namespace DataTransferProject
{
    /// <summary>
    /// Export data from source database to the XML data file
    /// </summary>
    public class ExportToXml
    {
        private const string strDOCROOT = "school_database";
        private SqlConnection m_conn = null;
        private XmlTextWriter m_xmlTextWriter = null;
        private Hashtable m_tblCatalog = null;
        private Hashtable m_tblAdapters = null;

        public ExportToXml(string strSource, string strDB)
        {
```

```
        // construct DB connection string
        string strConnection = String.Format("Data Source={0};Database={1};Integrated
    Security=SSPI;", strSource, strDB);
        // create DB connection object
        m_conn = new SqlConnection(strConnection);
    }

    public void Execute(string schemaFile, string xmlFile)
    {
        // create XML text writer object for exporting data from DB
        m_xmlTextWriter = new XmlTextWriter(xmlFile, Encoding.GetEncoding("utf-8"));
        // create hash table for sqldata adapters, each schema element has an adapter
        m_tblAdapters = new Hashtable();

        // parse the XML schema file to assemble tables, columns and relationships
        m_tblCatalog = TableInfo.Compile(new FileStream(schemaFile, FileMode.Open));

        // start from the root element
        TableInfo ti = (TableInfo)m_tblCatalog[strDOCROOT];
        m_xmlTextWriter.WriteStartDocument(true);

        // write the root element
        m_xmlTextWriter.WriteStartElement(strDOCROOT);

        // open DB connection
        m_conn.Open();
        // loop through all referenced tables
        foreach (string elementName in ti.InnerList)
        {
            ExecuteOneTable((TableInfo)m_tblCatalog[elementName], null, null);
        }

        // write end element
        m_xmlTextWriter.WriteEndElement();
        // end exporting data
        m_xmlTextWriter.WriteEndDocument();
        // close xml text writer
        m_xmlTextWriter.Close();
        // close DB connection
        m_conn.Close();

    }// end of Execute

    private void ExecuteOneTable(TableInfo ti, TableInfo directParent, Object[]
passValues)
    {
        // get the adapter for the element
        SqlDataAdapter adapter = (SqlDataAdapter)m_tblAdapters[ti.ElementName];
        if (adapter == null)
        {
            adapter = new SqlDataAdapter(FormatSqlSelect(ti, directParent), m_conn);
            m_tblAdapters.Add(ti.ElementName, adapter);

            // bind parameters
            if (directParent != null)
            {
                Relation r = ti.RelationInfo.AbstractRelation;
                for (int i = 0; i < r.ChildKeys.Length; i++)
                {
                    adapter.SelectCommand.Parameters.Add("@" + i,
r.ChildKeys[i].DbType);
                }
            }
        }

        // bind value
        for (int i = 0; i < adapter.SelectCommand.Parameters.Count; i++)
        {
            adapter.SelectCommand.Parameters[i].Value = passValues[i];
        }

        DataSet ds = new DataSet();
        try
        {
            adapter.Fill(ds);
        }
        catch (Exception error)
        {
            Console.WriteLine("Exception catch: {0}", error.Message.ToString());
        }

        foreach (DataTable dt in ds.Tables)
        {
            foreach (DataRow row in dt.Rows)
            {
                m_xmlTextWriter.WriteStartElement(ti.ElementName);

                int colNum = 0;
                foreach (ColumnInfo ci in ti.LocalColumns)
                {
                    if (!ci.IsAbstract && !row.IsNull(colNum++))
                    {
                        // write start element
                        if (ci.IsElement)
                            m_xmlTextWriter.WriteStartElement(ci.ElementName);
                        else
                            m_xmlTextWriter.WriteStartAttribute("id", null);

                        // write text of the element
                        m_xmlTextWriter.WriteString(row[colNum - 1].ToString().Trim());

                        // write end element
                        if (ci.IsElement)
                            m_xmlTextWriter.WriteEndElement();
                        else
                            m_xmlTextWriter.WriteEndAttribute();
                    }
                }

                if (ti.InnerList != null)
                {
                    foreach (string elementName in ti.InnerList)
                    {
                        TableInfo child = (TableInfo)m_tblCatalog[elementName];
                        Relation r = child.RelationInfo.AbstractRelation;
                        Object[] values = new Object[r.ParentKeys.Length];
                        for (int i = 0; i < values.Length; i++)
                        {
                            values[i] = row[colNum++];
                        }

                        ExecuteOneTable(child, ti, values);
                    }
                }

                m_xmlTextWriter.WriteEndElement();
            }
        }
    }// end of ExecuteOneTable

    private string FormatSqlSelect(TableInfo ti, TableInfo directParent)
    {
        StringBuilder strSql = new StringBuilder();

        if (ti.DirectiveInfo != null)
        {
            for (int i = 0; i < ti.DirectiveInfo.Count; i++)
            {
                string strSqlSelect = "";

                if (ti.InnerList != null)
                {
                    foreach (string elementName in ti.InnerList)
                    {
                        TableInfo child = (TableInfo)m_tblCatalog[elementName];
                        Relation r = child.RelationInfo.AbstractRelation;

                        foreach (ColumnInfo key in r.ParentKeys)
                        {
                            strSqlSelect += "," + FormatSqlColumn(ti.TableName,
                                            ti.Column(key.ElementName).ColumnName);
                        }
                    }
                }

                string tmp = ti.DirectiveInfo[i].Sql.ToUpper();
                strSql.Append(tmp.Insert(tmp.IndexOf("FROM") - 1, strSqlSelect));
            }
        }
        else
        {
            strSql.Append("SELECT ");

            // SELECT list for columns of the table
            foreach (ColumnInfo ci in ti.LocalColumns)
```

```
{
    if (!ci.IsAbstract)
    {
        strSql.AppendFormat("{0},", FormatSqlColumn(ti.TableName,
                                ci.ColumnName));
    }
}
// remove the extra comma at the end
strSql.Remove(strSql.Length - 1, 1);

// SELECT list for supplying data to inner elements(reference elements list)
if (ti.InnerList != null)
{
    foreach (string elementName in ti.InnerList)
    {
        TableInfo child = (TableInfo)m_tblCatalog[elementName];
        Relation r = child.RelationInfo.AbstractRelation;

        foreach (ColumnInfo key in r.ParentKeys)
        {
            strSql.AppendFormat(",{0}", FormatSqlColumn(ti.TableName,
                        ti.Column(key.ElementName).ColumnName));
        }
    }
}

// add tables and relations into FROM and WHERE clauses
string strSqlFrom = String.Format("FROM {0}", ti.TableName);
string strSqlWhere = null;
if (ti.JoinInfo != null)
{
    for (int i = 0; i < ti.JoinInfo.Count; i++)
    {
        Join join = ti.JoinInfo[i];
        if (strSqlFrom.IndexOf(join.Child) == -1)
            strSqlFrom += "," + join.Child;
        if (strSqlFrom.IndexOf(join.Parent) == -1)
            strSqlFrom += "," + join.Parent;

        for (int j = 0; j < join.ChildKeys.Length; j++)
        {
            if (strSqlWhere == null)
                strSqlWhere = "WHERE";
            else
                strSqlWhere += " AND";
            strSqlWhere += String.Format(" {0}.{1}={2}.{3}", join.Parent,
                        join.ParentKeys[i], join.Child, join.ChildKeys[i]);
        }
    }
}

if (ti.RelationInfo != null)
{
    string strSqlSelect = strSql.ToString();
    for (int i = 0; i < ti.RelationInfo.Count; i++)
```

```
            {
                Relation r = ti.RelationInfo[i];

                // if the parent table of this relation isn't used in select list, then
                // no need to join this table
                if (strSqlSelect.IndexOf(r.Parent.TableName) == -1)
                    continue;

                if (strSqlFrom.IndexOf(r.Parent.TableName) == -1)
                    strSqlFrom += "," + r.Parent.TableName;

                for (int j = 0; j < r.ChildKeys.Length; j++)
                {
                    if (strSqlWhere == null)
                        strSqlWhere = "WHERE";
                    else
                        strSqlWhere += " AND";
                    strSqlWhere += String.Format(" {0}.{1}={2}.{3}",
                        r.Parent.TableName, r.ParentKeys[i].ColumnName,
                        r.Child.TableName, r.ChildKeys[i].ColumnName);
                }
            }
        }

        if (directParent != null)
        {
            Relation r = ti.RelationInfo.AbstractRelation;
            for (int i = 0; i < r.ChildKeys.Length; i++)
            {
                if (strSqlWhere == null)
                    strSqlWhere = "WHERE";
                else
                    strSqlWhere += " AND";
                strSqlWhere += String.Format(" {0}.{1}=@{2}", ti.TableName,
                            r.ChildKeys[i].ColumnName, i);
            }
        }

        strSql.AppendFormat(" {0} {1}", strSqlFrom, strSqlWhere);
    }

    Console.WriteLine("{0}: {1}", ti.ElementName, strSql.ToString());
    return strSql.ToString();
}// end of FormatSqlSelect

private string FormatSqlColumn(string tn, string cn)
{
    // in the case of 1. SQL function calls, 2. full-qualified name, 3. NULL
    Regex regex = new Regex("[A-Za-z]+\\(.+\\)|.+\\..+|(NULL|null)");
    if (regex.IsMatch(cn))
    {
        return cn;
    }
    else
    {
```

```
                return (tn + "." + cn);
            }
        }//end of FormatSqlColumn

        [STAThread]
        public static void Main(string[] args)
        {
            ExportToXml exp2xml = new ExportToXml("localhost", "school_a");
            exp2xml.Execute("../../../schoolA.xsd", "../../../database.xml");
        }
    }// end of class ExportToXml
}
```

XmlToDatabase.cs

```
using System;
using System.IO;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Text;
using System.Xml;
using System.Xml.Schema;
using DataTransferProject.Schema;

namespace XmlToDatabase
{
    /// <summary>
    /// Import data from XML data file to the target database
    /// </summary>
    public class XmlToDatabase
    {
        private const string strDOCROOT = "school_database";
        private SqlConnection m_conn = null;
        private XmlDocument m_xmlDoc = null;
        private Hashtable m_tblCatalog = null;
        private Hashtable m_tblCache; // now it's only useful for ID/IDREF cases

        public XmlToDatabase(string strSource, string strDB)
        {
            string strConnection = String.Format("Data Source={0};Database={1};Integrated
        Security=SSPI;", strSource, strDB);
            m_conn = new SqlConnection(strConnection);
            m_tblCache = new Hashtable();
            m_xmlDoc = new XmlDocument();
        }

        public void Execute(string schemafile, string xmlfile)
        {
            XmlSchema xSchema = null;

            // compile schemafile, to assemble tableinfo, columninfo, etc.
```

```
m_tblCatalog = TableInfo.Compile(new FileStream(schemafile, FileMode.Open),
                                   ref xSchema);
// load and validate xmlfile against xSchema
ReadXml(xSchema, xmlfile);

TableInfo tiRoot = (TableInfo)m_tblCatalog[strDOCROOT];
Hashtable tblDependency = new Hashtable();
foreach (string strTableName in tiRoot.InnerList)
{
   TableInfo tiRefTable = (TableInfo)m_tblCatalog[strTableName];
   Hashtable tblDependents = new Hashtable();
   tblDependency.Add(tiRefTable, tblDependents);
   BuildDependency(tiRefTable, tblDependents);
   Console.Write("{0} depends on ", tiRefTable.ElementName);
   foreach (TableInfo ti in tblDependents.Keys)
   {
      Console.Write("{0} ", ti.ElementName);
   }
   Console.WriteLine();
}
// open the database connection
m_conn.Open();


while (tblDependency.Count > 0)
{
   // create enumerator object for reading the collection data
   IEnumerator it = tblDependency.Keys.GetEnumerator();
   // position the enumerator to the first element
   it.MoveNext();
   TableInfo current = (TableInfo)it.Current;

   while (true)
   {
      TableInfo dependent = null;
      foreach (TableInfo ti in ((Hashtable)tblDependency[current]).Keys)
      {
         if (tblDependency.Contains(ti))
         {
            dependent = ti;
            break;
         }
      }
      // found a candidate
      if (dependent == null)
      {
         tblDependency.Remove(current);
         Console.WriteLine("execute " + current.ElementName);
         ExecuteOneTable(current, m_xmlDoc.SelectSingleNode(strDOCROOT),
                         null);
         break;
      }

      current = dependent;
   }
```

```
        }

        // close the database connection
        m_conn.Close();
    }

    public void ReadXml(XmlSchema schema, string xmlfile)
    {
        // create namespace manager object
        XmlNamespaceManager nsManager =
                        new XmlNamespaceManager(new NameTable());
        // create context info object for parsing
        XmlParserContext psContext =
                        new XmlParserContext(null, nsManager, null, XmlSpace.Default);

        // get validator
        XmlValidatingReader xmlValidator =
            new XmlValidatingReader(new FileStream(xmlfile, FileMode.Open),
            XmlNodeType.Document, psContext);
        // validate against XSD schema type
        xmlValidator.ValidationType = ValidationType.Schema;
        // assign schema
        xmlValidator.Schemas.Add(schema);
        // register event handler for validation errors
        xmlValidator.ValidationEventHandler +=
                    new ValidationEventHandler(ValidationCallback);

        // parse and load the XML document
        m_xmlDoc.Load(xmlValidator);

        // close the validator
        xmlValidator.Close();
    }

    public void ValidationCallback(object sender, ValidationEventArgs args)
    {
        Console.WriteLine(args.Message);
    }

    private void BuildDependency(TableInfo ti, Hashtable dependents)
    {
        if (ti.RelationInfo != null)
        {
            for (int i = 0; i < ti.RelationInfo.Count; i++)
            {
                if (ti.RelationInfo[i].IsAbstract)
                    continue;

                // build a list of all tables needed to be traced
                ArrayList lstTables;
                if (ti.RelationInfo[i].Parent.HeirList != null)
                {
                    lstTables = (ArrayList)ti.RelationInfo[i].Parent.HeirList.Clone();
                }
                else
                {
                    lstTables = new ArrayList();
                }
                // avoid self dependency (recursive)
                if (ti != ti.RelationInfo[i].Parent)
                {
                    lstTables.Add(ti.RelationInfo[i].Parent);
                }

                // trace up to the outtest element
                foreach (TableInfo table in lstTables)
                {
                    TableInfo curr = table;
                    while (curr.Outter != null)
                    {
                        curr = curr.Outter;
                    }
                    if (!dependents.Contains(curr))
                    {
                        dependents.Add(curr, null);
                    }
                }
            }
        }

        if (ti.InnerList != null)
        {
            foreach (string strTableName in ti.InnerList)
            {
                BuildDependency((TableInfo)m_tblCatalog[strTableName], dependents);
            }
        }
    }

    private void ExecuteOneTable(TableInfo ti, XmlNode outter, object passvalue)
    {
        // create sqlCommand object
        SqlCommand sqlCommand = MakeInsertCommand(ti);
        Console.WriteLine(sqlCommand.CommandText);

        Stack stack = new Stack();

        foreach (XmlNode row in outter.SelectNodes(ti.ElementName))
        {
            ColumnInfo ci = ti.IDREF;
            if (ci != null)
            {
                if (m_tblCache.Contains(row.Attributes["id"].InnerText))
                    continue;
                stack.Clear();
                stack.Push(row);

                XmlNode column, curr = row;
                while ((column = curr.SelectSingleNode(ci.ElementName)) != null)
                {
                    curr = outter.SelectSingleNode(ti.ElementName + "[@id='" +
                column.InnerText + "']");
                    if (m_tblCache.Contains(curr.Attributes["id"].InnerText))
                        break;
                    stack.Push(curr);
                }

                while (stack.Count > 0)
                    ExecuteOneRow(sqlCommand, ti, (XmlNode)stack.Pop(), passvalue);
            }
            else
            {
                ExecuteOneRow(sqlCommand, ti, row, passvalue);
            }
        }
    }

    private Guid ExecuteOneRow(SqlCommand command, TableInfo ti, XmlNode row,
object passvalue)
    {
        Guid row_id;

        // subtyping
        if (ti.Super != null)
        {
            SqlCommand c2 = MakeInsertCommand(ti.Super);
            Console.WriteLine(c2.CommandText);
            c2.Parameters["@person_type"].Value = ti.ElementName;
            row_id = ExecuteOneRow(c2, ti.Super, row, passvalue);
        }
        else
        {
            row_id = Guid.NewGuid();
        }

        if (ti.IDREF != null)
        {
            m_tblCache.Add(row.Attributes["id"].InnerText, row_id);
        }

        // Bind value
        command.Parameters[0].Value = row_id;
        foreach (ColumnInfo ci in ti.LocalColumns)
        {
            if (ci.IsXmlOnly)
                continue;

            if (ci.IsAbstract)
            {
                command.Parameters["@" + ci.ColumnName].Value = passvalue;
            }
            else
            {
                XmlNode column = row.SelectSingleNode(ci.ElementName);
                if (column == null)
            {
                command.Parameters["@" + ci.ColumnName].Value = DBNull.Value;
            }
            else if (ci.XmlType.Equals("IDREF"))
            {
                command.Parameters["@" + ci.ColumnName].Value =
                                (Guid)m_tblCache[column.InnerText];
            }
            else
            {
                command.Parameters["@" + ci.ColumnName].Value = column.InnerText;
            }
        }
    }

    // Handle explicit foreign keys
    if (ti.RelationInfo != null)
    {
        for (int i = 0; i < ti.RelationInfo.Count; i++)
        {
            Relation r = ti.RelationInfo[i];
            if (r.IsAbstract)
                continue;

            SqlCommand sc = MakeSelectCommand(r);
            Console.WriteLine(sc.CommandText);

            for (int j = 0; j < r.ChildKeys.Length; j++)
            {
                XmlNode n = row.SelectSingleNode(r.ChildKeys[j].ElementName);
                if (n == null)
                {
                    sc.Parameters[j].Value = DBNull.Value;
                }
                else
                {
                    sc.Parameters[j].Value = n.InnerText;
                }
            }

            object rc = sc.ExecuteScalar();
            if (rc == null)
            {
                command.Parameters["@" + r.ChildKeys[0].ColumnName].Value =
                DBNull.Value;
            }
            else
            {
                command.Parameters["@" + r.ChildKeys[0].ColumnName].Value = rc;
            }
        }
    }

    for (int i = 0; i < command.Parameters.Count; i++)
    {
            Console.WriteLine("{0} <- {1}", command.Parameters[i].ParameterName,
            command.Parameters[i].Value);
        }

        command.ExecuteNonQuery();

        // Handle InnerList
        if (ti.InnerList != null)
        {
            foreach (string strTableName in ti.InnerList)
            {
                TableInfo child = (TableInfo)m_tblCatalog[strTableName];
                ExecuteOneTable(child, row, row_id);
            }
        }

        return row_id;
    }

    //
    // build sql command to insert to DB
    //
    private SqlCommand MakeInsertCommand(TableInfo ti)
    {
        // create sqlCommand object
        SqlCommand command = m_conn.CreateCommand();
        // build sql command string
        StringBuilder sql = new StringBuilder();

        // append table name
        sql.AppendFormat("INSERT INTO {0} (row_id", ti.TableName);

        // append column name
        if (ti.HeirList != null)
        {
            sql.Append(",person_type");
        }

        foreach (ColumnInfo ci in ti.LocalColumns)
        {
            if (ci.IsXmlOnly)
                continue;
            sql.AppendFormat(",{0}", ci.ColumnName);
        }

        // append values
        sql.Append(") VALUES (@row_id");
        command.Parameters.Add("@row_id", SqlDbType.UniqueIdentifier);

        if (ti.HeirList != null)
        {
            sql.Append(",@person_type");
            command.Parameters.Add("@person_type", SqlDbType.VarChar);
        }
```

```
            foreach (ColumnInfo ci in ti.LocalColumns)
            {
                if (ci.IsXmlOnly)
                    continue;
                sql.AppendFormat(",@{0}", ci.ColumnName);

                if (ci.IsFK)
                {
                    command.Parameters.Add("@" + ci.ColumnName,
                                    SqlDbType.UniqueIdentifier);
                }
                else
                {
                    command.Parameters.Add("@" + ci.ColumnName, ci.DbType);
                }
            }
            sql.Append(")");

            command.CommandText = sql.ToString();
            return command;
        }

        //
        // build sql command to select from DB
        //
        private SqlCommand MakeSelectCommand(Relation r)
        {
            SqlCommand command = m_conn.CreateCommand();

            // Compose a SQL insertion statement
            StringBuilder sql = new StringBuilder();
            sql.AppendFormat("SELECT row_id FROM {0} WHERE ", r.Parent.TableName);

            for (int i = 0; i < r.ParentKeys.Length; i++)
            {
                if (i > 0) { sql.Append(" AND "); }
                sql.AppendFormat("{0}.{1}=@{2}", r.Parent.TableName,
                            r.ParentKeys[i].ColumnName, i);
                command.Parameters.Add("@" + i, r.ParentKeys[i].DbType);
            }

            command.CommandText = sql.ToString();
            return command;
        }
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        public static void Main(string[] args)
        {
            XmlToDatabase xml2DB = new XmlToDatabase("localhost", "school_b");
            xml2DB.Execute("../../../schoolB.xsd", "../../../database.xml");
        }
    }
}
```