Abstract

The DBMS of various data models have proliferated into many companies, and become their legacy databases. Conventional databases are associated with a plurality of database models. Generally database models are distinct and not interoperable. Data stored in a database under a particular database model can be termed as "siloed data". Each database model acts as an individual silo such that data stored in one database silo is typically not readily accessible or interoperable with data stored in another database silo. Accordingly, a DBMS associated with a database silo as data stored under a first database model, is generally not interoperable with another database management system associated with another database silo as data stored under a second database model. This can limit the exchange of information stored in a database where those desiring to access the information are not employing a database management system associated with the database model related to the information.

There is a need to access these legacy databases using ODBC (open database connectivity). An ODBC is for the users to transform a legacy database into another legacy database. This thesis offers an architecture of Open Universal Database Gateway (OUDG) to supplement ODBC by transforming legacy database data into Flattened XML documents, and to transform Flattened XML document back into any other legacy database. The Flattened XML document is a mixture of relational and XML data models, which is user friendly and data standard on the Internet. Furthermore, Flattened XML document is a replication of legacy database, which is a backup copy of the legacy database in case of system failure, and can be used for internet computing and data processing in parallel, non-stop.

In other words, a source legacy database can be reengineered into a flattened XML document, which can be furthered reengineered into another target legacy database. As a result, a legacy database can be reengineered into another legacy database through Flattened XML document without loss of information. In this way, an user can access any legacy database by reengineering it into a legacy database which is accessible by the DBMS in his /her own computer. The result of reengineering database is information lossless by the preservation of their data semantics and data dependencies.

CITY UNIVERSITY OF HONG KONG
Qualifying Panel and Examination Panel

Surname:	WONG
First Name:	Ting Yan
Degree:	Master of Philosophy
College/Department:	Department of Computer Science

The Qualifying Panel of the above student is composed of:

Supervisor(s)

Dr. FONG Shi Piu Joseph	Department of Computer Science City University of Hong Kong

Co-supervisor(s)

Dr. WONG Hau San	Department of Computer Science City University of Hong Kong

Qualifying Panel Member(s)

Dr. LI Minming	Department of Computer Science City University of Hong Kong

This thesis has been examined and approved by the following examiners:

Dr. FONG Shi Piu Joseph	Department of Computer Science City University of Hong Kong
Dr. LI Minming	Department of Computer Science City University of Hong Kong
Prof. ZHOU Xiaofang	School of Information Technology & Electrical Engineering University of Queensland

Chapter 1 Introduction

Because of their historical importance and the existing user database for these DBMSs, these models and systems are now referred to as legacy database systems (Ramez & Shamkant, 2011, P.56). There are many type of legacy database since 1960s. In this thesis, we focus on 4 data models only.

(1) Network database.

Data structure: It is in flea structure which allows 2 owner records pointing to the same member, and each record can connect to any other record in a network "graph" structure. Johnson & Johnson is still using NDB (Raima users, 2014)

(2) Relational database.

Data structure: It is in table structure. Every relation is a table which must have a primary key with foreign key referring to a primary key of another table in values matching. NCR is still using RDB (Relational database users, 2014)

(3) Object-Oriented database.

Data structure: It is in class structure such that a class associating with another class by an object's stored OID (Object Identity) referring another class object OID. Also, a sub-class object can inherit data and method of a superclass object with the same OID which is system generated. Objectivity and Gemstone are OODBMS (Object-Oriented, 2014)

(4) XML.

Data structure: It is in a tree structure, with one root element. Elements are under root elements. Each element links with multiple sub-elements. Elements can also be linked by using IDREF attribute referring to another element attribute ID in the XML scheme DTD (Data Type Definition). Tomcat is still using XML. (XML users, 2014)

In fact, we consider both XML and hierarchical data models are in tree structure and therefore present them as an XML data model in this thesis.

A legacy system is any corporate computer system that isn't Internet-dependent.

Because of their historical importance and the existing user databases for these DBMSs, these models and systems are now referred to as legacy database systems.

(Reference: Ramez,E., Shamkant, B.(2011), "Database Systems, Models, Languages, Design , and Application programming", Pearson, 6th edition, P.56)

The following table show that RDB, OODB, XML, NDB are still being used in the industry. Therefore, we consider these 4 data model are "legacy" systems.

DBMS	Customers still using in industry today	Reference for evidence
RDB	NCR, Phoebe Putney Memorial Hospital, John Wayne Airport	https://www.oracle.com/search/customers/
NDB	Johnson & Johnson	http://raima.com/customers/
	IBM mainframe	www.ibm.com
OODB	Objectivity, Gemstone	http://www.objectivity.com/ http://www.gemstone.com/
	Orient Overseas Container Line (OOCL)	www.oocl.com
XML	Tomcat	https://tomcat.apache.org/tomcat-3.3-doc/serverxml.html

Because IBM is still using Hierarchical DBMS, so, there should be 5 current legacy databases.

The evolution of database technologies intends to meet different users requirements. For example, the complex Hierarchical and Network (Codasyl) databases (NDB) are good for business computing on the large mainframe computers. The user friendly relational databases (RDB) are good for end user computing on personal computers. The object-oriented databases (OODB) are good for multi-media computing on mini computers. The XML databases (XML DB) are good for Internet computing on the mobile devices. Table 1 shows the evolution of databases on various platforms. These are first generation Hierarchical and Network databases, second generation relational databases, and third generation post-relational such as Object-Oriented and XML databases.

Table 1 Platforms of Legacy Database technologies

	Network database	*Relational database*	*Object-Oriented database*	*XML database*
Computer Language	3GL Cobol / C	4GL, SQL/Visual Basic	4GL, OQL	XQuery, Web service
Operations	Batch Job	Triggers/ Stored procedures	Object-Oriented features	XQuery functions
User Interface	Text mode	Windows	Windows	Web pages
Machine	Mainframe	PC /Workstations	Web services/ Browsers	Web, Virtual machine

Flattened XML documents

Flattened XML documents are generic representation of any legacy database instance in any legacy database data model. It is because flattened XML structure combines tree structure and table structure data model, with relational database and object oriented database as a table structure data model and hierarchical database, network database and XML database as a tree structure data model. Therefore, Flattened XML can represent them as a data model.

Flattened XML can represent most data semantics just like other legacy database system, relational database, object oriented database, hierarchical database, network database and XML database. The model can represent the static data of five legacy data models only. It is not total representation of all legacy database data models.

Data semantic include ISA, cardinality, generalization:

ISA is a relationship between a superclass and a subclass. It is defined as, a subclass relation has same primary key as its superclass relation, and refers it as a foreign key in relational schema in isa relationship. It can also be implemented by a subclass inheriting its superclass's OID and attributes in object-oriented schema. It can also be implemented by an owner record that has same key as its member record in network schema via SET linkage. It can also be implemented by an element links one-to-one occurrence with its sub-element in XML schema.

Cardinality is one-to-one, one-to-many and many-to-many relationships set between two classes. 1:n is constructed by foreign key on "many" side referring to primary key on "one" side in relational schema. It can also be implemented by association attribute of a class object on "one" side pointing to objects on "many" side in another class in object-oriented schema. It can also be implemented by owner record occurrence on "one" side and member record occurrences on "many" side in network schema. It can also be implemented by element occurrence with IDREF on "many" side linking with element occurrence with ID on "one" side in XML schema.
As to m:n cardinality, it can be implemented by two 1:n cardinalities with 2 "one" side classes link with the same "many" side class.

Generalization is the relationship between one superclass and multiple subclasses.

They are in multiple isa relationships. For example A is a special kind of B, and C is also a special kind of B, then A and C subclasses can be generalized as B superclass. In relational schema, both superclass relation and subclass relations contain the same key, with subclass relations' keys referring to superclass key as foreign key in generalization. In object-oriented schema, multiple subclasses objects contain the same OID as their superclass object in generalization. In network schema, one owner record links with multiple member records through a SET in generalization. In XML, multiple subclass elements and their superclass element are in 1:1 linkage with same key attribute in generalization. Generalization can be implemented by multiple isa relationships such that multiple subclasses are generalized into one superclass.

Firstly, legacy database can be transformed into flattened XML documents which can be further transformed into another legacy database of Relational, Object-Oriented, Network and XML data models. Flattened XML document is a valid XML document which contains a collection of elements of various types and each element defines its own set of properties. The internal structure of the flattened XML document data file is a relational table structure. It has XML document tree structure syntax with internal elements in relational table structure. It replaces primary key with ID, and foreign key with sibling IDREF as follows:

```
<?xml version="1.0">
<root>
  <table1 ID="…" IDREF1="…" IDREF2="…" … IDREFN="…">
    <attribute1>…</attribute1>
      …
    <attributeN>…</attributeN>
  </table1>
  …
  <tableN ID="…" IDREF1="…" IDREF2="…" … IDREFN="…">
    <attribute1>…</attribute1>
    …
    <attributeN>…</attributeN>
  </tableN>
</root>
```

For each table, the name of the table determines its type name and the name of property (attribute) determines its property name. Each table defines an ID type attribute that can uniquely identify itself and there are optional multiple IDREF type attributes that can refer to this ID in other tables in their sibling elements. Each property XML element encloses a property value in a proper textual representation format. In order to ensure a flattened XML document instance to be valid, there must be either an internal or an external DTD document that defines the XML structures and attribute types, in particular for those ID and IDREF type attributes.

An open universal database gateway (OUDG) is a database middleware which provides more flexibility for the users to access legacy databases in their own chosen data model. In other words, users can apply OUDG to transform legacy databases into flattened XML documents, and then further transform them into user's own familiar legacy database for access. Since XML is the data standard on the Internet, it becomes information highway for user to access data.

The reason we choose flattened XML document is due to its openness for DBMS independence. All other data models are DBMS dependent. For example, an Oracle database can only be accessed by Oracle DBMS, and a MS SQL Server database can only be accessed by MS SQL Server DBMS. Nevertheless, users can access flattened XML documents on the Internet by Internet Explorer without programming. Furthermore, an Oracle user can access an MS SQL Server database after transforming the MS SQL Server database into flattened XML document, and then to Oracle database by OUDG.

Similarly, the reason we choose relational table structure for elements in the flattened XML document is that relational table structure has a strong mathematical foundation of relational algebra to implement the constraints of major data semantics such as cardinality, isa, generalization and aggregation to meet users' data requirements.

In fact, Vincent Lum (Lum, V.Y, 1976) attempted to propose a similar method by using sequential file as the medium for data conversion between legacy databases in logical level approach. But in his model, the source and target systems are limited to Hierarchical database, network database and relational database. This thesis is a further enhancement to include object-oriented database and XML.

The OUDG can transform legacy databases into flattened XML document, and then further transform the flattened XML document into another target legacy database of relational, object-oriented, XML or network. The result is that OUDG allows users transform a source legacy database into another target legacy database which is accessible in user's computer.

This thesis offers flattened XML documents as universal database medium for the interoperability of all legacy databases that can be accessed by the users using their own familiar legacy database language via OUDG. We consider hierarchical data model same as XML data model because they are all in tree structure. The five proprietary legacy data models can be interchangeable into flattened XML document as universal database as shown in Figure 1.

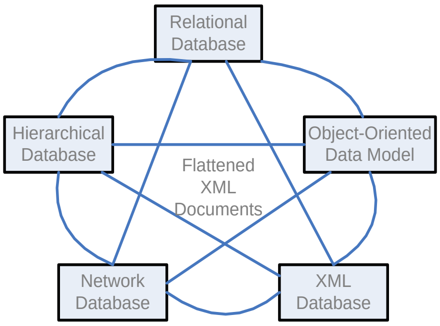


Figure 1 Cross model platform for Legacy Databases via Flattened XML documents

OUDG has 2 phases:

Phase I: transform user's legacy database into flattened XML documents

Phase II: transform the flattened XML document into a target's legacy database

Each phase has 2 steps:

Step 1: schema translation from source DB to target DB

Step 2: data conversion from source DB into target DB according to the translated target DB schema

There is a benefit for the design. Through Flattened XML in the OUDG, all legacy database system can be converted into each other. So user can use any legacy database language to access other legacy databases.

Because of OUDG, legacy DB of RDB, XML DB, NDB, OODB, HDB and flattened XML can be interchangeable to each other. As a result, a company can convert all of its heterogeneous DB into a particular legacy DB or flattened XML, as homogeneous DB, which uses a combined DB model of users' choice.

"The five proprietary legacy data models can be interchangeable into flattened XML document as universal database as shown in Figure 1."

Because through XML, all legacy databases can be interchangeable, we can view them as a one legacy database system. The legacy database can converted to another through Flattened XML. Therefore, multiple legacy databases can be converted into one legacy database.

For example, RDB can be converted into XML through Flattened XML. So, the user can view the DB as XML. Similarly, NDB and OODB can be converted into XML through Flattened XML. So, the user can view the DB as XML.

For example, XML can be converted into RDB through Flattened XML. So, the user can view the DB as RDB. Similarly, NDB and OODB can be converted into RDB through Flattened XML. So, the user can view the DB as RDB.

For example, OODB can be converted into NDB through Flattened XML. So, the user can view the DB as NDB. Similarly, RDB and XML can be converted into XML through Flattened XML. So, the user can view the DB as XML.

For example, NDB can be converted into OODB through Flattened XML. So, the user can view the DB as OODB.

Problems:

(1) Currently most XML documents are stored in XML database and are created on demand by converting a few relations into an XML document. However, this approach lacks of data semantic constraints, and is restricted to relational data model only. It cannot be converted into other legacy data models such as object-oriented, network and XML, which is a problem for e-commerce companies to transform their production relational database into XML documents.

(2) Most legacy database systems are proprietary. Database vendors do not facilitate tools to export their databases to other legacy databases. Thus, companies need to use ODBC to access other legacy databases, ie, database with no DBMS to access their target DB in their computers, which requires programming with a lot of time effort.

(3) Most users cannot access all legacy databases because they do not know all legacy database languages. They rely on ODBC, which is not easy to learn.

(4) It is difficult to convert legacy databases in different data models because the data conversion of legacy database involves data models transformation.

Solution:

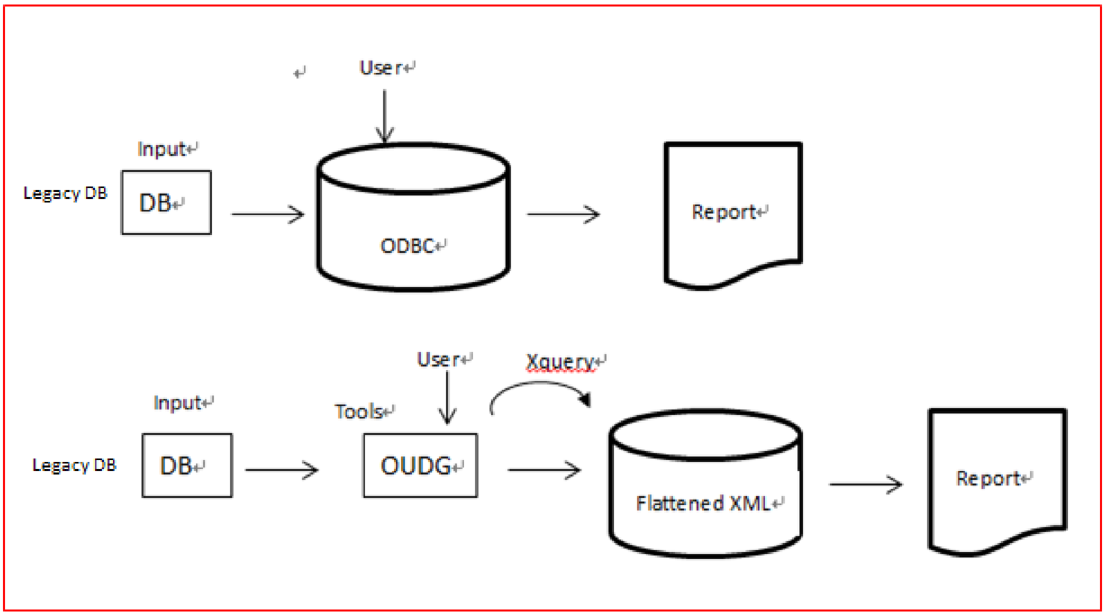
In computing, ODBC (Open Database Connectivity) is a standard programming language middleware API for accessing database management systems (DBMS). OUDG has a similar function of accessing different legacy database using Flattened XML as a middleware. (Reference http://en.wikipedia.org/wiki/Open_Database_Connectivity)

Both ODBC and OUDG allow users to access a legacy DB of his/her choice through different methods.

ODBC requires users to use an API programming solution to access a proprietary DB.

OUDG allows users to use DB conversion method to convert a legacy DB into a legacy DB model of his/her choice for the users to access.

As a result, OUDG is an alternative solution of ODBC for user to access a legacy DB without programming effort. Instead the user needs to use a software tool to transform the DB conversion, such as a DB middleware as shown below:



Internet provides an economical way for people to communicate around the world. It is obvious that businesses make use of this low cost communication method to communicate and exchange information with their business partners. XML document can be used in a myriad of ways across different platforms and in different applications.

This thesis offers a methodology that transforms legacy databases into an equivalent and maintainable flattened XML document to achieve the interoperability among all legacy databases because flattened XML document is user friendly and open for most computer systems on the Internet.

Through OUDG, users can use same database language access other legacy databases including relational, object-oriented, network and XML. The operation is more reliable and speedy because same data can be concurrently processed by legacy database and their replicated flattened XML document on the web at the same time.

Academic merit:

It is feasible to supplement ODBC by OUDG transforming legacy database into flattened XML document for database access. ODBC needs programming, but OUDG can be developed as an end user software tool.

Industrial merit:

The application of flattened XML document is for information highway on the Internet for data warehouse, decision support systems (Fong, Li & Huang, 2003), e-commerce, and cloud computing. The benefits are information sharing among users for database interoperability.

Application:

(1) OUDG can replace ODBC to access any legacy database by transforming them into a universal database of flattened XML document for accessing the same data.

Long-Term Impact:

At present, most database systems are proprietary. Each DBMS vendor has software tools which convert other legacy databases into their databases, but not vice versa for converting their own databases into other legacy databases (Hsiao & Kamel,1989) . The result makes legacy databases not open to each other. On the other hand, by using OUDG, any legacy database can be transformed into any other legacy database via flattened XML documents. The benefit is that data sharing and data conversion among legacy databases becomes possible.

Processing:

Pre-process: We can reverse engineer legacy database schema into legacy database conceptual schema to recover data semantics. Moreover, schema translation between legacy database schema and flattened XML schema must be performed before data transformation between them.

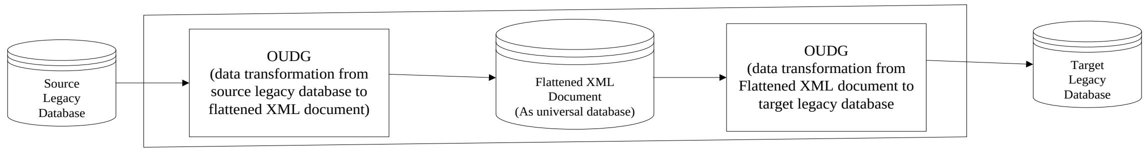
Step 1 Transform user's source legacy databases into flattened XML documents:
OUDG transforms the source legacy database into flattened XML document.

Step 2 Transform flattened XML documents into user's target legacy databases:
OUDG transforms the flattened XML documents into target's legacy database as shown in Figure 2.

OUDG as replacement for ODBC

Figure 2 shows the architecture of an open universal database gateway which transforms legacy databases into each other with different data models via flattened XML document as a supplement for open database connectivity.

OUDG as ODBC supplement



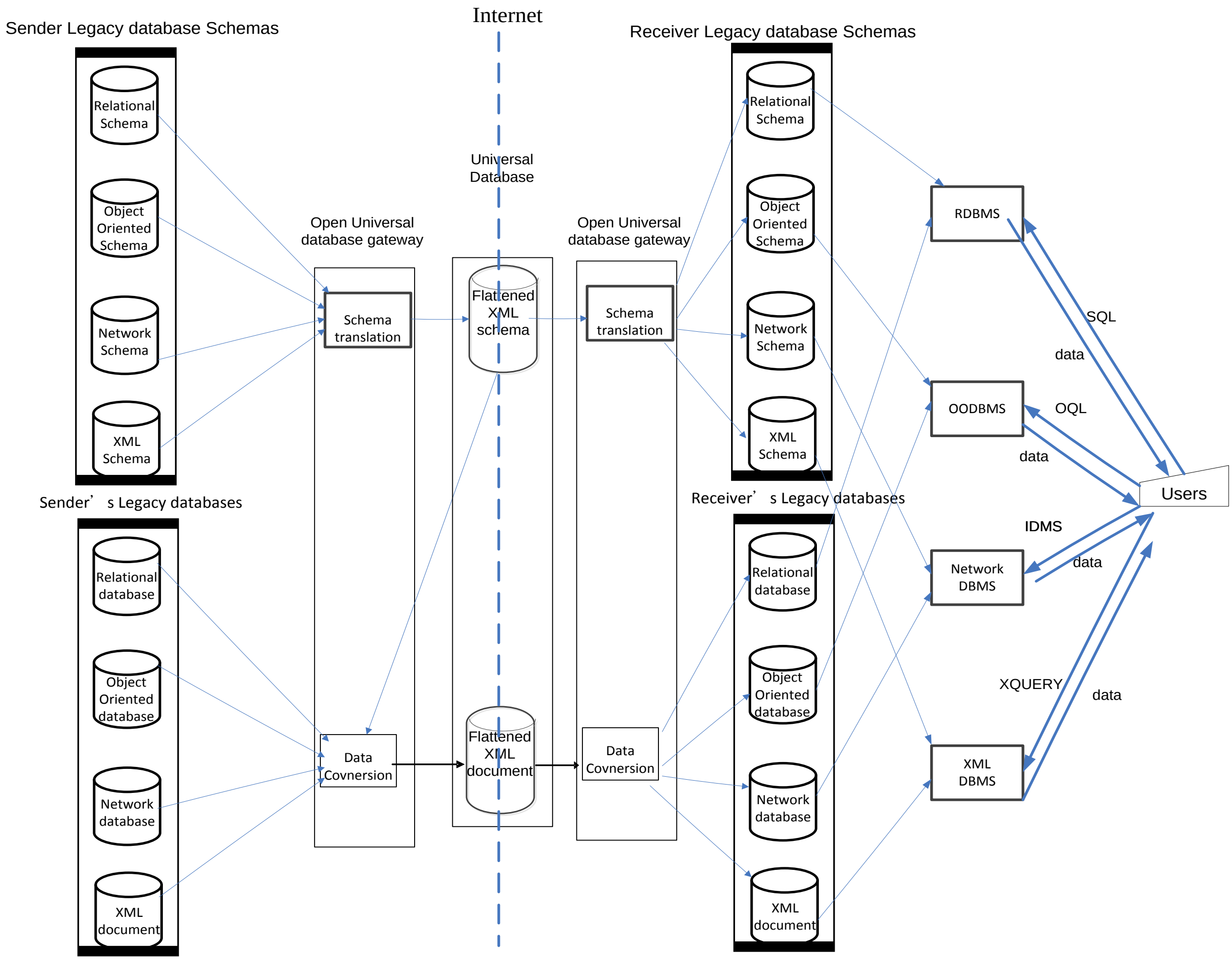


Figure 2 An open universal database gateway as supplement for open database connectivity

Data flows of Figure 2:

(1) The data semantics of an end user first legacy database schemas are captured into a meta data or conceptual schema.

(2) Legacy database schemas are mapped into a flattened XML document schema.

(3) The data of source legacy database are transformed into a flattened XML document.

(4) The flattened XML schemas are mapped into a target legacy database schemas.

(5) The flattened XML document are transformed into the target legacy database according to the mapped target legacy database schema.

Data Semantics preservation in legacy databases

Semantic constraints defined as, constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. We call these application-based or semantic constraints or business rules.

For example, the cardinality of one-to-one, one-to-many, many-to-many describe the data volume between two data fields which are their data constraints. (Referenced, P.64, Database Systems, Models, Languages, Design, and Application programming (6th edition), Ramez Elmasri, Shamkant B. Navathe, Pearson 2011)

Constraints are the general rules of data, eg, 1-to-many is a rule. Semantics constraints are the rules of relationship between data in the database.

Some of these rules can be enforced by database schema, but some of them cannot be enforced by database schema. So, those rules that cannot be enforced by database schema is database constraint. If the constraint of those rules cannot be enforced by database schema, programs must be used to enforce them.

If flattened XML enforced these semantic constraints, then, constraint can be interchangeable.

Moreover, some rules are very simple, eg, foreign key, 1-to-many. While some rules are complicated: ISA, categorization. So, flattened XML can represent all those rules, ie, rule of data.

<u>How to prove:</u>

They are 2 kinds semantic constraints

1) Primitive: semantic constraint such as cardinality and ISA
2) Other (Advanced data semantic constraint) such as generalization, categorization, participation.

However, the advanced data constraint can be derived by primitive data semantics. Eg, multiple ISA is equivalent generalization.

For example, generalization can be derivative from multiple ISA data semantics, such as, if a part-time student is a student, and a full-time student is a student, then a part-time and a full time student can be generalized as a student.

Data semantics describe data definitions and data application for users' data requirements, which can be captured in the database conceptual schemas. The following are the data semantics which can be preserved among the legacy conceptual schemas and their equivalent flattened XML schema:

(a) Cardinality: One-to-one, one-to-many and many-to-many relationships set between two classes

A one-to-one relationship between set A and set B is defined as: For all a in A, there exists at most one b in B such that a and b are related, and vice versa. The implementation of one-to-one relationship is similar to one-to-many relationship.

A one-to-many relationship from set A to set B is defined as: for all a in A, there exists one or more b in B such that a and b are related. For all b in B, there exists at most one a in A such that a and b are related.

A many-to-many relationship between set A and set B is defined as: For all a in A, there exists one or more b in B such that a and b are related. Similarly, for all b in B, there exists one or more a in A such that a and b are related.

1:n is constructed by foreign key on "many" side referring to primary key on "one" side in relational schema. It can also be implemented by association attribute of a class object on "one" side points to another class objects on "many" side in another class in object-oriented schema. It can also be implemented by owner record occurrence on "one" side and member record occurrences on "many" side in network schema. It can also be implemented by element occurrence with IDREF on "many" side links with element occurrence with ID on "one" side in XML schema.

As to m:n cardinality, it can be implemented by two 1:n cardinalities with 2 "one" side classes link with the same "many" side class.

(b) Isa relationship between a superclass and a subclass

The relationship A isa B is defined as: A is a special kind of B.

A subclass relation has same primary key as its superclass relation, and refers it as a foreign key in relational schema in isa relationship. It can also be implemented by a subclass inheriting its superclass's OID and attributes in object-oriented schema. It can also be implemented by an owner record that has same key as its member record in network schema via SET linkage. It can also be implemented by an element links one-to-one occurrence with its sub-element in XML schema.

(c) Generalization describes the relationship between one superclass and multiple subclasses.

They are in multiple isa relationships. For example A is a special kind of B, and C is also a special kind of B, then A and C subclasses can be generalized as B superclass. In relational schema, both superclass relation and subclass relation contain the same key, with subclass relations' keys referring to superclass key as foreign key in generalization. In object-oriented schema, multiple subclasses objects contain the same OID as their superclass object in generalization. In network schema, one owner record links with multiple member records through a SET in generalization. In XML, multiple subclass elements and their superclass element are in 1:1 linkage with same key attribute in generalization. Generalization can be implemented by multiple isa relationships with multiple subclasses generalized into one superclass.

Chapter 2 Framework of cross model data semantics preservation

Before data transformation, OUDG performs mapping of major data semantics of cardinality, isa, generalization and aggregation among legacy data models as shown in Table 2:

Table 2 Data semantics implementation in legacy data models and Flattened XML document

Data model\ Data Semantic	Relational	Object-Oriented	Network	XML (in DTD)	Flattened XML(in DTD)
1:n cardinality	Many child relations' foreign key referring to same parent relation's primary key.	A class's association attribute refers to another class's objects' OID(s) as a Stored OID.	An owner record points to many member records via SET linkage.	An element contains many sub-elements.	The IDREF(s) of a sibling element refer to an ID of another sibling element.
m:n cardinality	A relationship relation's composite key refers to 2 other relations' primary keys.	2 class's association attributes refer to same third class OID.	Two owner records point to same member record via 2 SETs linkages.	A sub-element of 1 element links another element by IDREF referring to ID.	A sibling element's 2 IDREF(s) refer to the ID of 2 other sibling elements under root element.
Isa	Subclass relation's primary key is also a foreign key referring to its superclass relation's same primary key.	A subclass inherit OID(s) and attributes of its superclass as its own attributes.	An owner record links to a member record in 1:1 occurrence with same key.	An element occurrence links its sub-element occurrence in 1:1 linkage.	The IDREF of a subclass sibling element data refers to the ID of its superclass sibling element with the same key.
Generalization	2 subclass	Two subclasses	An owner	An element	The IDREF(s)

	relations' primary keys are also foreign keys referring to same superclass relation's primary keys.	inherit OID(s) and attributes of same superclass as their own additional attributes.	record data points to two member records data with same key under 2 SET linkages.	occurrence links with two sub-elements in 1:1 occurrence linkages.	of 2 subclass sibling elements refer to the ID of a superclass sibling element with same key.

Functional dependencies

The preservation of data semantics among legacy databases can be verified by the preservation of their data dependencies as follows:

Definition of FD (functional dependency)

Given a relation R, attribute Y of R is functionally dependent on attribute X of R, i.e., FD: R.X → R.Y, iff each X-value in R has associated with it precisely one Y value in R. Attribute X and Y may be composite.

Definition of ID (inclusion dependency)

ID: Y $\subseteq$ Z states that the set of values appearing in attribute Y must be a subset of the set of values appearing in attribute Z.

Definition of MVD (multi-valued dependency)

Let R be a relation variable, and let A, B and C be the attributes of R. Then B is multi-dependent on A if and only if in every legal value of R, the set of B values matching a given AC pair value depends on the A value, and is independent of the C value.

In general, the presentation of the data semantics of cardinality, isa, generalization and aggregation among legacy databases schemas can be shown in Figure 3. The above data semantics can be preserved in flattened XML documents with sibling elements only, linking with each other via IDREF and ID as shown in Figure 4.

In this thesis, we use data dependencies FD, MVD and ID as a formal method to represent semantic constraints of different data models.

Our approach is to prove that the data dependencies are preserved before and after data transformation through OUDG.

For example, in proving one-to-many cardinality, we can use FD: any “many” side data determine one and only one “one” side data such as each that ID can determined the student’s department (one department many students)

Similarly, in proving isa relationship, we can use ID (Inclusion dependency) such that each part time student‘s is a subset of all students’ id because a part-time student must be also a student. Similarly, in proving many-to-many cardinality we can use MVD (Multi-valued dependency) such as a student can take many courses, and a class can be taken by many students’:

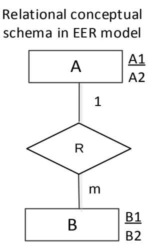
MVD: student ->> class

MVD: class ->> students

FD means functional dependence. (Defined in P.12 of my thesis.) ie, a determinant can determine the value of dependant fields. Eg, a student ID is determinant which can determine the student age as a dependant field.

In this thesis, we use FD to specify the data constraints before and after data conversion (transformation).

If the FD is preserved, before and after database conversion, then we claim that the data semantics are preserved before and after database conversion.



Relational Schema

Relation A (A1, A2)
Relation B (B1, B2, *A1)

Relations

R1

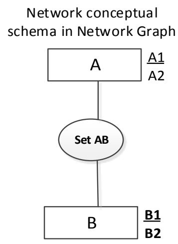
A1	A2
a11	a21
a12	a22

R2

B1	B2	*A1
b11	b21	a11
b21	b22	a12

FD: B → A

(a) one-to-many cardinality



Network Schema

Record Name is A
A1 Character
A2 Character
Record Name is B
B1 Character
B2 Character
Set AB
Owner is A
Member is B

Records

A

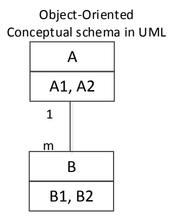
A1	A2
a11	a21
a12	a22

B

B1	B2
b11	b21
b12	b22

FD: B → A

(a) one-to-many cardinality



Object-Oriented Schema

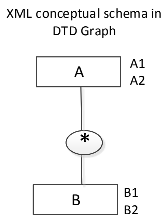
Class A
Attribute A1 Char
Attribute A2 Char
Attribute A_B set (B)
End
Class B
Attribute B1 Char
Attribute B2 Char
Attribute B_A (A)
End

Classes

OID_A	A1	A2	Stored_OID
#1	a11	a21	#3, #4
#2	a12	a22	#3, #4

OID_B	B1	B2	Stotred OID
#3	b11	b21	#1
#4	b12	b22	#2

FD: B → A



XML schema in DTD

<!ELEMENT A(B*)>
<!ATTLIST A1 CDATA #REQUIRED>
<!ATTLIST A2 CDATA #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B1 CDATA #REQUIRED>
<!ATTLIST B2 CDATA #REQUIRED>

XML Docment

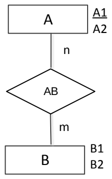
<A A1=" a11" A2=" a12">
<B B1="b11"></B>
<B B1="b12" ></B>
</A>
<A A2=" a21" A2=" a22">
<B B1="b21"></B>
<B B1="b22" ></B>
</A>

FD: B → A

Figure 3a Data semantics preservation in equivalent legacy databases (One-to-many Cardinality)

(b) many-to-many cardinality

Relational conceptual schema in EER model



Relational Schema

Relation A (A1, A2)
Relation B (B1, B2)
Relation AB (*A1, *B1)

Relations

A

A1	A2
a11	a21
a12	a22

B1	B2
b11	b21
b21	b22

AB

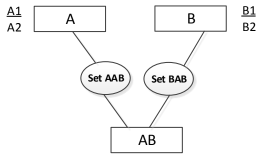
*A1	*B1
a11	b11
a12	b21

MVD: B →→ A

MVD: A →→ B

(b) many-to-many cardinality

Network conceptual schema in Network Graph



Network Schema

Record Name is A
A1 Character
A2 Character
Record Name is B
B1 Character
B2 Character
Record Name is AB
Set AAB
Owner is A
Member is AB
Set BAB
Owner is B
Member is AB

Records

A

A1	A2
a11	a21
a12	a22

B

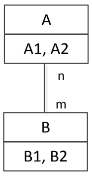
B1	B2
b11	b21
b12	b22

AB

A1	B1
a11	b11
a12	b21

MVD: A →→ B
MVD: B →→ A

Object-Oriented Conceptual schema in UML



Object-Oriented Schema

Class A
 Attribute A1 Char
 Attribute A2 Char
 Attribute A_B set (B)
End
Class B
Attribute B1 Char
Attribute B2 Char
Attribute B_A set (A)
Member is B

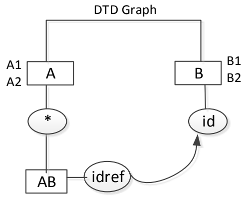
Classes

OIDA	A1	A2	Stored_OID
#1	a11	a21	#3, #4
#2	a12	a22	#3, #4

OIDB	B1	B2	Stotred OID
#3	b11	b21	#1, #2
#4	b12	b22	#1, #2

MVD: A →→ B
MVD: B →→ A

XML conceptual schema in DTD Graph



XML schema in DTD

<!ELEMENT A(AB*)>
<!ATTLIST A1 CDATA #REQUIRED>
<!ATTLIST A2 CDATA #REQUIRED>
<!ELEMENT AB EMPTY>
<!ATTLIST AB_iderf IDREF #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B id ID CDATA #REQUIRED>
<!ATTLIST B1 CDATA #REQUIRED>
<!ATTLIST B2 CDATA #REQUIRED>

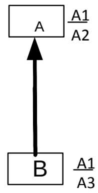
XML Docment

<A A1=” a11", A2=” a21">
<AB idref=” 1"></AB>
</A>
<A A1=” a12", A2=” a22">
<AB idref=” 1"></AB>
</A>
<B B1="b11" B2=” b12" id=1"></B>

MVD: A →→ B
MVD: B →→ A

Figure 3b Data semantics preservation in equivalent legacy databases (Many-to-Many Cardinality)

Relational conceptual schema in EER model



Relational Schema

Relation A(A1 , A2)
Relation B(*A1 , A3)

Relations

A

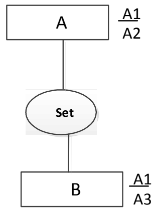
A1	A2
a11	a21
a12	a22

B

*A1	A3
a11	a31
a21	a32

ID : $B.A1 \sqsubseteq A.A1$

Network conceptual schema in Network Graph



Network Schema

Record Name is A
A 1 Character
A 2 Character
Record Name is B
A 1 Character
A 3 Character
Set AB
Owner is A
Member is B

Records

A

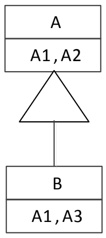
A1	A2
a11	a21
a12	a22

B

A1	A3
a11	a31
a21	a32

ID : $B.A1 \sqsubseteq A.A1$

Object- Oriented Conceptual schema in UML



Object- Oriented Schema

Class A
Attribute A1 Char
Attribute A2 Char
End
Class B subclass of class A
Attribute A1 Char
Attribute A3 Char
End

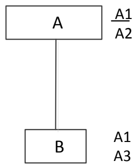
Classes

OID_A	A1	A2
#1	a11	a21
#2	a12	a22

OID_B	A1	A3
#1	a11	a31
#2	a12	a32

ID : $B.OID_A \sqsubseteq A.OID_A$

XML conceptual schema in DTD Graph



XML schema in DTD

<! ELEMENT A(B?)>
<! ATTLIST A1 # REQUIRED>
<! ATTLIST A2 # REQUIRED>
<! ELEMENT B EMPTY>
<! ATTLIST A1 # REQUIRED>
<! ATTLIST A3 # REQUIRED>

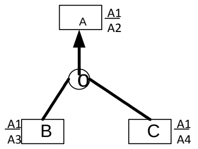
XML document

<A A1=" a11 " A2=" a12"></A>
<B A3="a31 " ></B>
</A>
<A A1=" a11 " A2=" a22"></A>
<B A3="a32 " ></B>
</A>

ID : $B.A1 \sqsubseteq A.A1$

Figure 3c Data semantics preservation in equivalent legacy databases (ISA relationship)

Relational conceptual schema in EER model



Relational Schema

Relation A(A1 , A2)
Relation B(*A1 , A3)
Relation C(*A1 , A4)

Relations

A

A1	A2
a11	a21
a12	a22

B

*A1	A3
a11	a31
a21	a32

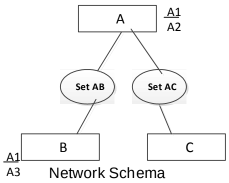
C

*A1	A4
a11	a41
a21	a42

ID : B.A1 $\sqsubseteq$ A.A1

ID : C.A1 $\sqsubseteq$ A.A1

Network conceptual schema in Network Graph



Network Schema
Record Name is A
A 1 Character
A 2 Character
Record Name is B
A 1 Character
A 3 Character
Record Name is C
A 1 Character
A 4 Character
Set AB
Owner is A
Member is B
Set AC
Owner is A
Member is C

Records

A

A1	A2
a11	a21
a12	a22

B

*A1	A3
a11	a31
a21	a32

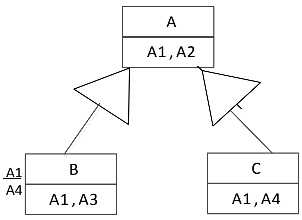
C

A1	A4
a11	a41
a12	a42

ID : B.A1 $\sqsubseteq$ A.A1

ID : C.A1 $\sqsubseteq$ A.A1

Object- Oriented Conceptual schema in UML



Object- Oriented Schema

Class A
Attribute A1 Char
Attribute A2 Char
End
Class B subclass of class A
Attribute A1 Char
Attribute A3 Char
End
Class C subclass of class A
Attribute A1 Char
Attribute A4 Char
End

Classes

A

OID_A	A1	A2
#1	a11	a21
#2	a12	a22

B

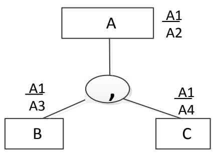
OID_A	A1	A2
#1	a11	a21

C

OID_A	A1	A3
#1	a11	a31

ID : B.OID_A $\sqsubseteq$ A.OID_A
ID : C.OID_A $\sqsubseteq$ A.OID_A

XML conceptual schema in DTD Graph



XML schema in DTD

<! ELEMENT A(B , C)>
<! ATTLIST A1 # REQUIRED>
<! ATTLIST A2 # REQUIRED>
<! ELEMENT B EMPTY>
<! ATTLIST A1 # REQUIRED>
<! ATTLIST A3 # REQUIRED>
<! ELEMENT C EMPTY>
<! ATTLIST A1 # REQUIRED>
<! ATTLIST A4 # REQUIRED>

XML document

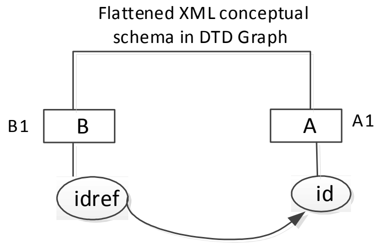
<A A1=" a11 " A2=" a12"></A>
<B A1=" a11 " A3="a31 " ></B>
</A>
<A A1=" a21 " A2=" a22"></A>
<C A1=" a11 " A4="a41 " ></C>
</A>

ID : B.A1 $\sqsubseteq$ A.A1

ID : C.A1 $\sqsubseteq$ A.A1

Figure 3d Data semantics preservation in equivalent legacy databases (Generalization)

(a) one-to- many cardinality



Flattened XML Document schema in DTD

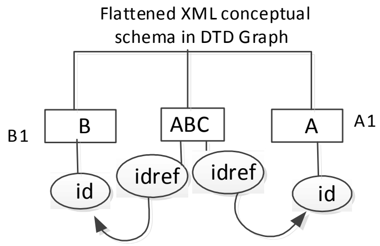
<! ELEMENT ROOT (A , B)>
<! ELEMENT A EMPTY >
<! ATTLIST A id ID # REQUIRED >
<! ATTLIST A A 1 CDATA # REQUIRED >
<! ELEMENT B EMPTY >
<! ATTLIST B idref IDREF # REQUIRED >
<! ATTLIST B B 1 CDATA # REQUIRED >

Flattened XML Document Data

< ROOT >
<A A1="a11 " id="1"></A>
<B B1="b11 " idref=1"></B>
<B B1="b12 " idref=1"></B>
</ ROOT >

FD : B.iderf → A.id

(b) many-to- many cardinality



Flattened XML Document schema in DTD

<! ELEMENT ROOT (A , AB , B)>
<! ELEMENT A EMPTY >
<! ATTLIST A id ID # REQUIRED >
<! ATTLIST A A 1 CDATA # REQUIRED >
<! ELEMENT B EMPTY >
<! ATTLIST B id ID # REQUIRED >
<! ATTLIST B B 1 CDATA # REQUIRED >
<! ELEMENT AB EMPTY >
<! ATTLIST AB idref 1 IDREF # REQUIRED >
<! ATTLIST AB idref 2 IDREF # REQUIRED >
<! ATTLIST AB C CDATA # REQUIRED >

Flattened XML Document Data

< ROOT >
<A A1="a11 " id="1"></A>
<B B1="b11 " id="2"></B>
<A B C="c11 " idref1="1" idref2="2” ></AB>
<A B C="c12 " idref1="2" idref2="1” ></AB>
</ ROOT >

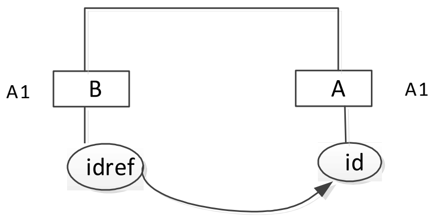
MVD : A.id→→ B.id
MVD : B.id →→ A.id

Figure 4a Data semantics preservation in flattened XML documents

(one-to-many cardinality, many-to-many cardinality)

(c) isa relationship

Flattened XML conceptual schema in DTD Graph



Flattened XML Document schema in DTD

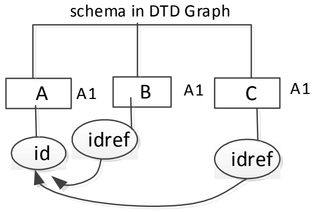
<! ELEMENT ROOT (A , B)>
<! ELEMENT A EMPTY >
<! ATTLIST A id ID # REQUIRED >
<! ATTLIST A A 1 CDATA # REQUIRED >
<! ELEMENT B EMPTY >
<! ATTLIST B idref IDREF # REQUIRED >
<! ATTLIST B A 1 CDATA # REQUIRED >

< ROOT >
< A A1="a11 " id="A1.1"></A>
< B A1="a11 " idref=A1.1"></B>
</ ROOT >

ID : B.idref → A.id

(d) generalization

Flattened XML conceptual schema in DTD Graph



Flattened XML Document schema in DTD

<! ELEMENT ROOT (A,B,C)>
<! ELEMENT A EMPTY >
<! ATTLIST A id ID # REQUIRED >
<! ATTLIST A A1 CDATA # REQUIRED >
<! ELEMENT B EMPTY >
<! ATTLIST B idref IDREF # REQUIRED >
<! ATTLIST B A1 CDATA # REQUIRED >
<! ELEMENT C EMPTY >
<! ATTLIST C idref IDREF # REQUIRED >
<! ATTLIST C A1 CDATA # REQUIRED >

Flattened XML Document Data

< ROOT >
< A A1="a11 " id="1"></A>
< A A1="a12 " id="2"></A>
<B A1="a11 " idref=1"></B>
<C A1="a11 " idref=2"></C>
</ ROOT >

ID : B.idref → A.id
ID : C.idref → A.id

Figure 4b Data semantics preservation in flattened XML documents (ISA, generalization)

Case 1: Mapping relational scheme into Flattened XML schema

Many data semantics in RDB are implemented by primary keys and foreign keys. The corresponding flattened XML tree structure contains id and idref for each element. Therefore, the generic approach is to export all primary keys and foreign keys as id and idref type attributes respectively. Also, all the attributes in Relation A (PK_A, $Attr_1$, … $Attr_n$) in RDB are mapped to all the attributes in element A (PK_A, $Attr_1$, … $Attr_n$, id) in Flattened XML. Similarly all the attributes in Relation B(PK_B, $Attr_1$, … $Attr_n$, *PK_A) in RDB are mapped to all the attributes in element B(PK_B, $Attr_1$, … $Attr_n$, idref) in Flattened XML as shown in Figure 5.

one-to-many cardinality

Given parent relation A and its child relation B, each child relation B foreign key can determine its parent relation primary key. Similarly, each corresponding sibling element B's idref can determine its associated sibling element A's id. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

ISA Relationship

Given a superclass relation A and a subclass relation B, the primary key of subclass B is asubset of its superclass relation A same primary key.
Similarly, given 2 sibling elements A and B, the idref of element B is a subset of the id of element A. These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

many-to-many cardinality

Given a relation A, a relation B and their relationship relation AB, each primary key of relation A can determine many primary keys of relation B through their relationship relation AB in multi-valued dependency.
Similarly, given sibling element A, element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Similarly, the id of element B can determine many id(s) of element A through their associated sibling element AB. These 2 multi-valued dependency are equivalent to each other because they represent the same data semantic of many-to-many cardinality.
(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

Case 2: Mapping Flattened XML schema into relational scheme

Many data semantics in flattened XML tree structure contains id and idref for each element. The corresponding RDB are implemented by artifact primary keys and artifact foreign keys. Therefore, the generic approach is to export all id and idref attributes as artifact primary keys and artifact foreign keys respectively.

Also, all the attributes in element A ($Attr_1$, … $Attr_n$, id) in Flattened XML are mapped to all the Relation A (OID_A, $Attr_1$, … $Attr_n$) in RDB and the OID_A will become the artifact key which is the primary key in relation A. Similarly all the attributes in element B(OID_B, $Attr_1$, … $Attr_n$, idref) in Flattened XML are mapped to all the attributes in Relation B(OID_B, $Attr_1$, … $Attr_{n,}$ * OID_A) in RDB and the idref will become the artifact foreign key in relation B as shown in Figure 5b.

one-to-many cardinality

Given 2 sibling element A and B in flattened XML, sibling element B's idref can determine its associated sibling element A's id. Similarly, given parent relation A and its child relation B in the corresponding RDB, each child relation B artifact foreign key can determine its parent relation artifact primary key. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

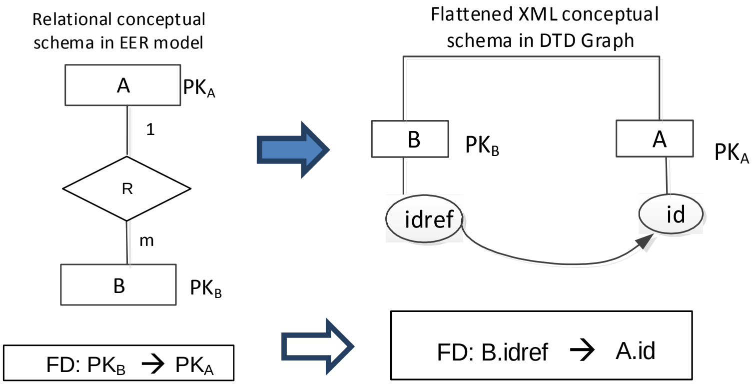
ISA Relationship

Given 2 sibling element A and B in flattened XML, the idref of element B is a subset of the id of element A. Given a superclass relation A and a subclass relation B in the corresponding RDB, the artifact primary key of subclass B is a subset of its superclass relation A artifact primary key.
These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

many-to-many cardinality

Given sibling element A, element B and their associate element AB in flattened XML, the id of element A can determine many id(s) of element B through their associated sibling element AB. Also, the id of element B can determine many id(s) of element A through their associated sibling element AB. Similarly, given a relation A, a relation B and their relationship relation AB, each artifact primary key of relation A can determine many artifact primary key of relation B through their relationship relation AB in multi-valued dependency.
These 2 multi-valued dependency are equivalent to each other because they represent the same data semantic of many-to-many cardinality.
(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

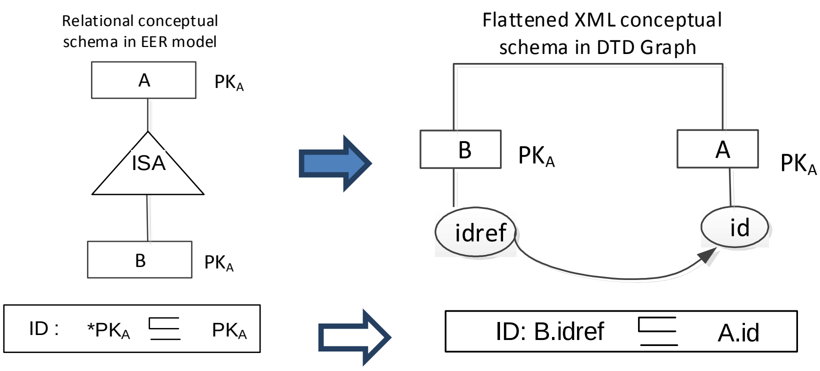
(a) one-to-many cardinality



Mapping: Relation A(PK_A, $Attr_1$, … $Attr_n$) → Element A(PK_A, $Attr_1$, … $Attr_n$, id)

Relation B(PK_B, $Attr_1$, … $Attr_n$, *PK_A) → Element B(PK_B, $Attr_1$, … $Attr_n$, idref)

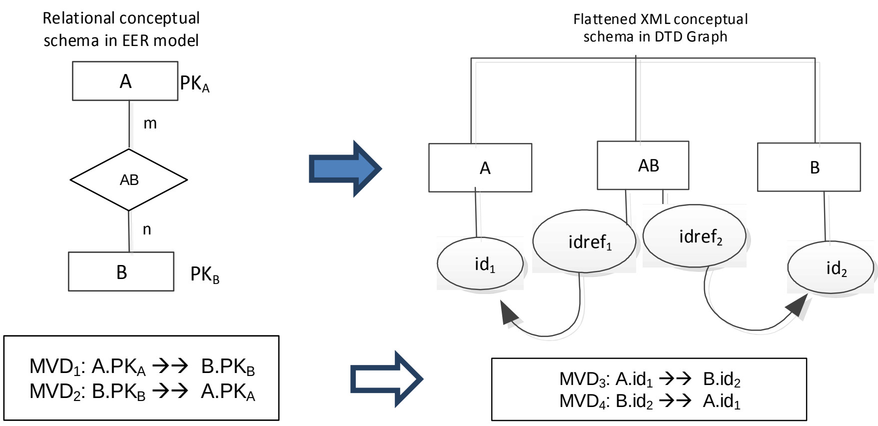
(b) ISA Relation



Mapping: Relation A(PK_A, $Attr_1$, … $Attr_n$) → Element A(PK_A, $Attr_1$, … $Attr_n$, id)

Relation B(*PK_A , PK_B, $Attr_1$, … $Attr_n$) → Element B(PK_A, $Attr_1$, … $Attr_n$, idref)

(c) many-to-many cardinality



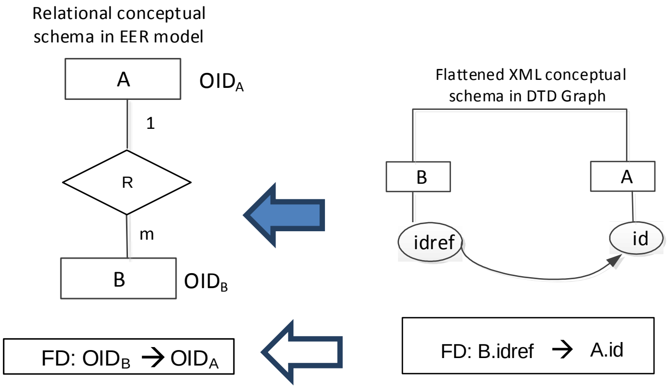
Mapping: Relation A(PK_A, $Attr_1$, … $Attr_n$) → Element A(PK_A, $Attr_1$, … $Attr_n$, id_1)

Relation B(PK_B, $Attr_1$, … $Attr_n$) → Element B(PK_B, $Attr_1$, … $Attr_n$, id_2)

Relation AB(*PK_A, *PK_B) → Element AB($idref_1$, $idref_2$)

Figure 5a Mapping from Relational to Flattened XML schema

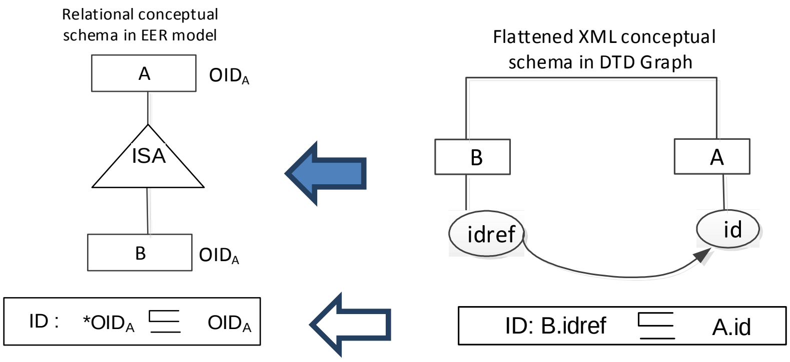
(a) one-to-many cardinality



Mapping: Relation A(OID_A, $Attr_1$, … $Attr_n$) ← Element A($Attr_1$, … $Attr_n$, id)

Relation B(OID_B, $Attr_1$, … $Attr_n$, *OID_A) ← Element B($Attr_1$, … $Attr_n$, idref)

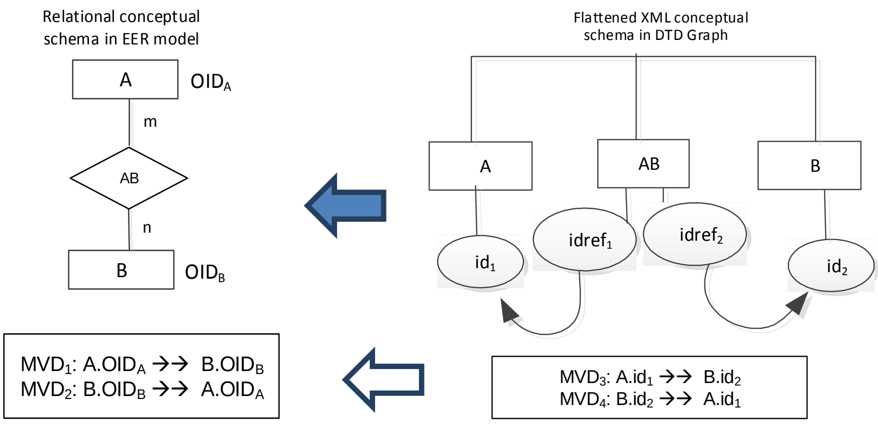
(b) ISA Relation



Mapping: Relation A(OID_A, $Attr_1$, … $Attr_n$) ← Element A($Attr_1$, … $Attr_n$, id)

Relation B(OID_A, PK_B, $Attr_1$, … $Attr_n$,) ← Element B($Attr_1$, … $Attr_n$, idref)

(c) many-to-many cardinality



Mapping: Relation A(OID_A, $Attr_1$, … $Attr_n$) ← Element A($Attr_1$, … $Attr_n$, id_1)

Relation B(OID_B, $Attr_1$, … $Attr_n$) ← Element B($Attr_1$, … $Attr_n$, id_2)

Relation AB(*OID_A, * OID_B) ← Element AB($idref_1$, $idref_2$)

Figure 5b Mapping from Flattened XML schema to Relational schema

Case 3: Mapping Network schema into Flattened XML scheme

Many data semantics in NDB are implemented by set, and each set contains owner and member. The corresponding flattened XML tree structure contains sibling element with an id and another associated sibling element with idref. Therefore, the generic approach is to export NDB owner record and member record to their corresponding sibling elements.

Also, all the attributes in Record A (K_A, $Attr_1$, … $Attr_n$) in NDB are mapped to all the attributes in element A (K_A, $Attr_1$, … $Attr_n$, id) in Flattened XML. Similarly all the attributes in Record B(K_B, $Attr_1$, … $Attr_n$) in NDB are mapped to all the attributes in element B(K_B, $Attr_1$, … $Attr_n$, idref) in Flattened XML as shown in Figure 6a.

one-to-many cardinality

Given owner record A and its member record B, each key attribute of member record B can determine key attribute of owner record. Similarly, each corresponding sibling element B's idref can determine its associated sibling element A's id. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

ISA Relationship

Given an owner record A and a member record B, the key attribute of record B is a subset of the key attribute of its owner record A.
Similarly, given 2 sibling elements A and B, the idref of element B is asubset of the id of element A. These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

many-to-many cardinality

Given an owner record A, an owner record B and their common member AB, each key attribute of record A can determine many key attribute of record B through their common member AB in multi-valued dependency.
Similarly, given sibling element A, element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Similarly, the id of element B can determine many id(s) of element A through their associated sibling element AB. These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality. (Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

Case 4: Mapping Flattened XML scheme into Network schema

Flattened XML is tree structure and contains sibling element with an id and another associated sibling element with idref.

Many data semantics in the corresponding NDB are implemented by set, and each set contains owner and member. Therefore, the generic approach is to export flattened XML element and sibling elements to their corresponding NDB owner record with artifact primary OID and member record.

Also, all the attributes in sibling element A ($Attr_1$, … $Attr_n$, id) in flattened XML are mapped to all the attributes in Record A (OID_A, $Attr_1$, … $Attr_n$) in NDB. Similarly all the attributes in element B($Attr_1$, … $Attr_n$, idref) in flattened XML are mapped to all the attributes in Record B(OID_B, $Attr_1$, … $Attr_n$) in NDB as shown in Figure 6b.

one-to-many cardinality

Given element A and its sibling element B, each corresponding sibling element B's idref can determine its associated sibling element A's id. Similarly, given owner record A and its member record B, each occurrence of member record B can determine an occurrence of owner record. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

ISA Relationship

Given element A and its sibling element B, 2 sibling elements A and B, the idref of element B is a subset of the id of element A. Similarly, given an owner record A and a member record B with same artifact key OID_A, the artifact key OID_A of record B is a subset of the artifact key OID_A of its owner record A. These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

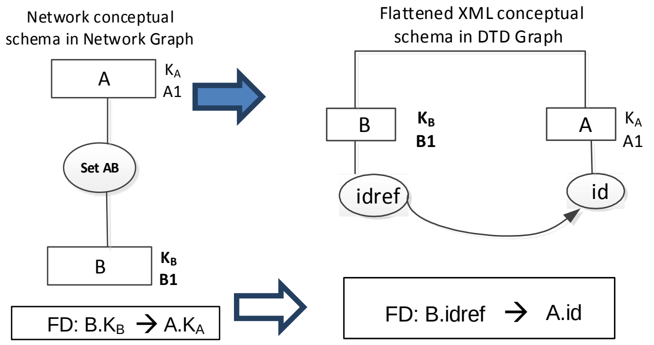
many-to-many cardinality

Given element A, element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Also, the id of element B can determine many id(s) of element A through their associated sibling element AB.

Similarly, given an owner record A with artifact key OID_A, an owner record B with artifact OID_A and their common member AB, each key with artifact OID_A can determine many artifact OID_B through their common member AB in multi-valued dependency and vice versa.

These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality. (Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

(a) one-to-many cardinality

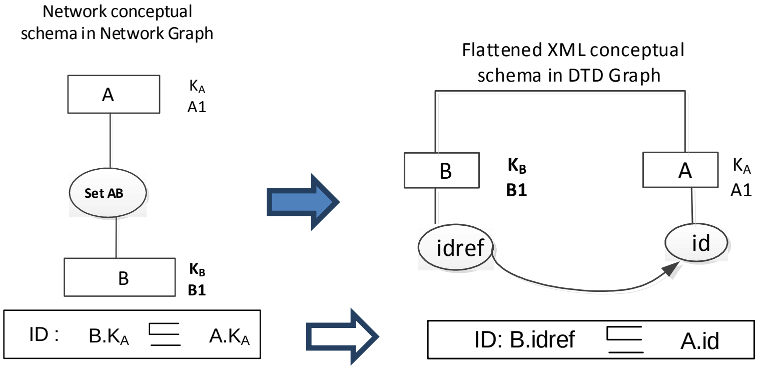


(b)

Mapping: Record A(K_A, Attr A_1, … Attr A_n) ↔ Element A(K_A, Attr A_1, … Attr A_n, id)

Record B(K_B, Attr B_1, … Attr B_n) ↔ Element B(K_B, Attr B_1, … Attr B_n, idref)

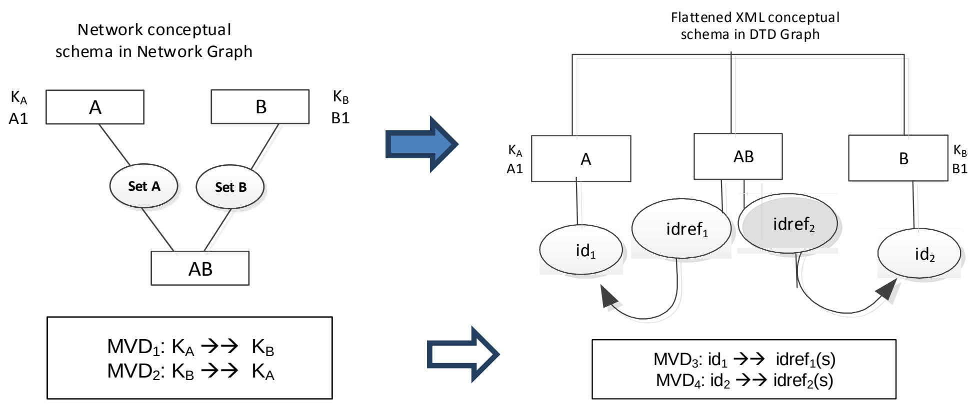
(b) ISA Relation



Mapping: Record A(K_A, Attr A_1, … Attr A_n) ↔ Element A(K_A, Attr A_1, … Attr A_n, id)

Record B(K_A, Attr B_1, … Attr $B_{n,}$) ↔ Element B(K_A, Attr B_1, … Attr B_n, idref)

(c) many-to-many cardinality



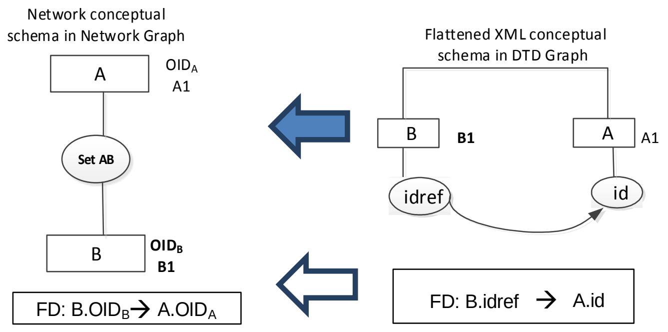
Mapping: Record A(K_A, Attr A_1, … Attr A_n) ↔ Element A(K_A, Attr A_1, … Attr A_n, id_1)

Record B(K_B, Attr B_1, … Attr B_n) ↔ Element B(K_B, Attr B_1, … Attr B_n, id_2)

Record AB(K_A, K_B) ↔ Element AB($idref_1$, $idref_2$)

Figure 6a Mapping from Network to Flattened XML schemas

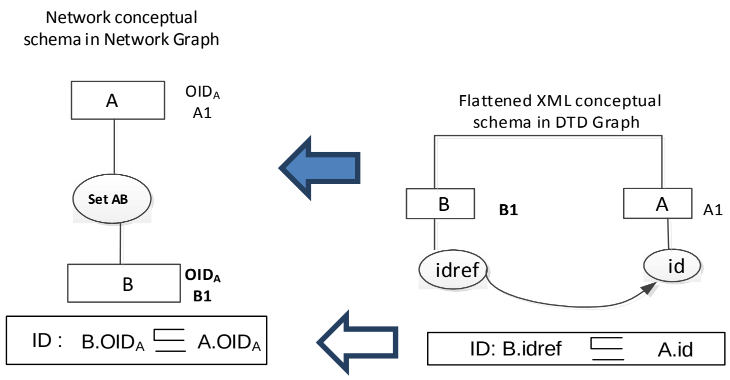
(a) one-to-many cardinality



Mapping: Record A(OID_A, Attr A_1, … Attr A_n) ← Element A(Attr A_1, … Attr A_n, id)

Record B(OID_B, Attr B_1, … Attr B_n) ← Element B(Attr B_1, … Attr B_n, idref)

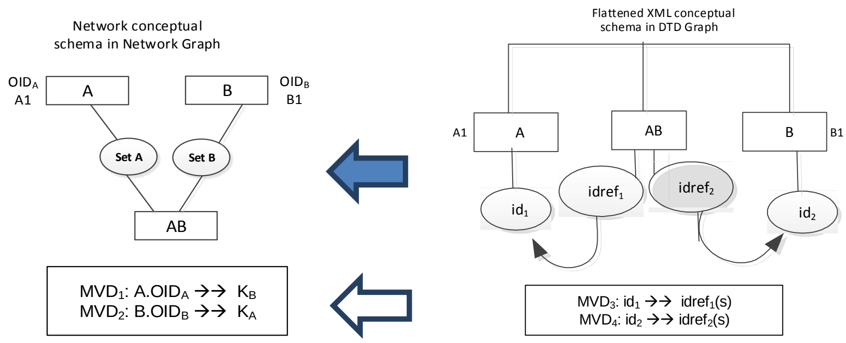
(b) ISA Relation



Mapping: Record A(OID_A, Attr A_1, … Attr A_n) ← Element A(Attr A_1, … Attr A_n, id)

Record B(OID_A, Attr B_1, … Attr $B_{n,}$) ← Element B(Attr B_1, … Attr B_n, idref)

(c) many-to-many cardinality



Mapping: Record A(OID_A, Attr A_1, … Attr A_n) ← Element A(Attr A_1, … Attr A_n, id_1)

Record B(OID_B, Attr B_1, … Attr B_n) ← Element B(Attr B_1, … Attr B_n, id_2)

Record AB($OID_{A,}$ OID_B) ← Element AB($idref_1$, $idref_2$)

Figure 6b Mapping from Flattened XML schema to Network Schema

Case 5: Mapping Object-oriented schema into Flattened XML scheme

Many data semantics in OODB are implemented by OID and stored OID. The corresponding flattened XML tree structure contains sibling element with an id referring to another associated sibling element with idref. Therefore, the generic approach is to export all OIDs and stored OIDs as id and idref type attributes respectively.

Also, all the attributes in class A (OID_A, $Attr_1$, … $Attr_n$) in OODB are mapped to all the attributes in element A ($Attr_1$, … $Attr_n$, id) in Flattened XML. Similarly all the attributes in class B(OID_B, $Attr_1$, … $Attr_n$) in OODB are mapped to all the attributes in element B($Attr_1$, … $Attr_n$, idref) in Flattened XML as shown in Figure 7.

one-to-many cardinality

Given class A and class B, each OID of class B can determine an OID of class A. Similarly, each corresponding sibling element B's idref can determine its associated sibling element A's id. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A occurrence corresponds to many B occurrences.

ISA Relationship

Given an superclass A and a subclass B, the OID of class B is asubset of the same OID of class A. Similarly, given 2 sibling elements A and B, the idref of element B is asubset of the id of element A. These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

many-to-many cardinality

Given a class A, a class B and their common associated class AB, each OID of class A can determine many OID(s) of class B through their common class AB in multi-valued dependency and each OID of associated class B can determine many OID(s) of class A through their common class AB.

Similarly, given sibling element A, element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Similarly, the id of element B can determine many id(s) of element A through their associated sibling element AB. These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality.
(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

Case 6: Mapping Flattened XML scheme into Object-oriented schema

Flattened XML is tree structure and contains sibling element with an id referring to another associated sibling element with idref. Many data semantics in the corresponding OODB are implemented by OID and stored OID. Therefore, the generic approach is to export all id and idref type attributes as OIDs and stored OIDs respectively.

Also, all the attributes in element A ($Attr_1$, … $Attr_n$, id) in Flattened XML are mapped to all the attributes in class A (OID_A, $Attr_1$, … $Attr_n$) in OODB. Similarly all the attributes in element B($Attr_1$, … $Attr_n$, idref) in flattened XML are mapped to all the attributes in class B(OID_B, $Attr_1$, … $Attr_n$) in OODB as shown in Figure 7.

one-to-many cardinality

Given element A and its sibling element B, each element B's idref can determine its associated sibling element A's id. Similarly, class A and class B, each OID of class B can determine an OID of class A. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

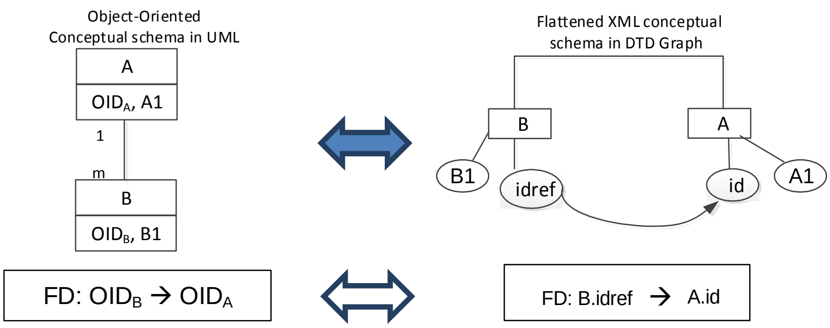
ISA Relationship

Given element A and its sibling element B, the idref of element B is a subset of the id of element A. Similarly, given an superclass A and a subclass B, the OID of class B is a subset of the same OID of class A.
These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each subclass data B must appear in its superclass data A.

many-to-many cardinality

Given element A, its sibling element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Also, the id of element B can determine many id(s) of element A through their associated sibling element AB.
Similarly, given a class A, a class B and their common associated class AB, each OID of class A can determine many OID(s) of class B through their common class AB in multi-valued dependency and each OID of associated class B can determine many OID(s) of class B through their common class AB.
These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality.
(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

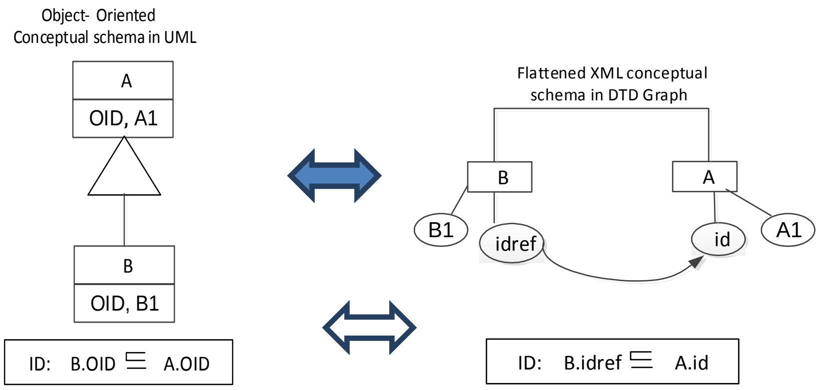
(a) one-to-many cardinality



Mapping: Class A(OID_A, A1) ↔ Element A(Attr A1, id)

Class B(OID_B, B1) ↔ Element B(Attr B1, idref)

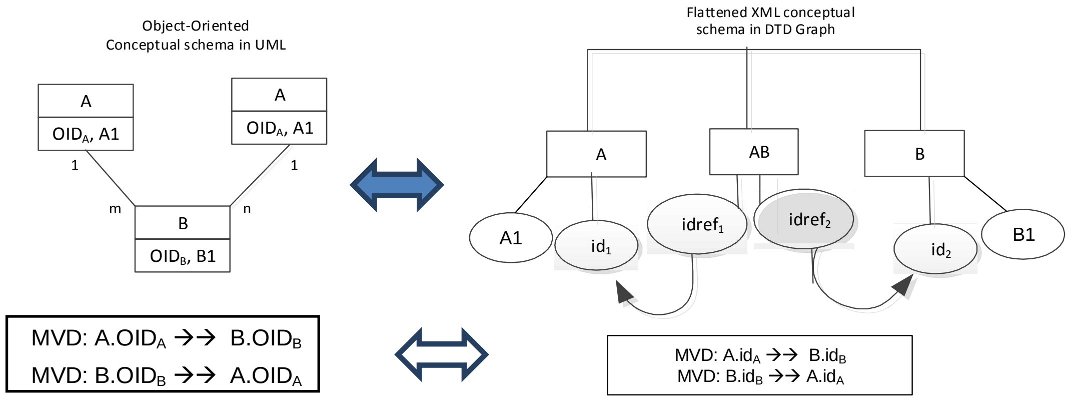
(b) ISA Relation



Mapping: Class A(OID, A1) ↔ Element A(Attr A1, id)

Class B(OID, B1) ↔ Element B(Attr B1, idref)

(c) many-to-many cardinality



Mapping: Class A(OID_A, A1) ↔ Element A(Attr A1, id_A)

Class B(OID_B, B1) ↔ Element B(Attr B1, id_B)

Class AB(OID_{AB}) ↔ Element AB($idref_A$, $idref_B$)

Figure 7 Mapping between OODB schemas and Flattened XML schemas

Case 7: Mapping XML into Flattened XML scheme

Many data semantics in XML are implemented by element and sub-element linkage. The corresponding flattened XML tree structure contains sibling element with an id and another associated sibling element with idref. Therefore, the generic approach is to export the element and sub-element linkage into the id and idref of sibling elements.

Also, all the attributes in element A ($Attr_1$, … $Attr_n$) in XML are mapped to all the attributes in sibling element A ($Attr_1$, … $Attr_n$, id) in Flattened XML. Similarly all the attributes in sub-element B($Attr_1$, … $Attr_n$) in XML are mapped to all the attributes in sibling element B($Attr_1$, … $Attr_n$, idref) in Flattened XML as shown in Figure 8.

one-to-many cardinality

Given element A and its sub-element B, each sub-element of class B can determine of its element A. Similarly, each corresponding sibling element B's idref can determine its associated sibling element A's id. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A corresponds to many B.

ISA Relationship

Given an element A and an sub-element B, the attribute of element B is a subset of the attribute of element A. Similarly, given 2 sibling elements A and B, the idref of element B is a subset of the id of element A.

These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each data in sub-element B must appear in its data in element A.

many-to-many cardinality

Given an element A, an element B and their common sub-element AB, each attribute of element A can determine many attribute of element B through their common sub-element AB in multi-valued dependency.

Similarly, given sibling element A, element B and their associate element AB, the id of element A can determine many id(s) of element B through their associated sibling element AB. Similarly, the id of element B can determine many id(s) of element A through their associated sibling element AB. These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality.

(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

Case 8: Mapping Flattened XML into XML scheme

Flattened XML is tree structure and contains sibling element with an id referring to another associated sibling element with idref. Many data semantics in the corresponding XML are implemented by id and stored idref. Therefore, the generic approach is to export all id and idref of sibling elements to the element and sub-element linkage respectively.

Also, all the attributes in sibling element A ($Attr_1$, … $Attr_n$, id) in flattened XML are mapped to all the attributes in element A ($Attr_1$, … $Attr_n$, id) in XML. Similarly all the attributes in sibling element B($Attr_1$, … $Attr_n$, idref) in flattened XML are mapped to all the attributes in sub-element B($Attr_1$, … $Attr_n$) in XML as shown in Figure 8.

one-to-many cardinality

Given sibling element A and its sibling element B in flattened XML, each corresponding sibling element B's idref can determine its associated sibling element A's id. Similarly, given element A and its sub-element B in XML, each sub-element of class B can determine its element of class A. Both functional dependencies are equivalent because they represent the same data semantic of one-to-many such that each A class occurrence corresponds to many B class occurencies.

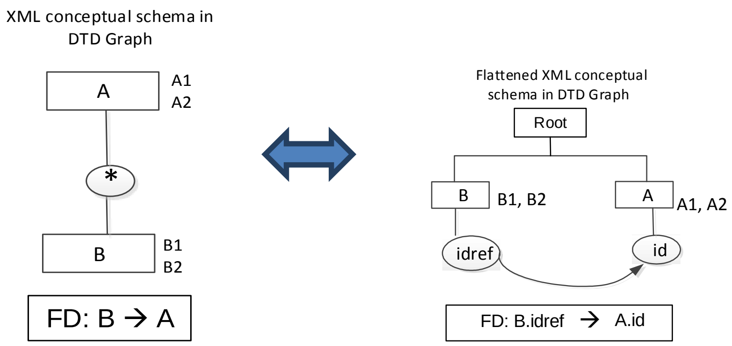
ISA Relationship

Given 2 sibling elements A and B in flattened XML, the idref of element B is asubset of the id of element A. Similarly, given an element A and a sub-element B in XML, the attribute of element B is a subset of the attribute of element A.
These 2 inclusion dependencies are equivalent to each other, because they represent the same data semantic ISA such that each data in sub-element B must appear in its data in element A.

many-to-many cardinality

Given sibling element A, element B and their associate element AB in flattened XML, the id of element A can determine many id(s) of element B through their associated sibling element AB. Also, the id of element B can determine many id(s) of element A through their associated sibling element AB. Similarly, given an element A, an element B and their common sibling element AB in XML, each attribute of element A can determine many attribute of element B through their common sibling AB in multi-valued dependency.
These 2 multi-valued dependencies are equivalent to each other because they represent the same data semantic of many-to-many cardinality.
(Note: 2 one-to-many cardinalities is equivalent to one many-to-many cardinality).

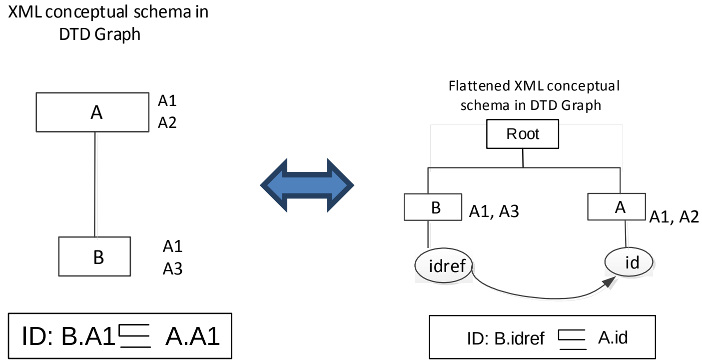
(a) one-to-many cardinality



Mapping: Element A(A1, A2, …) ↔ Element A(A1, A2, …, id)

Element B(B1, B2,…) ↔ Element B(B1, B2,… idref)

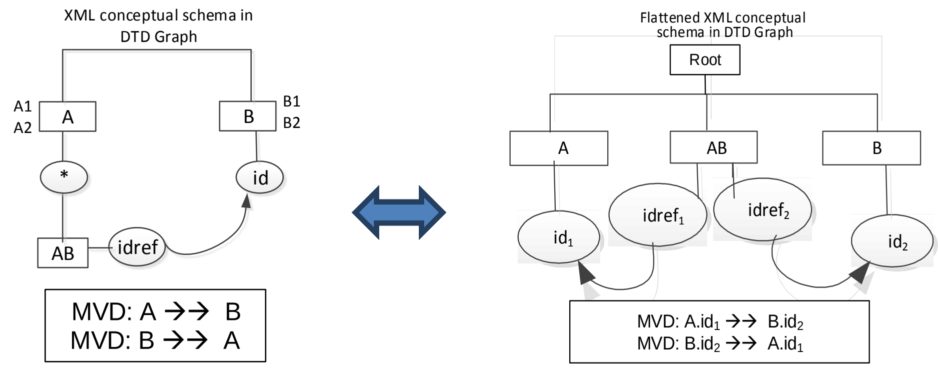
(b) ISA Relation



Mapping: Element A(A1, A2, …) ↔ Element A(A1, A2, …, id)

Element B(B1, B2,…) ↔ Element B(B1, B2,… idref)

(c) many-to-many cardinality



Mapping: Element A(A1, A2, …) ↔ Element A(A1, A2, …, id)

Element B(B1, B2,…) ↔ Element B(B1, B2,… idref)

Element AB(idref) ↔ Element AB($idref_1$, $idref_2$)

Figure 8 Mapping between XML and Flattened XML schemas

Chapter 3 Related Works

On data transformation

Lum et al (1976) showed how to construct data conversion languages SDDL and TDL to extract and restrict data from source legacy database into target legacy database. They defined two languages in this paper: (1) a language to describe the data structures, and (2) a language to specify the mapping between source and target data.

Fong, J and Bloor,C. (1994) described mapping navigational semantics of the network schema into a relational schema before converting data from network database to relational database. The methodology preserves the constraints of the network database by mapping the equivalent data dependencies of a loop-free network schema to a relational schema. The conversion process translates the existence and navigational semantics of the network database into a relational database without loss of information.

Fong, J (1997) suggested a methodology of the data conversion between object-oriented database objects and Relational database. Data conversion involves unloading tuples of relations into sequential files and reloading them into object-oriented classes files. He also presented a methodology of transformation by using SQL Insert statements in this paper.

Fong, J and Shiu, H. (2012) proposed a new interpretive approach to exporting data in a relational database to an XML document. They designed a Semantic Export Markup Language as a language for data conversion process in the paper.

Fong et al.(2003b) presented a semantic metadata to preserve database constraints when processing the database conversion. This paper also applied logical level approach for data materialization between relational database and object-oriented database using sequential file as medium.

I get the idea from Shoshani, A.(1975) about the logical level approach data conversion. From Fong et al.(2003b), I try to think a semantic metadata to preserve database constraints when processing the database conversion.

On Heterogeneous database

Given huge investment for a company put on heterogeneous databases, it is difficult for the company to convert them into homogeneous databases for new applications. Therefore, researchers have come up with a solution of universal databases that can be accessed as homogeneous databases by the user (Fong, J. and Huang, S.M., 1999). For instance, we can provide a relational interface to non-relational database such as Hierarchical, Network, Object-Oriented and XML (Fong, J., 1996).

Hsiao, D.K. and Kamel, M.N.(1989) offered a solution of multiple-models-and-languages-to-multiple-models-and–languages mapping to access heterogeneous databases. This paper talked about mainframe-based heterogeneous DBMS involved relational database and hierarchical database.

Based on Fong, J (1991), it is good to understand how to translate heterogeneous database schemas into Extended Entity Relationship Model as a conceptual schema for information retrieval.

On Universal database

Fong et al. (2003a) applied universal database system to access universal data warehousing for the integration of both relational databases (RDB) and object-oriented databases (OODB) with star schema and Online Analytical Processing (OLAP) functions. A star schema is derived from user requirements based on the integrated schema, catalogued in the metadata, which stores the schema of RDB) and OODB. OLAP is the object oriented view of the data warehouse through method call derived from the integrated schema.

Silverston, L. and Graziano, K. (2008) used a universal data model in a diagram to design the conceptual schema of different legacy data models of any legacy database. Common data model in a convenient format is needed in their design. Also, all data models are normalized in this paper.

Because Fong, and Huang (1999) proposed using a frame model metadata to unite different data models of various databases as a universal database, I get the idea of UDB in concrete.

On schema translation

Navathe et. al. (1998) offered a reverse engineering solution to extract the data semantics from the relationships of the primary keys and foreign keys in relational schema into an Extended Entity Relationship Model. They suggested to translate a logical hierarchical schema or a logical network schema into a conceptual schema based on the extended entity relationship (EER) model. The EER model is then translated into a logical relational schema.

Because Funderburk et al. (2002) proposed that a bridge is needed to develop XML based applications in relational database technology, I think that flattened XML is a good idea to be the middleware of UDB.

On Cloud Database or Cloud Computing

Harris, D. (2012) defined cloud database as databases in virtual machines. In this article, the writer listed out some cloud database company and software which provide SQL services or NoSQL services. Also, some virtualization and network-based architecture of cloud database were described.

Wang, S.P., Ledley, R.S. (2013) defined virtual machine(VM) is the practical implementation of virtualization. This book gives the idea how to configure and partition multiple independent "virtual" servers into one physical servers. The advantages of VM are: (1) save lots of hardware resources when conducting the large scale prototype of Universal Database system. (2) act as cloud computing service to provide software as a service(SAS), platform as a service(PAS), and infrastructure as a service(IAS). (3) with its flexibility of computing power.

Rhoton, J. and Haukioja. R., (2013) defined cloud computing is a technology of network computing where an application can run on several connected servers. The book provides a concept how to implement our Universal Database in a cloud computing platform and distributed different database model in different Virtual machine(VM) server. This book is really helpful in the performance analysis of our research.

On Relational Interface

Fong, J (1996) applied a relational API (application program interface) to access hierarchical and network databases by SQL, schema translation pre-processing and online transaction translation. He proposed the relational API should be developed by embedded SQL programs and providing a relational-to-hierarchical interface.

Gilmore, W.J. (2000) defined entity and its features in databases. The paper also introduced relationship, included one-to-one, one-to-many, many-to-many. Moreover, The author applied three normal forms by using MYSQL as example.

Janssen, C. (2014) defined a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. It describes how an ERD represent the entity framework infrastructure. Also, it explains why ERD is crucial to creating a good database design.

Fong, J (2006) published this book for describing database conversion techniques, reverse engineering and forward engineering, and re-engineering methodology for information systems by taking a practical approach. This book offers a systematic software engineering approach for reusing existing database systems built with "old" technology. Many examples, illustrations and case studies are used, making the methodology easy to follow.

Fong, J (1992) describes a method to translate from a non-relational to a relational schema. The methodology uses reverse engineering to extract entities and relationships into an extended entity-relationship model from the semantics of a hierarchical or network schema. The logical equivalence of the translated relational schema with the hierarchical or network schema is validated by verifying the preservation of the functional and inclusion dependencies in the schemas. A reverse translation to recover the original hierarchical or network schema is also used to validate the translation.

Chen, P. (1976) introduced Entity–Relationship (ER) modeling for unification of different views of data: the network model, the relational model and the entity set model. In the paper, it discussed how to handle semantics of data and use n-ary relationships when everything is treated as an entity.

CODD, E. F. (1970) suggested the concept of a universal data sublanguage based on n-ary relations. He also discussed that sublanguage in certain operations on relations which could be applied to the problems of redundancy and consistency in the user's model. This was a old paper and the idea is limited by hierarchical and relational database.

Fong, J (2004) presented a methodology, XTOPO to transmit relational database on the Internet using XML document as medium. XTOPO divides an XML document hierarchical structure into four different topologies: single sub-element (element, sub-element), multiple sub-elements (element, multiple sub-elements), group (element, group of sub-elements) and referral element (element, element) and capture their semantics into classification tables as a knowledge-based repository. The view of a sender company's information in a relational database is mapped into four topological XML documents according to their data semantics constraints.

Fong, J (2001) suggested Converting Relational Database into XML Documents with DOM. Fong said the schema translation must be done before data conversion. Fong suggested that relational databases should be denormalized by joining the normalized relations into tables according to their data dependencies constraints. Finally, the joined tables are mapped into DOMs, which are then integrated into XML document trees. In this paper, the writer proposed a method to convert the relational database into XML. The data dependencies constraints in the relational databases are represented in the relationship between Element and Sub-element in the XML documents.

Fong, J and San Kuen Cheung (2005) translated relational schema into XML schema definition with data semantic preservation and XSD graph. This paper is related to schema level. Data semantics of participation, cardinality, generalization, aggregation, categorization, N-ary and U-ary relationship are preserved in the translated XML schema definition.

Guardalben, G. (2004). proposed a method of XML-to-RDB mapping to integrate XML and relational data. But semantic constraints are <u>not mentioned</u> in this paper.

Kanagaraj, S. and Sunitha, A. (2012) proposed a method of converting relational database into Xml document. But semantic constraints present in the source databases are <u>*not included*</u> in the conversion.

Lee, D., Mani, M and Chu, W. W. (2002) presented three semantics-based schema transformation algorithms. They used Inclusion Dependencies and Tuple-Generating Dependencies (TGDs), but schema level only.

Lee, D., Mani, M and Chu, W. W. (2012) proposed a schema conversion methods between XML and Relational Models. They used of Inclusion Dependencies, but schema level only.

On DBMS

Raima (2014) is a network model database. It provides all basic features of network database, included owner/member records, set, etc. The DMBS also provides software utilities so that we can conduct the prototype and performance analysis of UDB in window platform without too much programming.

Oracle (2014) is a Relational Database DBMS. Oracle provides lots of function for administrator, including memory control, SQL command for administrator, etc. Its performance is very good in window platform. The Oracle DBMS can store and execute stored procedures and functions within itself by PLSQL. We used Oracle as our prototype software.

eXist (2014) is an XML DBMS and high-performance native XML database engine. It provides a graphical user interface for execute the Xquery command. It is very user friendly to conduct the XML performance analysis in window platform.

On Flattened XML document

Referenced from Fong et al.(2009), who converted an XML document into Relational database by transforming XML document into flattened XML document with relational table structure by Extensible Stylesheet Language Transformation, I can deduce that I can cover more data models in my research. They are NDB, RDB, XML, OODB and flattened XML.

My thesis is talking about UDB(Universal Database), so it must have (1) Schema translation between legacy DBs, then (2) Data transformation between legacy DBs, and (3) Universal DB of legacy DBs interoperability.

In my thesis, even though it is talking about data transformation methodology, the application is UDB. Because our UDB can access any database, ie, any database can be related to each other such that, everyone can access it.

On Homogeneous database

Based on Sellis, T., Lin, C.C. and Raschid, L. (1993), who presented a solution to decompose and store the condition elements in the antecedents of rules such as those used in production rule-based systems in homogeneous databases environment using relational data model, I know the difference of UDB application between homogenous and heterogeneous database.

<u>On XML export and import</u>

XML export and import (2014) told about that occasionally migrate data from one instance to another, one can export the XML data from one instance and import it to another.

http://wiki.servicenow.com/index.php?title=Exporting_and_Importing_XML_Files

<u>XML as web service</u>

XML as web service (2014) described that XML provides the web service, ie, output the data as XML format file, then the browser can input that XML file and display as webpage.

XML as web service (2014), http://www.altova.com/downloadxmltools.html

Compared to the above references and the other papers, this thesis has 3 uniqueness:

1. Cover more data model

   All other database research only involve 2 or 3 data models in the universal database. This thesis involves 5 data models in our research. They are NDB, RDB, XML, OODB and flattened XML.

2. Use cloud platform

   None of researcher uses cloud platform to conduct the search involve universal database. All of our database and the UDB prototype are developed in cloud platform.

3. New idea: use flattened XML as middleware

   This thesis offers an architecture of Open Universal Database Gateway (OUDG) to transform legacy database data into Flattened XML documents, and to transform

Flattened XML document back into any other legacy database. We use this file format as middleware for data conversion.

This thesis extends the work of universal database into an "open" universal database gateway. The limitation of a universal database gateway is restricted to a particular DBMS. For example, the user can access all legacy databases by using SQL on the non-relational database even though their DBMS(s) may not be all relational. Nevertheless, the restriction of such solution is that the user must depend on a particular relational database language to access heterogeneous databases.

This thesis offers an open database in flattened XML document. The "openness" of universal database gateway is "flexible DBMS" independent while the universal database is "fixed DBMS" dependent. The OUDG provides users the flexibility of choosing any DBMS for legacy databases.

Similarly, the OUDG differs from ODBC because ODBC requires programming solution to access various relational databases while OUDG transforms all legacy databases into each other for e-commerce through a database gateway middleware. Furthermore, OUDG can reengineer the obsolete Hierarchical or Network database into XML documents on the Internet, which is the trend of IT technology.

Chapter 4 Methodology of open universal database gateway (OUDG)

This thesis offers OUDG as a database middleware to access legacy databases via flattened XML documents as follows:

Source Legacy databases→Flattened XML documents →Target Legacy databases

Hypothesis: Since OUDG in feasible, legacy DB and flattened XML are interchangeable, and since flattened XML can be accessed on the internet , therefore, any legacy DB can be accessed as flattened XML representation on the Internet. Therefore OUDG can become an end user computing tool to connect most legacy DB, such as Internet can connect most computers.

Our contribution is, based on our theory, OUDG could act as a database middleware to access 5 legacy databases via flattened XML documents at the same time. The 5 legacy databases are relational, hierarchical, network, object-oriented and XML database. While research from the others only allowed 2 legacy databases transformation, e.g., relational-to-XML, relational-to-hierarchical, etc, there is no such contribution among 5 legacy databases interchangeable to each other in the same paper

Our theory: There are different legacy databases with different data models. They need to be interchangeable without loss of information. Our method is using flattened XML as the middleware to interchange among 5 legacy databases, including relational, hierarchical, network, object-oriented and XML database.

Limitation: The theory is only limited to 5 legacy database model, relational, hierarchical, network, object-oriented and XML database and 3 data semantic(cardinality, ISA and generalization).

We select four data models to represent legacy databases for illustration: Network model for network database in network structure, relational model for relational database in table structure, XML model for XML database in tree structure, and Object-Oriented model for Object-Oriented database in class structure. In order to develop OUDG, we apply two steps methodology, transforming user's legacy database into flattened XML documents in Step 1, and transform the flattened XML document into a target's legacy database in step 2.

The methodology procedure for conversion between legacy databases and the flattened XML documents and vice versa is shown in Figure 2 with two basic steps:

Main algorithm:
Begin
    If   legacy database conceptual schema does not exist
    Then Reverse engineering legacy logical schema into legacy database conceptual schema;   /* pre-process */
     Transform source's legacy database into flattened XML document;   /* step 1*/
     Transform flattened XML document into a target legacy database; /* step 2 */
End;

Pre-process: Reverse engineer legacy database logical schemas into their conceptual schemas
As shown in Table 2, for the structural constraints of each legacy database, we can recover their data semantics accordingly.

For example, to reverse relational schema into an Extended Entity Relationship model, a classification table can be used to define the relationship between keys and attributes in all relations, and data semantics can be recovered accordingly. A 1:n cardinality in relational schema can be recovered from a foreign key(FKA) between two relations in classification table, with foreign key relation on "many" side and referred primary key relation on "one" side (Fong, J., 1992).

Similarly, we can reverse engineer object-oriented schema into UML by recovering 1:n association between two associated objects with a Stored OID on "many" side in a class referring to an OID on "one" side in another associated class in OODB. We can also reverse Network schema into Network database conceptual schema Network Graph by recovering owner record on "one" side and member records on "many" side. Similarly, we can reverse engineer XML schema DTD into XML conceptual schema DTD Graph because their logical and conceptual schemas are identical except the latter is in graph format.

Define a Root element.
We recover legacy database conceptual schema in a diagram. The selection of root element of flattened XML schema represents the view of users data requirement on each legacy database. To select a root element, its relevant information must be put into an flattened XML schema.

Relevance is concerned with entities that are related to an entity selected by the user for processing. The relevant classes include the selected entity and all its related entities that are navigable. Navigability specifies whether traversal from an entity to its related entity is possible.

For example, given an entity relationship model as shown in Figure 9. We can select entity E as root element for flattened XML schema. As a result, the mapped flattened XML schema is extracted from the EER model as shown in Figure 9. On the other hand, we can also select an artifact root element which include all entities in the ER model for data transformation as shown in the case study.

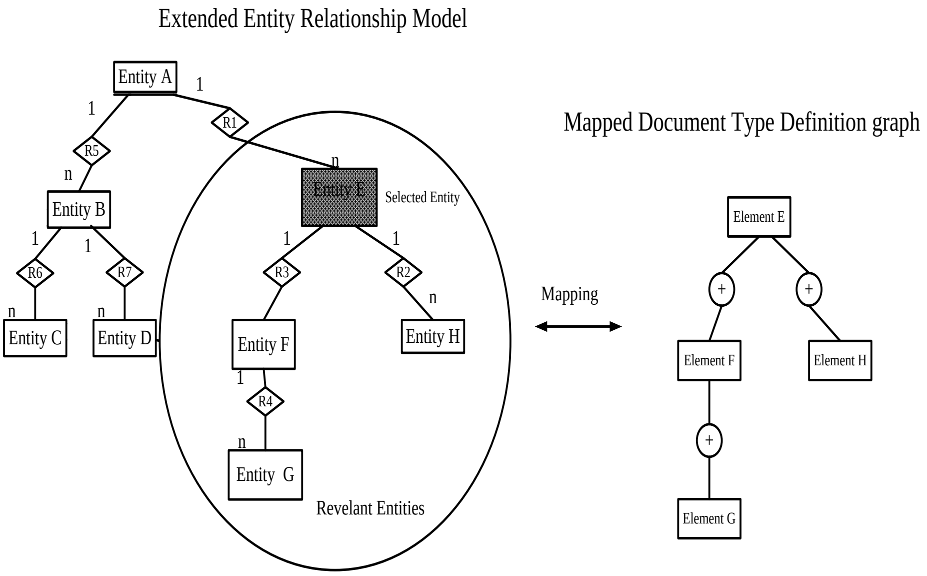


Figure 9 Selected “Root element” and Relevant Entities are mapped into a DTD graph

Step 1: Transform user's source legacy databases into flattened XML documents

Firstly, in pre-process we capture the data semantics of a legacy database into its conceptual schema, for example, EER model for relational database, UML for object-oriented database, Network graph for network database and DTD graph for XML database. These data semantics can be mapped into the flattened XML document schema by storing each data semantic in XML DTD (data type definition) schema. The data semantics include one-to-one, one-to-many, many-to-many cardinalities and relationship, generalizations, and which can be mapped among the flattened XML document and the legacy databases.

Secondly, , we perform data transformation from legacy database into flattened XML document using logical level approach (Shoshani, A.,1975, Lum, V.Y., 1976, Fong, J., 2006).

Case 1: Transform relational databases into flattened XML documents

Firstly, we perform the preprocess of mapping relational schema into flattened XML schema. Secondly, we perform their correspondent data transformation. The Input is a relational database and the output is an flattened XML document. The system will read relational table according to the legacy relational schema. In one-to-many data semantic, it will post parent and child relations into flattened sibling XML elements linked with id and idref. In many-to-many data semantic, it will post 2 relations and their relationship relation into flattened XML sibling elements linked with idref(s) and id(s). In isa data semantic, it will post superclass and subclass relations into table structured XML sibling elements linked with id and idref with the same key. In generalization data semantic, it will post superclass relation and subclasses relations into XML sibling elements linked with id(s) and idref(s) with the same key in sibling elements.

```
Preprocess algorithm: Map relational schema into flattened XML schema:
Begin
      Select a root element for flattened XML schema;
      If relation B foreign key refers to relation A primary key
Then begin
          /*Map relations A and B of 1:n cardinality into sibling elements A and B
            of 1:n cardinality; where A is one and B is many */
            Map relation A into sibling element A with ID;
```

```
6   Map relation B into sibling element B with IDREF refer to the above
ID;
 7        end;
 8        If relation B has a primary key which is also a foreign key refers to
          relation A primary key
 9   Then begin
     /*Map relation A isa relation B into sibling element A isa sibling element B;
       where A is subclass and B is superclass */
10          Map relation A into sibling element A with ID value of relation key
     value;
11          Map relation B into sibling element B with IDREF value of the same
      relation key value;
 12        end;
 13    If (relation A and relation B is in 1:n cardinality) And (relation C and
 relation B is in 1:n)
 14    Then relation A and relation C are in m:n cardinality;
 15  If (relation A isa relation B) and (relation C isa relation B)
 16  Then relation A and relation C are generalized into relation B;
     /* A and C are subclasses, and B is their superclass */
 17 End;
```

Process algorithm: Transform relational database to flattened XML document

Input: Relational database

Output: Flattened XML document

```
begin
    Create a raw XML document with an arbitrary root element r
     For each table do
     begin
      For each record rec do
      begin
      Create an XML element e named as its table name
         If table of the record defines a primary key pk
         then begin
      Create an ID attribute id named table-name.column-name with value
                    table-name.primary-key-value;
                    Add the above id as attribute of e;
              end;
             For each foreign key fk of the table do
             begin
             Create an IDREF attribute idref named
             primary-table-name.foreign-key-column-name with value
                    primary-table-name.foreign-key-value;
                    Add the above idref as attribute of e;
             end
      end
      Add e as child element of r;
 end
```

Case 2: Transform XML databases into flattened XML documents

Firstly, we perform the preprocess of mapping XML schema into flattened XML schema. Secondly, we perform their correspondent data transformation. The Input is an XML document and the output is a flattened XML document. The system will read XML document according to the XML schema. In one-to-many data semantic, it will post element and sub-element into flattened XML document sibling elements linked with id and idref. In many-to-many data semantic, it will post 3 elements linked with id(s) and idref(s) into flattened XML document sibling elements linked with id(s) and idref(s). In isa data semantic, it will post superclass and subclass elements into flattened XML document sibling elements linked with id and idref with the same key. In generalization data semantic, it will post element and sub-elements into flattened XML document sibling elements linked with id(s) and idref(s) with the same key in DTD "," separator in the flattened XML schema.

Preprocess algorithm: Map XML schema into flattened XML schema:

```
Begin
  If element A and its sub-element B have same key attribute a1 in XML schema
Then begin
    Map element A isa element B into sibling elements A isa B;
      /*A is subclass and B is superclass */
    Map element A with attributes into sibling element A with same attributes and
    an ID value into flattened XML schema;
    Map element B with attributes into sibling element B with same attributes and
    IDREF referring above ID value into XML schema;
      end;
  If (sub-element B under element A) or (element B has an IDREF referring to
      element A ID value)
    Then begin
      Map sibling elements A and B in 1:n cardinality into elements A and B in
      1:n cardinality;/*A is subclass & B is superclass */
      Map element A into sibling element A wth an ID value in flattened XML
      schema;
      Map element B into sibling element B with an IDREF referring to the above
      ID value in flattened XML schema;
      End;
If (element A and element B is in 1:n cardinality) And (element C and element B
```

```
is in 1:n)
Then element A and element C are in m:n cardinality in XML schema;
   If (element A isa element B) and (element C isa element B)
   Then element A and element C are generalized into element B;
          /* A,C are subclasses to superclass B */
End;
```

Process algorithm: Transform an XML document to a flattened XML document

Input: an XML document

Output: a flattened XML document

```
Begin
      Read XML document elements instances by using depth first search;
      While not at end of instances do
      begin
            For each element obtained
            Add a sibling element with an ID attribute id with value
            “entity:sequence_number”;
            For each sub-element obtained
            Add a sibling element with an IDREF attribute idref with value
            “parent_element_name:seqeuence number of its element;

      end;
end;
```

Case 3: Transform Object Oriented database into flattened XML document

Firstly, we perform the preprocess of mapping object-oriented schema into flattened XML schema. Secondly, we perform their correspondent data conversion. The Input is an OODB and the output is a flattened XML document. The system will read OODB according to OODB schema. In one-to-many data semantic, it will post object and set of associated objects into XML sibling elements linked with id and idref. In many-to-many data semantic, it will post 2 sets of associated objects with a common object into XML sibling elements linked with id(s) and idref(s). In isa data semantic, it will post superclass and subclass objects with same OID into XML sibling elements linked with id and idref with the same key. In generalization data semantic, it will post superclass and multiple subclasses objects into sibling elements linked with id(s) and idref(s) with the same key in DTD "," separator in the flattened XML document schema.

```
Preprocess algorithm: Map object-oriented schema into flattened XML schema:
1 Begin
2      If B is subclass of class A
3      Then begin
       /* Map classes A and class B into sibling element A and B
         where B is subclass and A is superclass */
4        Map class A with OID into sibling element A with ID value same as
           OID;
5          Map class B with same OID as above into sibling element B with
           IDREF referring to the above ID value;
6        end;
7   If class A has association attribute referring to class B's multiple objects
8     Then begin
   /*Map Classes A and B in 1:n cardinality into sibling elements B and C in
           1:n cardinality where A is one and B is many */
9   Map class A with OID into sibling element A with ID value same as OID;
10 Map class B with stored OID into sibling element B with IDREF referring
```

```
            to the above ID value;
        End;
    If (sibling element A and sibling element B are in 1:n cardinality) And
            (sibling element C and sibling element B is in 1:n)
Then sibling element A and sibling element C are in m:n cardinality;
If (sibling element A isa sibling element B) and
(sibling element C isa sibling element B)
Then sibling element A and sibling element C are generalized into sibling
element B; /*A,C are subclasses to superclass B*/
End;
```

Process algorithm of transforming OODB to flattened XML documents

Input: An OODB instance

Output: A flattened XML document

```
Begin
        Create a flattened XML document with a root element
        For each class c in OODB do
        Begin
              For each object obj in class c do
              Begin
                Derive an OID for class c for object obj;
                Create a sibling XML element for object obj as a sibling element
                  of flattened XML document with OID as ID type attribute;
         End
            For each association attribute of obj do
                 Begin
                          For each referred obj with stored OID do
                Begin
                         Locate the corresponding sibling XML element e in flattened
                         XML document:
                                  Create an IDREF attribute for element e:
                End
```

17 End

18 For each association attribute of *obj* do

19 Begin

20 Map the superclass object into sibling element with an ID and OID as key value;

21 Map the subclass object with another sibling element with an IDREF referring to the above ID and OID as key value

22 End

23 End

24 End

Case 4: Transform Network databases into flattened XML documents

Firstly, we perform the preprocess of mapping network schema into flattened XML schema. Secondly, we perform their correspondent data conversion. The Input is a Network database(NDB) and the output is a table structured flattened XML document. The system will read NDB according to NDB schema. In one-to-many data semantic, it will post owner and member records into XML sibling elements linked with id and idref. In many-to-many data semantic, it will post 2 owners and 1 common member records into XML sibling elements linked with id(s) and idref(s). In isa data semantic, it will post an owner and a member records into XML sibling elements linked with id and idref with the same key. In generalization data semantic, it will post owner and member records into table structured XML sibling elements linked with id(s) and idref(s), with the same key in the flattened XML document schema DTD "," separator.

Preprocess algorithm: Map Network schema into flattened XML schema:

1 Begin

2 If (owner record A has a key value attribute a1) and (member record B under owner record A has same key value a1)

3 Then begin

/* Map Record B isa record A into sibling element A isa sibling element B where A is subclass and B is superclass */

4 Map record A into sibling element A with key attribute a1 and with ID value

into flattened XML schema;

5 Map record B into sibling element B with same key attribute a1 and an IDREF referring to above ID value into flattened XML schema;

6 End;

7 If member record B under owner record A

8 Then begin

/*Map Records A and B in 1:n cardinality into sibling elements A and B in 1:n cardinality where A is one and B is many */

9 Map record A into sibling element A with ID value into flattened XML schema;

10 Map record B into sibling element B with IDREF referring to the above ID value into flattened XML schema;

11 End;

12 If (sibling element A and sibling element B is in 1:n cardinality) And (sibling element C and sibling element B is in 1:n)

13 Then sibling element A and sibling element C are in m:n cardinality;

14 If (sibling element A isa sibling element B) and (sibling element C isa sibling element B)

15 Then sibling element A and sibling element C are generalized into sibling element B;

/* A,C are subclasses to superclass B*/

16 End;

Process algorithm: Transform a NDB to a flattened XML document

Input: A NDB instance
Output: a flattened XML document

1 Begin
2 Read NDB record occurrences by using depth first search;
3 While not at end of occurrences do
4 begin

```
            For each owner record occurrence obtained
            Add a sibling element with an ID attribute id with value
            “entity:sequence_number”;
            For each member record occurrence obtained
            Add a sibling element with an IDREF attribute idref with value
            “parent_element_name:seqeuence number of its element;
        end;
end;
```

Step 2: Transform flattened XML documents into target's legacy databases[17]

In step 2, we can translate the flattened XML schema into another legacy database schema, followed by the data transformation of the flattened XML documents into a legacy database according to the translated legacy database schema. In this way, each source database data type can be read by the legacy database schema. Therefore, there is no need for physical data type conversion in this approach as shown in Figures 2. Therefore, we can post the flattened relational structured XML document into a legacy database of relational, object-oriented, network or XML.

Case 1: Transform flattened XML documents into relational databases

Firstly, we perform the preprocess of mapping flattened XML schema into relational database schema. Secondly, we perform their correspondent data conversion. The Input is a flattened XML document and the output is a relational database. The system will read flattened XML document according to flattened XML document schema. In one-to-many data semantic, it will post XML sibling elements into parent and child relations. In many-to-many data semantic, it will post XML sibling elements linked with id(s) and idref(s) into 2 parents and 1 child relations. In isa data semantic, it will post XML sibling elements into superclass relation and sub-class relation. In generalization data semantic, it will post XML sibling elements into a superclass relation and 2 subclass relations.

```
Preprocess algorithm: Map flattened XML schema into relational schema:
Begin
    If (sibling element A with ID value of relation key value) and (sibling
    element B with IDREF value of the same relation key value)
    Then begin
  /* Map Siblings elements A and B into relations A and B where B is a
    subclass to A */
  Map sibling element A into relation A with primary key = ID value;
    Map sibling element B into relation B with primary key = foreign key with
    same value;
      End;
  If (sibling element A with ID value) and (sibling element B with IDREF
```

```
        value of the same value)
    Then begin
   /* Map Sibling elements A and B in 1:n cardinality into relations A and B in
        1:n cardinality where A is one and B is many*/
      Map sibling element A into relation A with primary key = ID value;
      Map sibling element B into relation B with foreign key referring to
         primary key ID value;
       End;
  If  (sibling element A and sibling element B is in 1:n cardinality)
         And (sibling element C and sibling element B is in 1:n)
       Then sibling element A and sibling element C are in m:n cardinality;
If (sibling element A isa sibling element B) and (sibling element C isa
         sibling element B)
    Then sibling A and sibling element C are generalized into sibling element B;
    /* A,C are subclasses, and B is their superclass */
End;
```

Process algorithm: Create RDB SQL statements from flattened XML document

Input: flattened XML document

Output: A sequence of SQL statements

```
Begin
    Let s be an empty statement sequence;
    For each sibling XML element with entity prefix e do
       begin
          Derive table name t from sibling element name of e without entity prefix;
             For each sibling element c of e do
    /* extract attributes from the sibling-elements in flattened XML document */
    Begin
                   Derive col from name of c without property prefix;
                 Derive val from child text node contents of c;
             If c is the first sibling element
             Then begin
                   Let cols = "col";
                   Let vals = "'val'";
             End;
             Else begin
                   Append ",col" to cols;
                   Append ",'val'" to vals;
             End;
             End
             Let i = "INSERT INTO t (cols) VALUES (vals)";
                   Add i to s;
        End
        Return s
    End
```

Case 2: Transform flattened XML documents into object-oriented databases

Firstly, we perform the preprocess of mapping flattened XML schema into object-oriented schema. Secondly, we perform their correspondent data conversion. The Input is a flattened XML document and the output is an object-oriented database. The system will read flattened XML document according to flattened XML document schema. In one-to-many data semantic, it will post XML sibling elements into a pair of associated objects with OID and Stored OID. In many-to-many data semantic, it will post XML sibling elements linked with id(s) and idref(s) into a pair of associated objects. In isa data semantic, it will post XML sibling elements into superclass and its sub-class object. In generalization data semantic, it will post flattened structured XML sibling elements with the same key into objects and their subclass objects with the same OID.

Preprocess algorithm: Map flattened XML schema into object-oriented schema:

```
1 Begin
2     If (sibling element A with key attribute a1 and an ID value) And (sibling element
      B with same key attribute a1 and an IDREF value same as the above ID value)
3     Then begin
          /* Map sibling element B isa sibling element A into class B isa class A;
             where subclass B refer to superclass A*/
4            Map sibling element A into class A with attribute a1 into object-oriented
             schema;
5            Map sibling element B into subclass B of class A in object-oriented schema;
6            end;
7       If (sibling element A with an ID value)
      And (sibling element B with an IDREF value referring to the above ID value)
8     Then begin
        /* Map sibling elements A and B in 1:n cardinality into classes A and B in 1:n
        cardinality where A is one and B is many */
9       Map sibling element A into class A with association attribute A2B referring
          to class B's multiple objects in OODB schema;
10        Map sibling element B into class B with association attribute B2A referring
          to class A's object in OODB schema;
```

11 end;

12 If (sibling element A and sibling element B is in 1:n cardinality)
And (sibling element C and sibling element B is in 1:n)

13 Then sibling element A and sibling element C are in m:n cardinality;

14 If (sibling element A isa sibling element B) and (sibling element C isa sibling element B)

15 Then sibling A and sibling element C are generalized into sibling element B;
/* A,C are subclass, and B is their superclass */

16 end;

Process algorithm: Create OODB statements from flattened XML documents

Input: flattened XML document

Output: A sequence of OODB OQL statements

1 Begin

2 Given sibling element A_1 is with idref=id as “one” side only;

3 For i = 1 to m do
/* for each sibling element Ai with data occurrence A1….Am */

4 For j = 1 to n do
/* for each sibling element Aj data occurrence A1…An such that i≠j*/

5 Begin

6 If (sibling element Ai ID name = sibling element Ai IDREF name)
and (sibling element Aj ID name = sibling element Ai IDREF name)

7 Then sibling element Ai isa sibling element Aj;
/* subclass element Ai and superclass element Aj */

8 If sibling element Ai ID name = sibling element Aj IDREF name

9 Then sibling element Ai and sibling element Aj are in 1:n cardinality;
/* element Ai links many element Aj */

10 If sibling element Ai IDREF name = sibling element Aj ID name

11 Then sibling element Ai and sibling element Aj are in n:1 cardinality;
/* many element Ai links element Aj */

```
        Case sibling element Ai and sibling element Aj are in
1:n begin
        Output insert statement with Ai data + association attribute value "{}";
        Output insert statement with Aj data;
        End;

    n:1 begin
        Output insert statement with Ai data;
        Output insert statement with Aj data + association attribute null value;
        End;

Isa: begin
        Output insert statement with Ai data + to-be-inherited superclass attributes
  null value;
        Output insert statement with Aj data;
        End;
Case end;
End;

 For i = 1 to m do
/* for each sibling element Ai with data occurrence A1….Am */
 For j = 1 to n do
/* for each sibling element Aj data occurrence A1…An such that i≠j*/
 Begin
        Case sibling element Ai and sibling element Aj are in
    1:n:      Output update statement of Aj to replace "{}" value by selected
                OID(s);
    n:1:      Output update statement of Aj to replace null value by selected
              OID;
    isa:   Output update statement of Ai to replace null value with inherited Aj
                  data by select statement;

  case end;
        end;
end
```

Case 3: Transform flattened XML documents into network databases:

Firstly, we perform the preprocess of mapping flattened XML schema into network schema. Secondly, we perform their correspondent data conversion. Network database model is the earliest database model among the four legacy databases being concerned. There are no standard data definition language (DDL) and data manipulation language (DML). Database in network database model are accessed by making function invocations of the application-programming interface (API) that comes with the database products. Database manipulation operations are written in third-generation languages (3GL's) such as COBOL and C.

The Raima database is used as the reference network database implement. In order to import data to the NDB, Raima provides utility that can read sequence data file. Therefore, the algorithm provided below is to translate the flattened XML document file into plain text sequential file.

For example, the Raima database defines its own data definition language. To define an entity type with properties, use a record definition:

```
record investor {
double money_mkt;
char name;
unique key short invID;}
```

To define the linkages among the entities, use the set definition:

```
set inv_trans {
order last;
owner investor;
member asset;}
```

Once the database definition is properly defined with a DDL file, Raima provides utility application and API for creating the database.

The Input is a flattened XML document and the output is a network database. The system will read flattened XML document according to flattened XML document schema. In one-to-many data semantic, it will post XML sibling elements into a pair of owner and member records. In many-to-many data semantic, it will post XML sibling elements linked with id(s) and idref(s) into 2 owners link with 1 member record with the same key. In isa data semantic, it will post XML sibling elements into 1 owner and 1 member record with the same key. In generalization data semantic, it will post XML sibling elements linked with id(s) and idref(s) into 1 owner and 2 member records with the same key.

Preprocess algorithm: Map flattened XML schema into network schema:

1 Begin

2 If (sibling element A with an attribute a1 and an ID value) And (sibling element B with same attribute a1 and an IDREF value same as the above ID value)

3 Then begin

/*Map sibling element B isa sibling element A into record B isa record A where B is subclass, and A is superclass */

4 Map sibling element A with key attribute a1 into owner record A with key attribute a1 into network schema;

5 Map sibling element B with key attribute a1 into member record B under record A with same key attribute a1 into network schema;

6 end;

7 If (sibling element A with an ID value) And (sibling element B with an IDREF value referring to the above ID value)

8 Then begin

/* Map sibling elements A and B in 1:n cardinality into records A and B in 16 1:n cardinality where A is one and B is many */

9 Map sibling element A with attribute ID value a1 into owner record A with 18 key attribute a1 into network schema;

10 Map sibling element B into member record B under record A into network schema;

11 end;

12 If (sibling element A and sibling element B is in 1:n cardinality)

And (sibling element C and sibling element B is in 1:n)

13 Then sibling element A and sibling element C are in m:n cardinality;

14 If (sibling element A isa sibling element B) and (sibling element C isa sibling

```
    element B)

        Then sibling A and sibling element C are generalized into sibling element B;
          /* A,C are subclasses, and B is their superclass */

end;
```

Process algorithm Step 1: Create CSV file from flattened XML document

```
Read flattened XML document
For each XML element e do
Begin
    Derive the internal table name t from element name of e;
    Use t as the CSV file name;
    For each sibling element c of e do;
    Begin
        Derive val from attribute contents of c;
        If c is the first sibling element
        Then begin
                Let vals =’val’;
            End;
        Else begin
                Let vals =’,’;
                Append “val” to vals;
            End;
        Add vals to the CSV file;
    End;
    Export the CSV file;
End
```

//Step 2: Macro-call program

Macro-call: We need to use a utility provided by NDB DBMS (Raima) to import the data from CSV file to the database. The utility is named “dbimp”. “dbimp” is in command format and only executable in command prompt. Before we use “dbimp”, we must write a text-based import file. The import file first defined which database we want to import data. Then, for each record, we need to specify the CSV file to import data. This is achieved by “foreach” command and followed by the CSV file name. After this, we used “{“ and “}” to include the record name and attribute name. We used the keyword “field” in front of each attribute. For example,

```
Database !network database name
foreach “!data file name.csv” {
    record ! =”record name”
field !field name = 1;
…
Field !field name = n”
}
```

//Step 3: Upload data and query the NDB instance (in Raima) by computer automation

Import data to the NDB instance by use of utility “dbimp”.

If the data import successfully, all data will be query and output simultaneously.

End

Case 4: Transform flattened XML documents into XML databases

Firstly, we perform the preprocess of mapping flattened XML schema into XML schema. Secondly, we perform their correspondent data conversion. The flattened XML documents format is in XML format with three nested levels, which are root, entity element and column element, and each column element instance encloses a text node for the column value. On the other hand, usual XML document can be in any nested structure and the number of nested level is unlimited. Therefore, in order to convert arbitrary XML documents into the corresponding flattened relational structured XML document format structure, the following process is used:

The Input is a flattened XML document and the output is an XML document. The system will read flattened XML documents according to flattened XML documents schema. In one-to-many data semantic, it will post XML sibling elements into a pair of XML element and sub-elements. In many-to-many data semantic, it will post XML sibling elements linked with id(s) and idref(s) into XML elements and sub-element. In isa data semantic, it will post XML sibling elements with the same key into XML element and sub-elements with the same key. In generalization data semantic, it will post XML sibling elements into XML element and sub-elements with the same key.

Preprocess algorithm: Map flattened XML schema into XML schema:

```
Begin
    If   (sibling element A with an attribute a1 and an ID value) And (sibling
  element B with same attribute a1 and an IDREF value same as the above ID value)
   Then begin
        /*Map sibling element B isa sibling element A into element B isa element A
     where B is subclass and A is superclass */
         Map sibling element A into element A with attribute a1 in XML schema;
        Map sibling element B into element B with attribute a1 and IDREF value
               same as the above ID value in XML schema;
         end;
   If   (sibling element A with an ID value)
     And (sibling element B with an IDREF value referring to the above ID value)
   Then begin
```

/*Map sibling elements A and B in 1:n cardinality into elements A and B in 1:n cardinality where A is one and B is many */

9 Map sibling element A into element A in XML schema;

10 Map sibling element B into element B under element A in XML schema;

11 end;

12 If (sibling element A and sibling element B is in 1:n cardinality)

And (sibling element C and sibling element B is in 1:n)

13 Then sibling element A and sibling element C are in m:n cardinality;

14 If (sibling element A isa sibling element B) and (sibling element C isa sibling element B)

15 Then sibling A and sibling element C are generalized into sibling element B;

/* A,C are subclasses, and B is their superclass */

16 end;

Process algorithm: Post flattened XML document into an XML document

Input: A flattened XML document

Output: An XML document

1 Begin

2 Let xml = replicate of flattened XML document;

3 Call Restructure XML with xml;

4 Return xml

5 End

6 Function: Restructure XML

7 Begin

8 For each sibling XML element e with one IDREF attribute idref do

9 begin

10 Locate sibling element e’ with ID referred by idref;

11 Move e as child element of e’;

12 Remove attribute idref from element e;

13 End

14 End

Chapter 5 Case study with prototype

The prototype below is to prove that the methodology in Chapter 4 is feasible. By experiment, chapter 5 emphasizes in the preservation of data constraint of functional dependency, inclusion dependency and multi-value dependence before and after data conversion (transformation).

The prototype is to prove that the data dependencies of a source RDB in a case study can be transformed into an XML DB, which can be further transformed into a target RDB with the preservation of its data semantics in the form of FD, ID and MVD.

In general, a DB (database) can be converted without any loss of information if p maps a state of a legacy database into another legacy DB, and p' maps a state of a legacy DB into another legacy DB, then it can be shown that p(p'(N)) = N where N is the legacy DB before conversion.

A logistic system records the customer shipment information including which orders are being packed and what the packing information is. Based on the relational schema below, there are three intermediate independent entities: PL_INFORMAION recording the general information of the shipment, PL_LINE_INFORMATION storing the packing information — particularly information about the BOXES — and ORDER_INFORMATION storing the information of orders such as the product information. A many-to-many relationship between ORDER_INFORMATION and PL_LINE_DETAIL must be resolved early in the modeling process to eliminate repeating information when representing PL_INFORMATION or ORDER_INFORMATION. The strategy for resolving many-to-many relationship is to replace the relationship with two one-to-many cardinalities. As a result, these two one-to-many relationships are between PL_LINE_INFORMATION and PL_LINE_DETAIL, and between ORDER_INFORMATION and PL_LINE_DETAIL. Similarly, the superclass ORDER_INFOR MATION can be divided into two subclasses BulkOrder and CustomerOrder in generalization as shown in Figure 10.

Table 3: Source Relational database

Table PL_INFORMATION

PL_INFORMATION_SEQNO	ISSUE_DATE	DATA_LAST_MODIFIED	LAST_MODIFIED_BY	PL_STATUS	PL_HEADER+REMARKS	SHIPMENT_TYPE	SHIPMENT_DATE	EXPERCTED_ARRIVAL_DATE
EFG123DS	2004-07-31	2004-08-02	JOEY	S	SOME GOODS ARE BREAKABLE	TRAIN	2004-08-03	2004-08-03

Table PL_LINE_INFORMATION

*PL_INFROAMATION-SEQNO	PL_LINE_INFORMATOIN_SEQNO	PACKAGE_TYPE	LENGTH_UNIT_OF_MEASURE	WIDTH_UNIT_OF_MEASURE	HEIGHT_UNIT_OF_MEASURE	WEIGHT_UNIT_OF_MESSAGE
EFG123DS	ABCV234F	BOX	20	20	20	40
EFG123DS	ABCN439WS	BAG	7	13	10	13

Table PL_LINE_DETAIL

*PL_INFORMATION_SEQNO	*PL_LINE_INFORMATION_SEQNO	*ORDER_NUMBER	ITEM_NUMBER	TOTAL_PACKED_QTY	TOTAL_GROSS_WEIGHT	TOTAL_VOLUME_LENGTH	TOTAL_VOLUME_WIDTH	TOTAL_VOLUMEN_HEIGHT
EFG123DS	ABCV234F	135792468	1	4	12	5	2	6
EFG123DS	ABCV234F	123469999	2	1	28	8	4	6
EFG123DS	ABCH439WS	135792468	1	4	12	5	2	6

Table ORDER_INFORMATION

ORDER_NUMBER	BRAND	DIVISION	CUSTOMER_ORDER_NUMBER	CUSTOMER_NUMBER	ORDER_TPYE	MODEL_NUMBER	MODEL_DESCRIPTION	ORDER_DATE	ORDERD_QTY	PRICE_PRE_UNIT	DISCOUNT
135792468	ABC	CLOTHING	135792468	MA23456	MAIL	AS1234	ADULT T-SHIRT SIZE M	2004-07-27	8	10.5	
123469999	DONY	TOYS	123456999	MA23456	PHONE	PS2	PLAYSTATION	2004-07-29	1	1399	10

Table BulkOrder

*ORDER_NUMBER	CUSTOMER_NAME	SIZE_INDEX	ORDERED_QTY	UNIT_PRICE
135792468	AMAZON	S	2000	12.1

Table TailorMadeOrder

*ORDER_NUMBER	CUSTOMER_NAME	SIZE_INDEX	ORDERED_QTY	UNIT_PRICE
123469999	PETER CHAN	L	3000	12.3

Step 1: Transform from relational database into flattened XML document:

Example 1: Transform from relational database into flattened XML document

The input relational conceptual schema in Extended Entity Relationship model

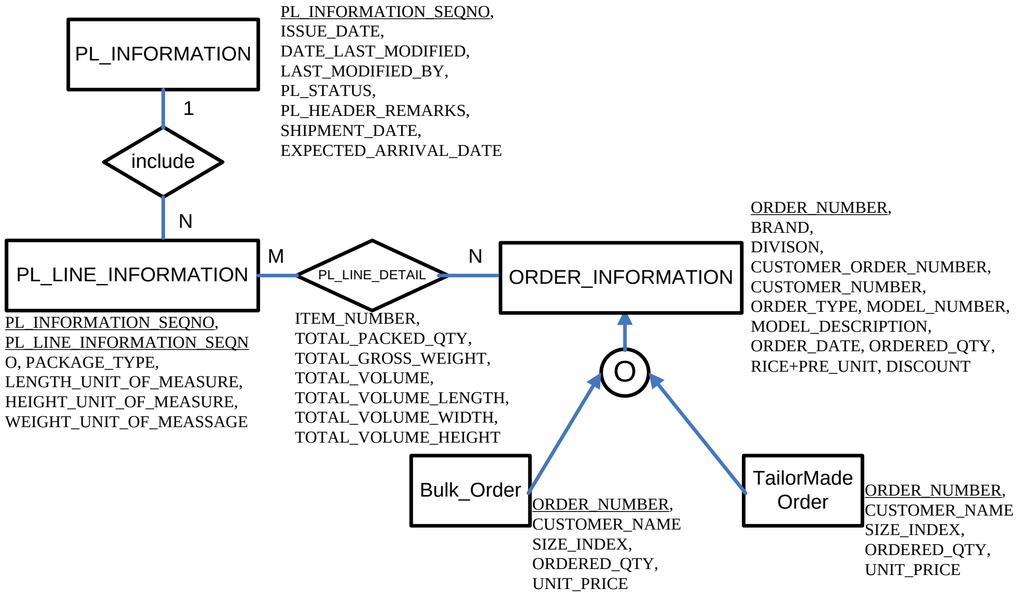


Figure 10 Input relational database in Extended Entity Relationship model

There are six tables. Each table has its primary key in italic, and foreign key prefixed with "*". Their data dependencies (DD) are such that each foreign key determines its referred primary key in FD, and subclass foreign key is a subset of its superclass primary key in ID as follows:

FD_1: *PL_Line_information*. PL_INFORMATION_SEQNO → *PL_information*.PL_INFORMATION_SEQNO

FD_1 represent PL_INFORMAION and PL_Line_information are in one-to-many cardinality.

ID_1: *Bulk_Order*. BulkOrder.ORDER_NUMBER $\sqsubseteq$ *Order_information* . *Order_NUMBER*

ID_1 represent subclass BulkOrder and superclass ORDER_INFORMATION are in ISA relationship.

ID_2: *TailorMadeOrder*.TailorMadeOrder.ORDER_NUMBER $\sqsubseteq$ *Order_information* . *Order_NUMBER*

ID_2 represent subclass TailorMadeOrder and superclass ORDER_INFORMATION ae in ISA relationship.

MVD_1: *PL_Line_information*. PL_INFORMATION_SEQNO →→ *Order_information* . *Order_NUMBER*

MVD_2: *Order_information* . *Order_NUMBER* →→ *PL_Line_information*. PL_INFORMATION_SEQNO

Therefore: MVD_1 and MVD_2 represent that PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. (Note: 2 one-to-many cardinalities are equivalent to many-to-many cardinality)

Example: The layout of the input relational database can be shown in Figure 11.

mysql> select * from pl_information;

SHIPMENT_DATE	ISSUE_DATE	DATE_LAST_MODIFIED	PL_INFORMATION_SEQNO	PL_STATUS	SHIPMENT_TYPE	PL_HEADER_REMARKS	EXPECTED_ARRIVAL_DATE	LAST_MODIFIED_BY
2004-08-03	2004-07-31	2004-08-02	EFG123DS	S	TRAIN	SOME GOODS	2004-08-03	JOEY

1 row in set (0.00 sec)

mysql> select * from pl_line_information;

PL_INFORMATION_SEQNO	PL_LINE_INFORMATION_SEQNO	WIDTH_UNIT_OF_MEASURE	HEIGHT_UNIT_OF_MEASURE	PACKAGE_TYPE	WEIGHT_UNIT_OF_MESSAGE	LENGTH_UNIT_OF_MEASURE
EFG123DS	ABCH439WS	13	10	BAG	13	7
EFG123DS	ABCV234F	20	20	BOX	40	20

2 rows in set (0.00 sec)

mysql> select * from pl_line_detail;

TOTAL_VOLUME_LENGTH	PL_INFORMATION_SEQNO	TOTAL_GROSS_WEIGHT	TOTAL_VOLUME_WIDTH	TOTAL_VOLUMEN_HEIGHT	ORDER_NUMBER	TOTAL_PACKED_QTY	PL_LINE_INFORMATION_SEQNO	ITEM_NUMBER
5	EFG123DS	12	2	6	135792468	4	ABCH439WS	1
8	EFG123DS	28	4	6	123469999	1	ABCV234F	2
5	EFG123DS	12	2	6	135792468	4	ABCV234F	1

3 rows in set (0.00 sec)

mysql> select * from order_information;

CUSTOMER_ORDER_NUMBER	DISCOUNT	CUSTOMER_NUMBER	ORDER_TPYE	BRAND	MODEL_DESCRIPTION	MODEL_NUMBER	ORDER_NUMBER	ORDERD_QTY	DIVISION	ORDER_DATE	PRICE_PER_UNIT
123456999	10	MA23456	PHONE	DONY	PLAYSTATIO	PS2	123469999	1	TOYS	2004-07-29	1399.0
135792468	5	MA23456	MAIL	ABC	ADULT T-SH	AS1234	135792468	8	CLOTHING	2004-07-27	10.5

2 rows in set (0.00 sec)

mysql> select * from bulk_order;

CUSTOMER_NAME	UNIT_PRICE	ORDER_NUMBER	SIZE_INDEX
AMAZON	12.1	135792468	S

1 row in set (0.00 sec)

mysql> select * from tailor_made_order;

CUSTOMER_NAME	UNIT_PRICE	ORDER_NUMBER	SIZE_INDEX
PETER CHAN	12.3	135792468	L

1 row in set (0.00 sec)

Figure 11 Source Relational database in case study

We map input relational schema into a flattened XML OUDG schema with relational structure in two levels tree only as shown in Figure 13. Notice that the second level sibling elements (under root elements) are linked together using idref referring to id, which is similar to foreign key referring to primary key in relational database.

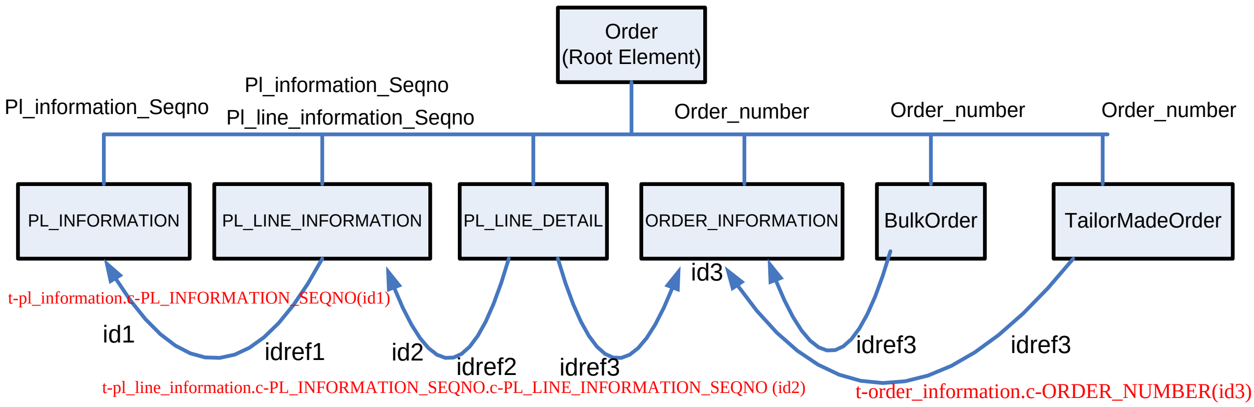


Figure 12 Transformed flattened XML document conceptual schema in DTD Graph

There are seven elements. The second level elements has id(s) and/or idref(s). Their data dependencies are such that each idref determines its referred id FD as follows:

FD_1: t-pl_line_information. *t-pl_information.1*→ t-pl_information. *t-pl_information.1*

FD_1 represent PL_INFORMAION and PL_Line_information are in one-to-many cardinality. FD_1 in flattened XML is same as FD_1 in RDB source. Therefore, one-to-many cardinality between PL_INFORMAION and PL_Line_information is preserved.

ID_1: t-bulk_order. *t-order_information.1* $\sqsubseteq$ t-order_information. *t-order_information.1*

ID_1 represent BulkOrder and ORDER_INFORMATION are in ISA relationship. ID_1 in flattened XML is same as ID_1 in RDB source. Therefore, ISA relationship between BulkOrder and ORDER_INFORMATION is preserved.

ID_2: t-tailor_made_order. *t-pl_information.2* $\sqsubseteq$ t-order_information. *t-pl_information.2*

ID_2 represent TailorMadeOrder and ORDER_INFORMATION are in ISA relationship. ID_2 in flattened XML is same as ID_2 in RDB source. Therefore, ISA relationship between TailorMadeOrder and ORDER_INFORMATION is preserved.

MVD_1: $id_2 \rightarrow\rightarrow id_3$

MVD_2: $id_3 \rightarrow\rightarrow id_2$

Therefore: MVD_1 and MVD_2 represent PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. MVD_1 and MVD_2 in flattened XML is same as MVD_1 and MVD_2 in RDB source. Therefore, many-to-many cardinality between TailorMadeOrder and ORDER_INFORMATION is preserved.

The flattened XML document is shown in Figure 13.

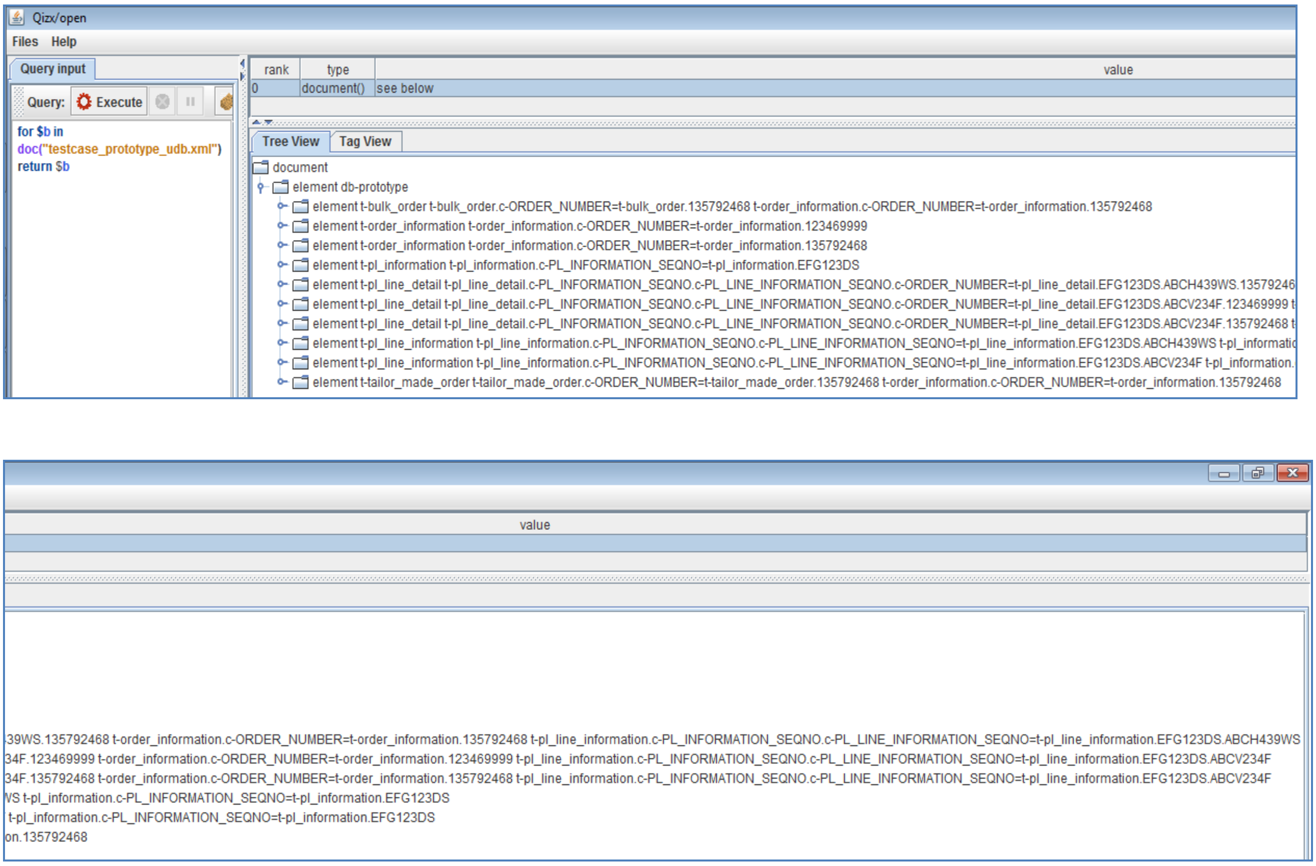


Figure 13 Transformed flattened XML document

Step 2: Transform from flattened XML document into legacy databases

We can then map the open universal database schema into legacy databases as follows:

(a) Data transformation from flattened XML document into XML document

We can map the open universal database schema into XML database schema as shown in Figure 14 which shows that elements Pl_information and Pl_line_information are in element and sub-element 1:n association. Elements Pl_line_information, Pl_line_detail and Order_information are in m:n association linked by pairs of idref referring to id. Elements Order_information and Bulk_Order are in ISA relationship. Elements Order_information and TailorMadeOrder are also in in ISA relationship. This XML structure has multiple levels of elements which is different from the 2 levels elements in flattened XML document.

Example 2: Transform from flattened XML document into legacy databases

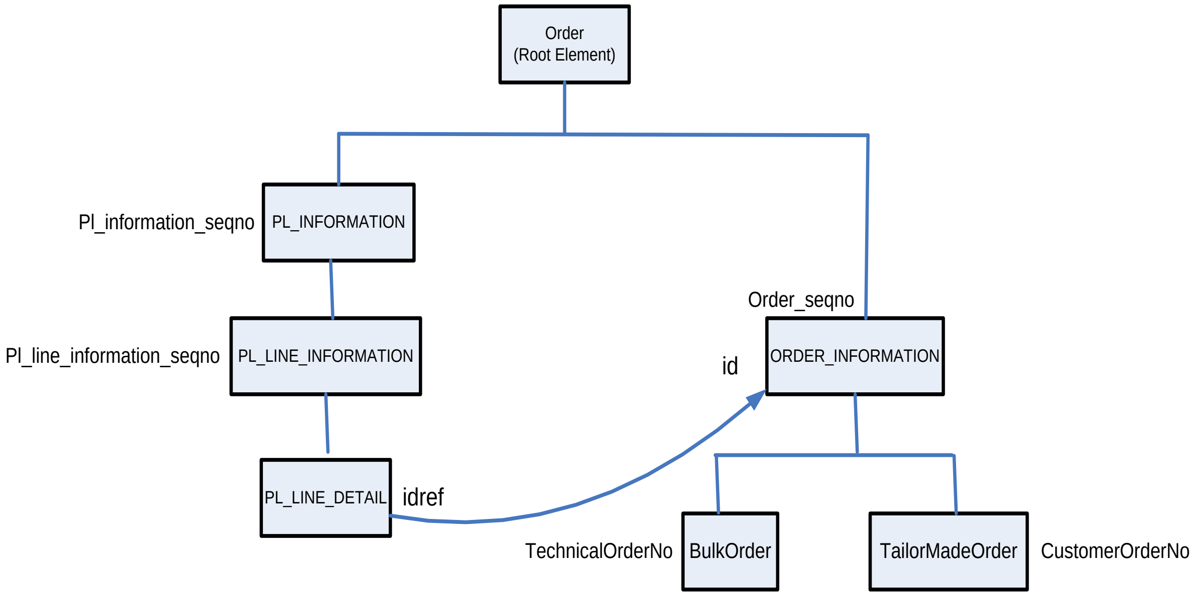


Figure 14 Translated XML document schema in DTD Graph

For example, figure 14 shows the mapping from flattened XML schema into object-oriented database schema in UML. The class PL_Information and class Pl_line_information are in 1:n association. Classes Pl_line_information and Order_information are in m:n association with class Pl_line_detail as association class in between sub classes BulkOrder and TailorMadeOrder which are in disjoint generalization under their superclass Order_information such that the two subclasses data are mutually exclusive.

There are seven elements. The sub-element key determines its element key in FD. The idref can determine its referred id in FD. The subclass element key is a subset of its superclass key in ID as follows:

FD$_1$: pl_line_information. *pl_information.seqno* → pl_information. *pl_information seqno*

FD$_1$ represent PL_INFORMAIION and PL_Line_information are in one-to-many cardinality. FD$_1$ in XML is same as FD$_1$ in flattened XML schema. Therefore, one-to-many cardinality between PL_INFORMAION and PL_Line_information is preserved.

ID$_1$: bulk_order. *order_number* $\sqsubseteq$ order_information. *order_number*

ID_1 represent BulkOrder and ORDER_INFORMATION are in ISA relationship. ID_1 in XML is same as ID_1 in flattened XML schema. Therefore, ISA relationship between BulkOrder and ORDER_INFORMATION is preserved.

ID_2: tailor_made_order. *order_number* $\subseteq$ order_information. *order_number*

ID_2 represent TailorMadeOrder and ORDER_INFORMATION are in ISA relationship. ID_2 in flattened XML is same as ID_2 in flattened XML schema. Therefore, ISA relationship between TailorMadeOrder and ORDER_INFORMATION is preserved.

MVD_1: *PL_Line_information*. PL_INFORMATION_SEQNO $\rightarrow\rightarrow$ *Order_information* . *Order_NUMBER*

MVD_2: *Order_information* . *Order_NUMBER* $\rightarrow\rightarrow$ *PL_Line_information*. PL_INFORMATION_SEQNO

MVD_1 and MVD_2 represent PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. MVD_1 and MVD_2 in XML is same as MVD_1 and MVD_2 in flattened XML schema. Therefore, many-to-many cardinality between TailorMadeOrder and ORDER_INFORMATION is preserved.

The transformed XML database document is:

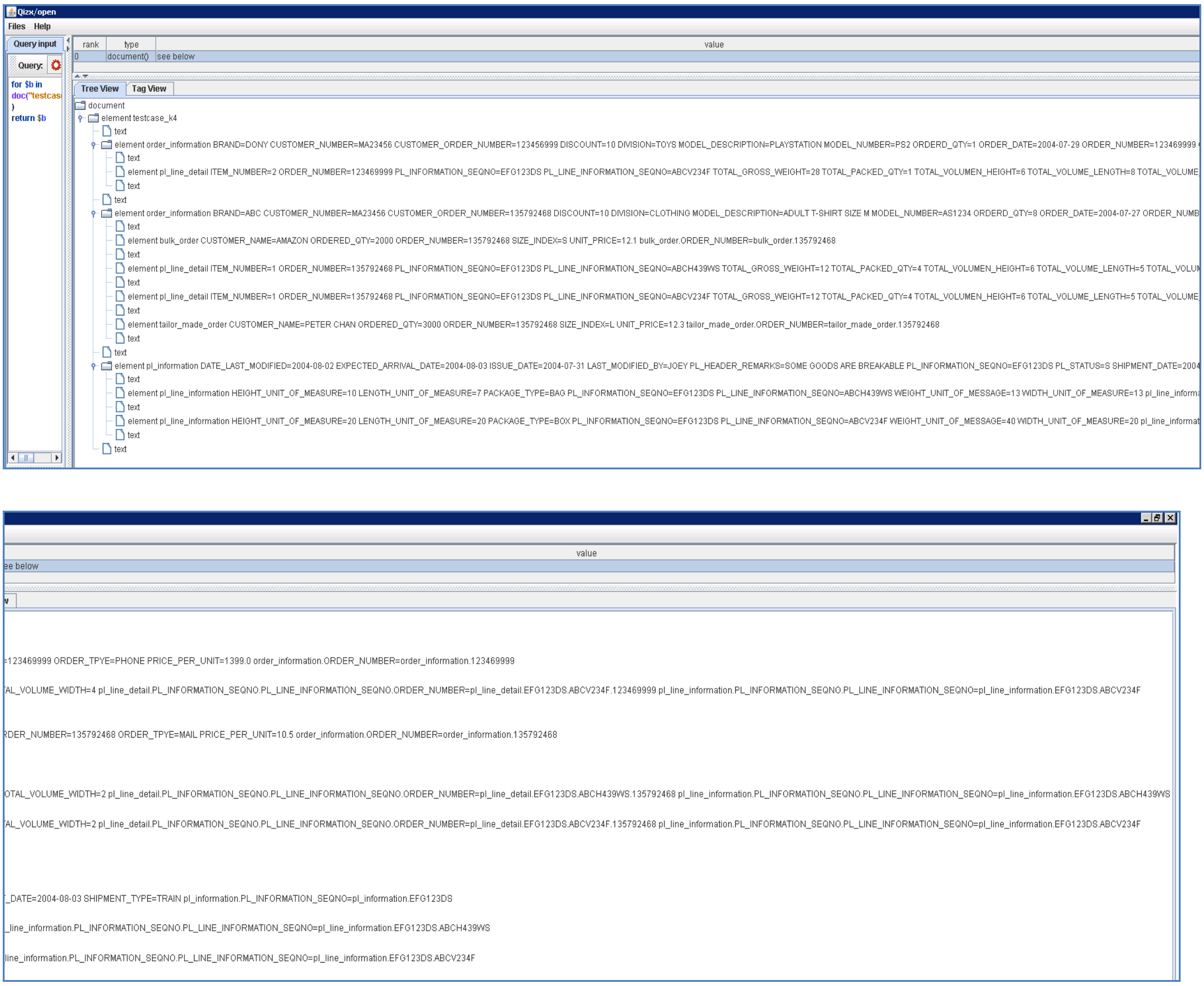


Figure 15 Transformed XML document

(b) Transform flattened XML documents into Object-Oriented databases

Example 3: Transform flattened XML documents into Object-Oriented databases

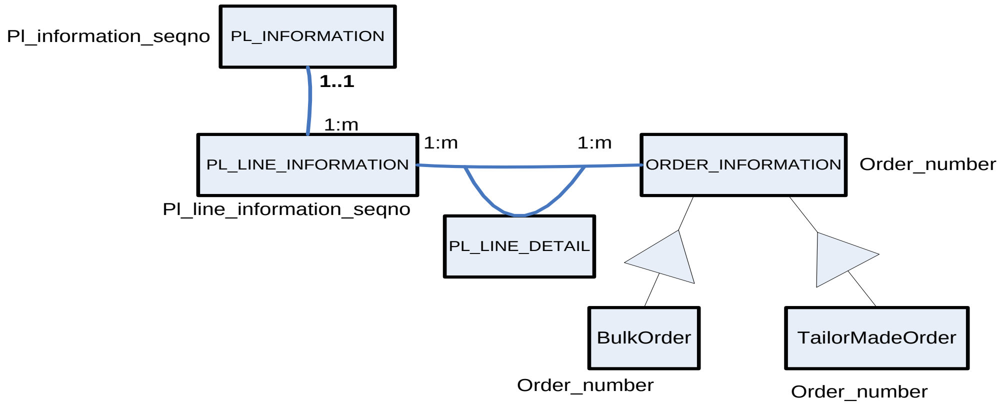


Figure 16 Translated legacy object-oriented database schema in UML

In figure 16, record Pl_informations and Order_information are under object-oriented DBMS as first records for database navigation access path. The path can go from class Pl_information to class Pl_line_information in 1:n relationship. Classes Pl_line_information, Order_information and Pl_line_detail are in m:n relationship. Classes Order_information and BulkOrder are in ISA relationship. Similarly, records Order_information and TailorMadeOrder are in ISA relationship.

There are six classes. Each class has its OID, and Stored OID. Their data dependencies are such that each Stored OID key determines its referred OID in FD, and each subclass OID is a subset of its superclass OID in ID as follows:

FD_1: pl_line_information. Stored_OID → pl_information. *OID*

FD_1 represent PL_INFORMAION and PL_Line_information are in one-to-many cardinality. FD_1 in OODB is same as FD_1 in object-oriented schema source. Therefore, one-to-many cardinality between PL_INFORMAION and PL_Line_information is preserved.

ID_1: bulk_order. OID $\subseteq$ order_information. *OID*

ID_1 represent BulkOrder and ORDER_INFORMATION are in ISA relationship. ID_1 in OODB is same as ID_1 in object-oriented schema source. Therefore, ISA relationship between BulkOrder and ORDER_INFORMATION is preserved.

ID_2: tailor_made_order. OID $\subseteq$ order_information. *OID*

ID_2 represent TailorMadeOrder and ORDER_INFORMATION are in ISA relationship. ID_2 in OODB is same as ID_2 in object-oriented schema source .Therefore, ISA relationship between TailorMadeOrder and ORDER_INFORMATION is preserved.

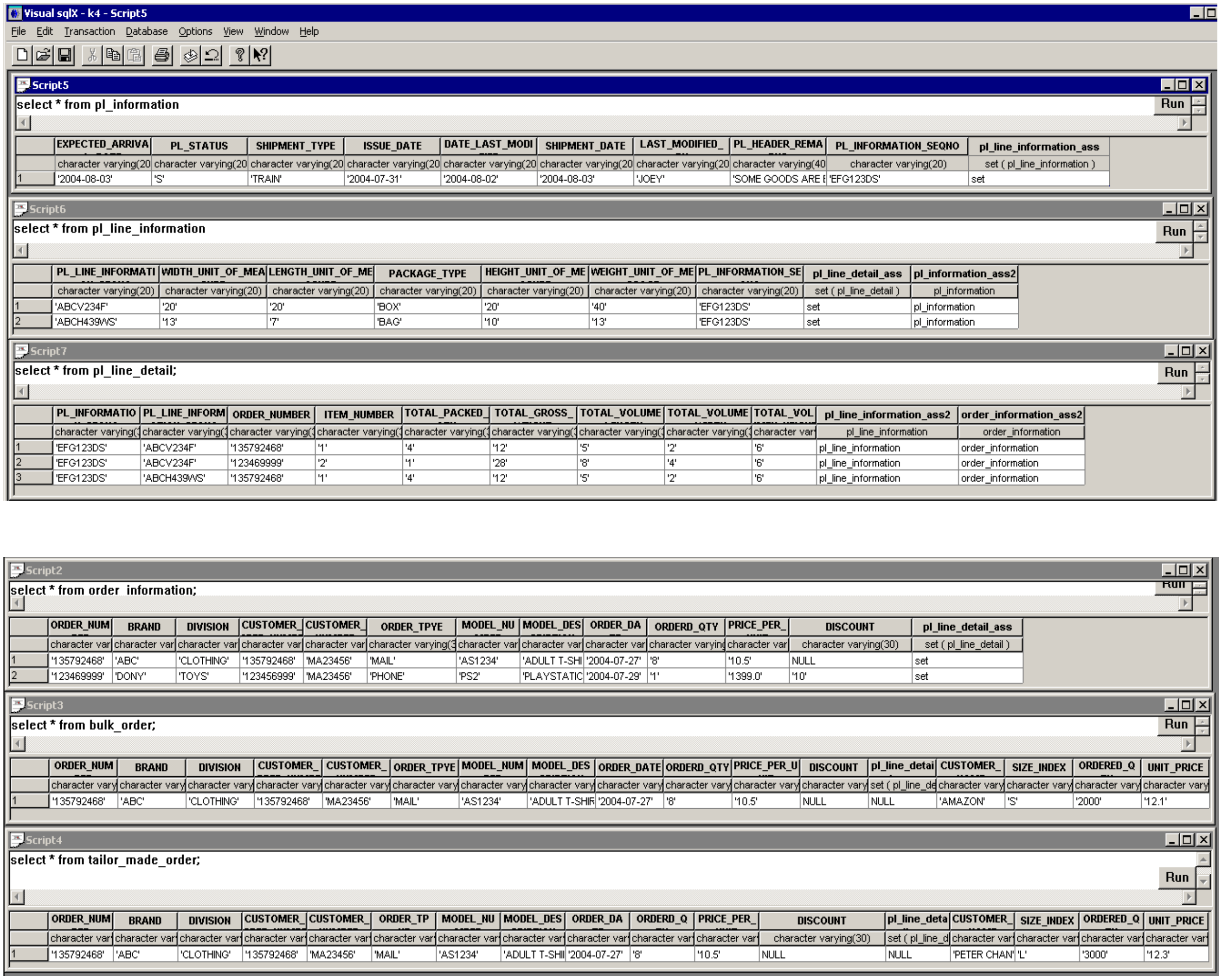
MVD_1: *PL_Line_information*. PL_INFORMATION_SEQNO →→ *Order_information* . *Order_NUMBER*

MVD_2: *Order_information* . *Order_NUMBER* →→ *PL_Line_information*. PL_INFORMATION_SEQNO

MVD_1 and MVD_2 represent PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. Therefore, MVD_1 and MVD_2 in OODB is same as MVD_1 and MVD_2 in flattened XML schema source, many-to-many cardinality between subclass TailorMadeOrder and superclass ORDER_INFORMATION is preserved.

(Note: 2 one-to-many cardinalities are equivalent to many-to-many cardinality)

The transformed Object-Oriented Database Base is:



Visual sqlX - k4 - Script5

Script5

select * from pl_information

	EXPECTED_ARRIVA	PL_STATUS	SHIPMENT_TYPE	ISSUE_DATE	DATE_LAST_MODI	SHIPMENT_DATE	LAST_MODIFIED_	PL_HEADER_REMA	PL_INFORMATION_SEQNO	pl_line_information_ass
	character varying(20	character varying(20	character varying(20	character varying(20	character varying(20	character varying(20	character varying(20	character varying(40	character varying(20)	set (pl_line_information)
1	'2004-08-03'	'S'	'TRAIN'	'2004-07-31'	'2004-08-02'	'2004-08-03'	'JOEY'	'SOME GOODS ARE I	'EFG123DS'	set

Script6

select * from pl_line_information

	PL_LINE_INFORMATI	WIDTH_UNIT_OF_MEA	LENGTH_UNIT_OF_ME	PACKAGE_TYPE	HEIGHT_UNIT_OF_ME	WEIGHT_UNIT_OF_ME	PL_INFORMATION_SE	pl_line_detail_ass	pl_information_ass2
	character varying(20)	character varying(20)	character varying(20)	character varying(20)	character varying(20)	character varying(20)	character varying(20)	set (pl_line_detail)	pl_information
1	'ABCV234F'	'20'	'20'	'BOX'	'20'	'40'	'EFG123DS'	set	pl_information
2	'ABCH439WS'	'13'	'7'	'BAG'	'10'	'13'	'EFG123DS'	set	pl_information

Script7

select * from pl_line_detail;

	PL_INFORMATIO	PL_LINE_INFORM	ORDER_NUMBER	ITEM_NUMBER	TOTAL_PACKED_	TOTAL_GROSS_	TOTAL_VOLUME	TOTAL_VOLUME	TOTAL_VOL	pl_line_information_ass2	order_information_ass2
	character varying(	character varying(	character varying(	character varying(	character varying(	character varying(	character varying(	character varying(	character var	pl_line_information	order_information
1	'EFG123DS'	'ABCV234F'	'135792468'	'1'	'4'	'12'	'5'	'2'	'6'	pl_line_information	order_information
2	'EFG123DS'	'ABCV234F'	'123469999'	'2'	'1'	'28'	'8'	'4'	'6'	pl_line_information	order_information
3	'EFG123DS'	'ABCH439WS'	'135792468'	'1'	'4'	'12'	'5'	'2'	'6'	pl_line_information	order_information

Script2

select * from order_information;

	ORDER_NUM	BRAND	DIVISION	CUSTOMER_	CUSTOMER_	ORDER_TPYE	MODEL_NU	MODEL_DES	ORDER_DA	ORDERD_QTY	PRICE_PER_	DISCOUNT	pl_line_detail_ass
	character var	character var	character var	character var	character var	character varying(3	character var	character var	character var	character varying	character var	character varying(30)	set (pl_line_detail)
1	'135792468'	'ABC'	'CLOTHING'	'135792468'	'MA23456'	'MAIL'	'AS1234'	'ADULT T-SHI	'2004-07-27'	'8'	'10.5'	NULL	set
2	'123469999'	'DONY'	'TOYS'	'123456999'	'MA23456'	'PHONE'	'PS2'	'PLAYSTATIC	'2004-07-29'	'1'	'1399.0'	'10'	set

Script3

select * from bulk_order;

	ORDER_NUM	BRAND	DIVISION	CUSTOMER_	CUSTOMER_	ORDER_TPYE	MODEL_NUM	MODEL_DES	ORDER_DATE	ORDERD_QTY	PRICE_PER_U	DISCOUNT	pl_line_detai	CUSTOMER_	SIZE_INDEX	ORDERED_Q	UNIT_PRICE
	character vary	character vary	character vary	character vary	character vary	character vary	character vary	character vary	character vary	character vary	character vary	character vary	set (pl_line_de	character vary	character vary	character vary	character vary
1	'135792468'	'ABC'	'CLOTHING'	'135792468'	'MA23456'	'MAIL'	'AS1234'	'ADULT T-SHIR	'2004-07-27'	'8'	'10.5'	NULL	NULL	'AMAZON'	'S'	'2000'	'12.1'

Script4

select * from tailor_made_order;

	ORDER_NUM	BRAND	DIVISION	CUSTOMER_	CUSTOMER_	ORDER_TP	MODEL_NU	MODEL_DES	ORDER_DA	ORDERD_Q	PRICE_PER_	DISCOUNT	pl_line_deta	CUSTOMER_	SIZE_INDEX	ORDERED_Q	UNIT_PRICE
	character var	character var	character var	character var	character var	character var	character var	character var	character var	character var	character var	character varying(30)	set (pl_line_d	character var	character var	character var	character var
1	'135792468'	'ABC'	'CLOTHING'	'135792468'	'MA23456'	'MAIL'	'AS1234'	'ADULT T-SHI	'2004-07-27'	'8'	'10.5'	NULL	NULL	'PETER CHAN'	'L'	'3000'	'12.3'

Figure 17 Transformed object-oriented database

(c) Transform from flattened XML OUDG into Network database

Example 4: Transform from flattened XML OUDG into Network database

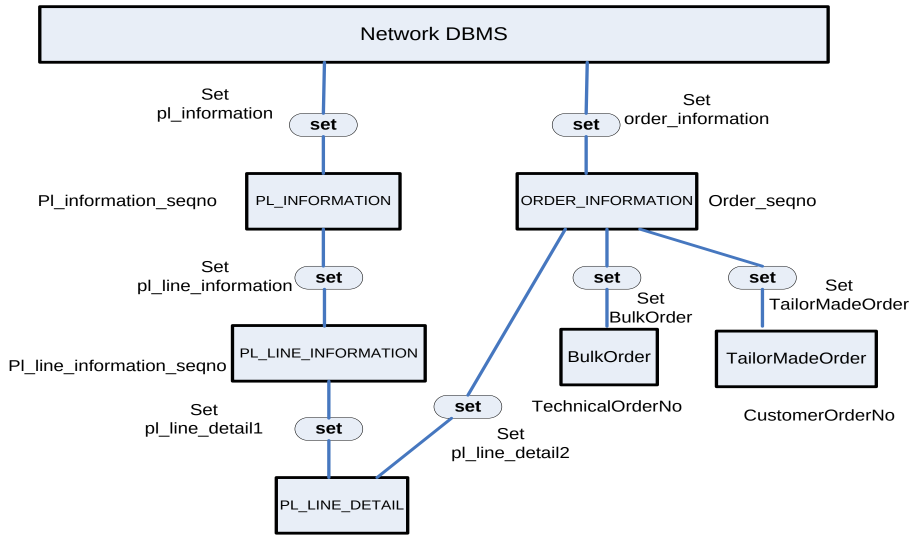


Figure 18 Translated legacy network database schema in Network Graph

In figure 18, record Pl_informations and Order_information are under network DBMS as first records for database navigation access path. The path can go from record Pl_information to record Pl_line_information in owner and member record in 1:n relationship. Records Pl_line_information (owner), Order_information(owner) and Pl_line_detail (member) are in flex structure such that records Pl_line_information and Order_information they are in m:n relationship. Records Order_information and BulkOrder are in 1:1 relationship. Similarly, records Order_information and TailorMadeOrder are in 1:1 relationship. The set records are pointers only.

There are six records. Each record class has key. The member record key determines its owner record key in FD, and subclass record key is a subset of its superclass record key as follows:

FD_1: pl_line_informtion. PL_INFORMATION_SEQNO → pl_information. PL_INFORMATION_SEQNO

FD_1 represent PL_INFORMAION and PL_Line_information are in one-to-many cardinality. FD_1 in NDB is same as FD_1 in flattened XML schema source. Therefore, one-to-many cardinality between PL_INFORMAION and PL_Line_information is preserved.

ID_1: BulkOrder.ORDER_NUMBER $\sqsubseteq$ order_information. ORDER_NUMBER

ID_1 represent BulkOrder and ORDER_INFORMATION are in ISA relationship. ID_1 in NDB is same as ID_1 in flattened XML schema source. Therefore, ISA relationship between BulkOrder and ORDER_INFORMATION is preserved.

ID_2: TailorMadeOrder.ORDER_NUMBER $\subseteq$ order_information. ORDER_NUMBER

ID_2 represent TailorMadeOrder and ORDER_INFORMATION are in ISA relationship. ID_2 in NDB is same as ID_2 in flattened XML schema source. Therefore, ISA relationship between TailorMadeOrder and ORDER_INFORMATION is preserved.

MVD_1: *PL_Line_information*. PL_INFORMATION_SEQNO →→ *Order_information* . *Order_NUMBER*

MVD_2: *Order_information* . *Order_NUMBER* →→ *PL_Line_information*. PL_INFORMATION_SEQNO

Therefore: MVD_1 and MVD_2 represent PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. Therefore, MVD_1 and MVD_2 in NDB is same as MVD_1 and MVD_2 in flattened XML schema source, many-to-many cardinality between TailorMadeOrder and ORDER_INFORMATION is preserved.

(Note: 2 one-to-many cardinalities are equivalent to many-to-many cardinality)

The transformed network database records are:

```
C:\Raima\RDM\11.0\win32\bin>initdb NDB1

Database Initialization Utility
Raima Database Manager 11.0.1 Build 551 [3-20-2012] http://www.raima.com/
Copyright (c) 2012 Raima Inc., All rights reserved.

Document Root: C:\Raima\RDM\11.0\win32\bin\
Initialization of database: NDB1

WARNING: this will destroy contents of the following files:

   NDB1.000
   NDB1.001
   NDB1.002
   NDB1.003
   NDB1.004
   NDB1.005
   NDB1.006
   NDB1.k01
   NDB1.k02
   NDB1.k03
   NDB1.k04
   NDB1.k05
   NDB1.k06

continue? (y/n) y

NDB1 initialized


C:\Raima\RDM\11.0\win32\bin>dbimp NDB1.imp

Database Import Utility
Raima Database Manager 11.0.1 Build 551 [3-20-2012] http://www.raima.com/
Copyright (c) 2012 Raima Inc., All rights reserved.

Document Root: C:\Raima\RDM\11.0\win32\bin\
Compilation complete:

Starting data import
[pl_information.csv:000001] '2004-08-03','S','TRAIN','2004-07-31','2004-08-02','2004-08-03','JOEY','SOME GOODS ARE BREAK
ABLE','EFG123DS'
[pl_line_information.csv:000001] 'ABCH439WS','13','7','BAG','10','13','EFG123DS'
[pl_line_information.csv:000002] 'ABCV234F','20','20','BOX','20','40','EFG123DS'
[pl_line_detail.csv:000001] '2','ABCH439WS','6','4','5','1','135792468','12','EFG123DS'
[pl_line_detail.csv:000002] '4','ABCV234F','6','1','8','2','123469999','28','EFG123DS'
[pl_line_detail.csv:000003] '2','ABCV234F','6','4','5','1','135792468','12','EFG123DS'
[order_information.csv:000001] '1399.0','TOYS','2004-07-29','123456999','PHONE','PS2','PLAYSTATION','MA23456','DONY','10
','123469999','1'
[order_information.csv:000002] '10.5','CLOTHING','2004-07-27','135792468','MAIL','AS1234','ADULT T-SHIRT SIZE M','MA2345
6','ABC','10','135792468','8'
[bulkorder.csv:000001] '10.5','2004-07-27','AMAZON','12.1','13592468','MAIL','123456789','S','8','CLOTHING','AS1234','AD
ULT T-SHIRT SIZE M','ABC','MA23456','5','2000'
[tailormadeorder.csv:000001] '1399','2004-07-29','Peter Chan','12.3','123456999','PHONE','123469999','L','1','TOYS','PS2
','PLAYSTATION','DONY','MA23456','10','3000'
Successful import
```

Figure 19 **Transformed Network database**

(d) Transform from flattened XML documents to relational database

Example 5: Transform from flattened XML documents to relational database

The transformed relational conceptual schema in Extended Entity Relationship model

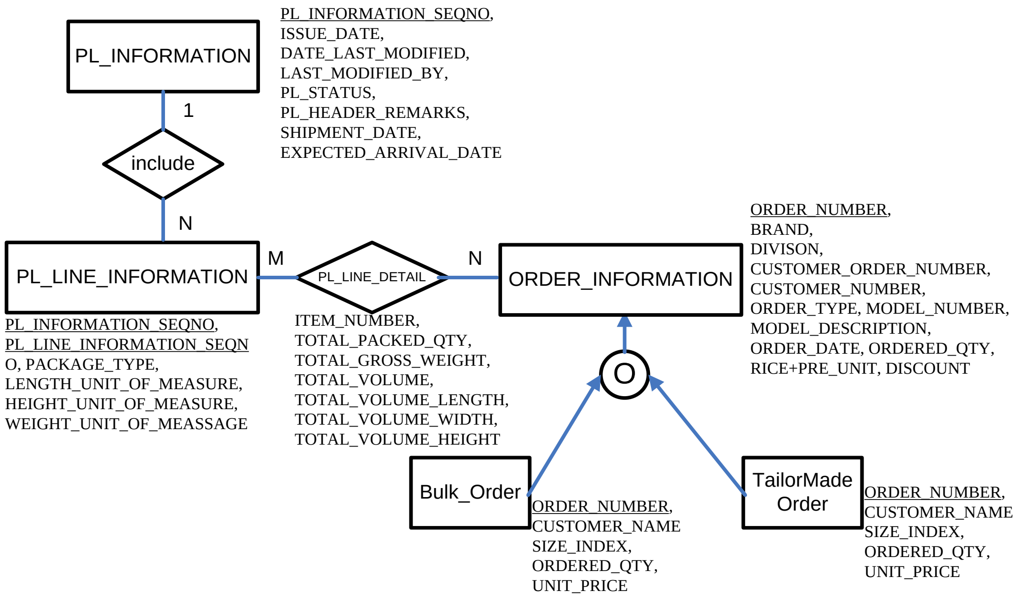


Figure 20 Transformed relational database in Extended Entity Relationship model

There are six tables. Each table has its primary key in italic, and foreign key prefixed with "*". Their data dependencies (DD) are such that each foreign key determines its referred primary key in FD, and subclass foreign key is a subset of its superclass primary key in ID as follows:

FD_1: *PL_Line_information*. PL_INFORMATION_SEQNO → *PL_information*.PL_INFORMATION_SEQNO

FD_1 represent PL_INFORMAION and PL_Line_information are in one-to-many cardinality. FD_1 in RDB is same as FD_1 in flattened XML schema source. Therefore, one-to-many cardinality between PL_INFORMAION and PL_Line_information is preserved.

ID_1: *Bulk_Order*. BulkOrder.ORDER_NUMBER $\sqsubseteq$ *Order_information* . *Order_NUMBER*

ID_1 represent subclass BulkOrder and superclass ORDER_INFORMATION are in ISA relationship. ID_1 in RDB is same as ID_1 in flattened XML schema source. Therefore, ISA relationship between BulkOrder and ORDER_INFORMATION is preserved.

ID_2: *TailorMadeOrder*.TailorMadeOrder.ORDER_NUMBER $\subseteq$ *Order_information* . *Order_NUMBER*

ID_2 represent subclass TailorMadeOrder and superclass ORDER_INFORMATION ae in ISA relationship. ID_2 in RDB is same as ID_2 in flattened XML schema source. Therefore, ISA relationship between TailorMadeOrder and ORDER_INFORMATION is preserved.

MVD_1: *PL_Line_information*. PL_INFORMATION_SEQNO $\rightarrow\rightarrow$ *Order_information* . *Order_NUMBER*

MVD_2: *Order_information*.*Order_NUMBER* $\rightarrow\rightarrow$ *PL_Line_information*. PL_INFORMATION_SEQNO

Therefore: MVD_1 and MVD_2 represent PL_Line_information and ORDER_INFORMATION are in many-to-many cardinality. Therefore, MVD_1 and MVD_2 in RDB is same as MVD_1 and MVD_2 in flattened XML schema source, many-to-many cardinality between TailorMadeOrder and ORDER_INFORMATION is preserved.

(Note: 2 one-to-many cardinalities are equivalent to many-to-many cardinality)

```
mysql> select * from pl_information;
+---------------+------------+--------------------+---------------------+-----------+---------------+-------------------+-----------------------+------------------+
| SHIPMENT_DATE | ISSUE_DATE | DATE_LAST_MODIFIED | PL_INFORMATION_SEQNO | PL_STATUS | SHIPMENT_TYPE | PL_HEADER_REMARKS | EXPECTED_ARRIVAL_DATE | LAST_MODIFIED_BY |
+---------------+------------+--------------------+---------------------+-----------+---------------+-------------------+-----------------------+------------------+
| 2004-08-03    | 2004-07-31 | 2004-08-02         | EFG123DS            | S         | TRAIN         | SOME GOODS        | 2004-08-03            | JOEY             |
+---------------+------------+--------------------+---------------------+-----------+---------------+-------------------+-----------------------+------------------+
1 row in set (0.00 sec)

mysql> select * from pl_line_information;
+----------------------+---------------------------+-----------------------+------------------------+--------------+------------------------+------------------------+
| PL_INFORMATION_SEQNO | PL_LINE_INFORMATION_SEQNO | WIDTH_UNIT_OF_MEASURE | HEIGHT_UNIT_OF_MEASURE | PACKAGE_TYPE | WEIGHT_UNIT_OF_MESSAGE | LENGTH_UNIT_OF_MEASURE |
+----------------------+---------------------------+-----------------------+------------------------+--------------+------------------------+------------------------+
| EFG123DS             | ABCH439WS                 | 13                    | 10                     | BAG          | 13                     | 7                      |
| EFG123DS             | ABCV234F                  | 20                    | 20                     | BOX          | 40                     | 20                     |
+----------------------+---------------------------+-----------------------+------------------------+--------------+------------------------+------------------------+
2 rows in set (0.00 sec)

mysql> select * from pl_line_detail;
+---------------------+----------------------+--------------------+--------------------+----------------------+--------------+------------------+---------------------------+-------------+
| TOTAL_VOLUME_LENGTH | PL_INFORMATION_SEQNO | TOTAL_GROSS_WEIGHT | TOTAL_VOLUME_WIDTH | TOTAL_VOLUMEN_HEIGHT | ORDER_NUMBER | TOTAL_PACKED_QTY | PL_LINE_INFORMATION_SEQNO | ITEM_NUMBER |
+---------------------+----------------------+--------------------+--------------------+----------------------+--------------+------------------+---------------------------+-------------+
| 5                   | EFG123DS             | 12                 | 2                  | 6                    | 135792468    | 4                | ABCH439WS                 | 1           |
| 8                   | EFG123DS             | 28                 | 4                  | 6                    | 123469999    | 1                | ABCV234F                  | 2           |
| 5                   | EFG123DS             | 12                 | 2                  | 6                    | 135792468    | 4                | ABCV234F                  | 1           |
+---------------------+----------------------+--------------------+--------------------+----------------------+--------------+------------------+---------------------------+-------------+
3 rows in set (0.00 sec)

mysql> select * from order_information;
+-----------------------+----------+-----------------+------------+-------+-------------------+--------------+--------------+------------+----------+------------+----------------+
| CUSTOMER_ORDER_NUMBER | DISCOUNT | CUSTOMER_NUMBER | ORDER_TPYE | BRAND | MODEL_DESCRIPTION | MODEL_NUMBER | ORDER_NUMBER | ORDERD_QTY | DIVISION | ORDER_DATE | PRICE_PER_UNIT |
+-----------------------+----------+-----------------+------------+-------+-------------------+--------------+--------------+------------+----------+------------+----------------+
| 123456999             | 10       | MA23456         | PHONE      | DONY  | PLAYSTATIO        | PS2          | 123469999    | 1          | TOYS     | 2004-07-29 | 1399.0         |
| 135792468             | 10       | MA23456         | MAIL       | ABC   | ADULT T-SH        | AS1234       | 135792468    | 8          | CLOTHING | 2004-07-27 | 10.5           |
+-----------------------+----------+-----------------+------------+-------+-------------------+--------------+--------------+------------+----------+------------+----------------+
2 rows in set (0.00 sec)

mysql> select * from bulk_order;
+---------------+------------+--------------+-------------+------------+
| CUSTOMER_NAME | UNIT_PRICE | ORDER_NUMBER | ORDERED_QTY | SIZE_INDEX |
+---------------+------------+--------------+-------------+------------+
| AMAZON        | 12.1       | 135792468    | 2000        | S          |
+---------------+------------+--------------+-------------+------------+
1 row in set (0.00 sec)

mysql> select * from tailor_made_order;
+---------------+------------+--------------+-------------+------------+
| CUSTOMER_NAME | UNIT_PRICE | ORDER_NUMBER | ORDERED_QTY | SIZE_INDEX |
+---------------+------------+--------------+-------------+------------+
| PETER CHAN    | 12.3       | 135792468    | 3000        | L          |
+---------------+------------+--------------+-------------+------------+
1 row in set (0.00 sec)
```

Figure 21 Transformed relational database

Performance Analysis

1) Performance system platform

To access the relative performance of the database legacy, we performed the OODB experiment in a VM installed on an IBM server (xSeries 335 / 8676) with Intel(R) Xeon(R) CPU X5650 with clock rate of 2.67 GHz, 2GB of main memory. The other experiments are performed on an IBM blade server with Intel(R) Xeon(R) CPU X5660 with clock rate of 2.80 GHz, 2GB of main memory. The operating system and DMBS using for the experiment are recorded in the table 4. The UDB software is written in Java 2.

2) DBMS for database

Table 4 DBMS table

	RDB source	Flattened XML	RDB	XML	OODB	NDB

Server OS	Window 7	Window 7	Window 7	Window 7	Window2000	Window 7
DBMS	MySQL	eXist	Oracle	eXist	UniSQL	Raima

3) Result in Diagram

First, we bulk load 400 record of Relational database source of the prototype OUDG. Then we measure the time for these 4 output database legacy for this bulk load in table 5.

Second, we query the data from one table from each database legacy. We measure the time and recorded in table 6.

Table 5 **Bulk load**

Dataset	RDB source	Flattened XML	RDB(Oracle)	XML	OODB	NDB
x400	27 sec	0.51 sec	2 sec	0.51 sec	0.5 sec	0.53 sec
x4000	180 sec	3 sec	3 sec	3 sec	5 sec	7 sec

Table 6 **Selection (Based on a condition, eg, Select bulk_order table)**

Dataset	RDB source	Flattened XML	RDB(Oracle)	XML	OODB	NDB
x400	4 sec	0.006 sec	1 sec	0.006 sec	0.5 sec	0.14 sec
x4000	30 sec	0.007 sec	0.7 sec	0.014 sec	5 sec	1 sec

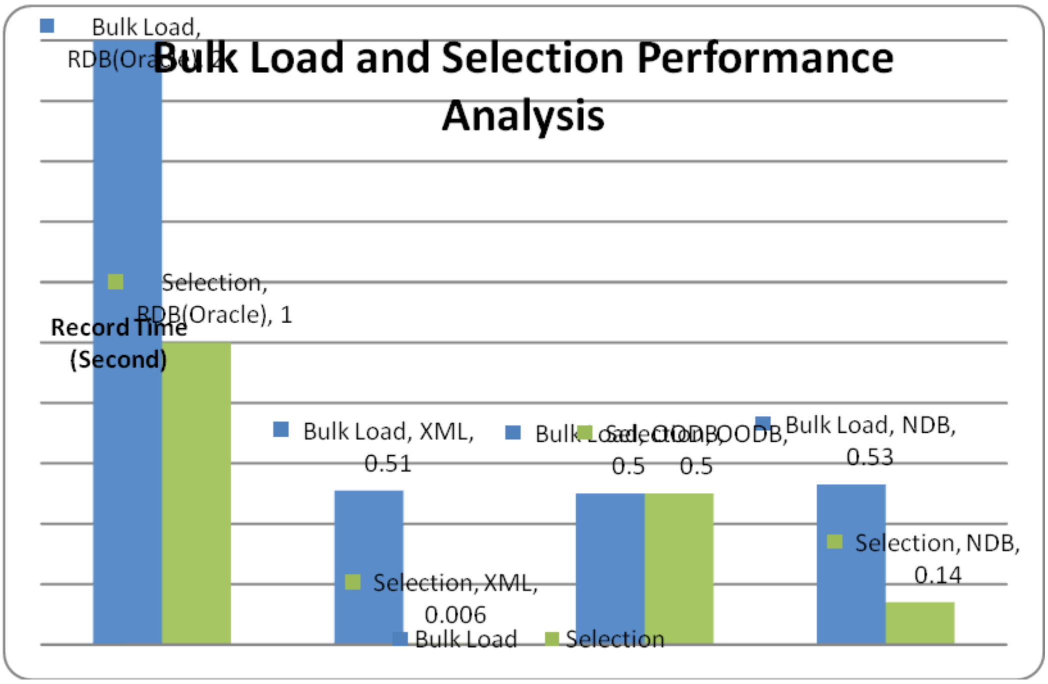


Figure 21 Performance analysis among legacy databases

Figure 21 compared the bulk load and selection performance analysis in 4 transformed legacy databases. The X axis represents the record time(second) while the Y axis represents 4 transformed legacy databases from OUDB. From the figure above, RDB is poorest in performance in both bulk load and selection while XML is the best for selection.

Result

In bulk load, the performance of OODB, NDB, XML are better than RDB, in the sequence of selection performance is XML > NDB > OODB> RDB. It is because XML are in Dom Tree structure which is the best for selection. RDB requires values matching, therefore its performance is poorest. NDB is pointer structure. Therefore it is better than OODB which requires table format and pointer structure.

As a result, we showed that it is valuable for user to transform the Relational database to other legacy databases by OUDG if the user wants to have a higher performance of their databases.

Chapter 6 Conclusion

Since relational database is the most user friendly legacy database, and XML database is the most portable database for information highway on the Internet. In this thesis, we offer Flattened XML database as a universal database such that it can be a user friendly database middleware for all legacy databases.

The contributions of this thesis are:

(1) The data models of legacy databases are compatible with each other for the preservation of their data semantic such as cardinality, ISA and generalization.

(2) The legacy databases can be reengineered into each other through flattened XML document such that a source legacy database can be transformed into a flattened XML document which can be further transformed into another target legacy database.

(3) The performance of OUDG (Open Universal Database Gateway) is acceptable through a prototype performance analysis.

(4) Use cloud computing: All of the legacy databases and the OUDB are developed in cloud platform.

The application of this thesis are:

(1) Openness of a universal database: The reason we choose flattened XML document is its openness, and DBMS independence. All other data models are DBMS dependent. Nevertheless, users can use OUDG to access any legacy database via flattened XML documents on the Internet through Internet Explorer without programming. Furthermore, an Oracle user can access an MS SQL Server database after transforming the Oracle database into flattened XML document, and then to MS SQL Server database by OUDG.

(2) Recovery of legacy database: Since flattened XML document is an information equivalent legacy database such that it can be used to recover any legacy database whenever the production legacy database is down. As a result, an equivalent XML document can be parallel processing with legacy database in non-stop computing as their backup copy.

(3) Heterogeneous databases integration for data warehousing: By transforming all in-house legacy databases into a common legacy database, companies can use OUDG to transform its heterogeneous databases into homogeneous databases, and integrate them into a logical view for data warehousing application.

(4) Portability of Flattened XML document as Universal database: The OUDG solution is not limited to using a particular DBMS, but also allows users to access any legacy database through OUDG, which is similar to ODBC.

In summary, the OUDG unites all legacy database data models into one data model of flattened XML schema. The portability of the proposed flattened XML document can be transferred into any open platform. The methodology of this OUDG is to download the raw data of source legacy database into flattened XML document according to source legacy database schema, and upload it into target database using translated target legacy database schema, which is a logical level approach to avoid physical data type conversion. Therefore, the methodology can transform any legacy database into any other legacy database. The reason of using flattened XML document as medium is to reduce the number of data conversion programs. Without OUDB, we need 4 * 4 = 16 programs. With OUDG, we need 4 + 4 = 8 programs for data conversion.

Above all, all legacy databases can be transformed into each other via flattened XML documents for data access in the same way as computers connect to each other via Internet for information retrieval.

Appendix shows the schema of prototype.

List of Publication

1. Joseph Fong, Kenneth Ting Yan Wong, and Tracy Wu, PTA System: Mobile Computing Student Assessment by Parent and Teacher Association: LNCS of Hybrid Learning, 5th International Conference, ICHL 2012
2. Joseph Fong, Kenneth Ting Yan Wong, Fu Lee Wang, Cheng Wing Tung and Titus Lo, Environmental Friendly Real Time Quiz Using Mobile Devices with Auto Marking: Selected Paper of Hybrid Learning, 5th International Conference, ICHL 2012
3. Joseph Fong, Kenneth Ting Yan Wong, Generating E-book System Using Cloud Computing: A Cognitive Map and Open Forum Approach: LNCS of Hybrid Learning, 5th International Conference, ICHL 2013
4. Joseph Fong, Kenneth Ting Yan Wong, A personal assistant authoring eBook for eLearning in Higher Education using Inverted Files of Hyperlinks, International Journal of Innovation and Learning 2013
5. Joseph Fong, Kenneth Ting Yan Wong, Brian Lam, Herbert Shiu, a pending patent on "**Cross model datum access with semantic preservation for legacy databases"**, 2014

Working Paper

There is a working paper on universal database, which has been accepted for publication by, Sixth International conference on Database Management Systems (DMS-2015).

The paper will be titled, cross model datum access with semantic preservation for legacy databases.

Reference

Chen, P. (1976), "The Entity-Relationship Model: Toward a Unified View of Data", ACM on Database Systems, Vol. 1, No. 1.

CODD, E. F. (1970), A Relational Model of Data for Large Shared Data Banks Communications of the ACM, Volume 13 Issue 6, Pages 377-387

eXist (2014) , XML DBMS. http://exist-db.org/exist/apps/homepage/index.html

Fong, J. (1991), "The Framework of Data and Transaction Translation in the Distributed Heterogeneous Database Management System: An Extended Entity Relationship Approach", Proceedings of the 10th South East Asia Regional Computer Confederation Conference, Bali, Indonesia, 4-6 December 1991, Section 47, pp. 1-26.

Fong, J. (1992), "Methodology for Schema Translation from Hierarchical or Network into Relational", ***Information and Software Technology***, *Volume 34, Number 3, pp. 159- 174.*

Fong, J and Bloor,C. (1994), "Data Conversion Rules from Network to Relational Databases", Information and Software Technology, Volume. 36 No. 3, pp. 141-154.

Fong, J. (1996), "Adding Relational Interface to Non-relational Database", **IEEE Software**, pp. 89-97.

Fong, J (1997), "Converting Relational to Object-Oriented Databases", SIGMOD RECORD, Volume 26, Number 1, pp53-58.

Fong, J. and Huang, S.M. (1999), "Architecture of a Universal Database: A Frame Model Approach", International Journal of Cooperative Information Systems, Volume 8, Number. 1, pp. 47-82.

Fong, J. (2001), "**Converting Relational Database into XML Documents with DOM",** Proceeding DEXA '01 Proceedings of the 12th International Workshop on Database and Expert Systems Applications, Page 61, IEEE Computer Society Washington, DC, USA.

Fong, J., Li, Q. and Huang, S.M. (2003a), "Universal Data Warehousing Based on a Meta-Data Modeling Approach", International Journal of Cooperative Information Systems, Volume 12, Number 3, pp.325-363.

Fong, J., Pang, R., Fong, A., Pang, F., and Poon, K. (2003), "Concurrent data materialization for object-relational database with semantic metadata", International Journal of Software Engineering and Knowledge Engineering, Volume 13, Number 3, pp.257-291.

Fong, J. (2004), "**XTOPO: An XML-based topology for information highway on the Internet**", Journal of Database management, Volume 15, Issue 3.

Fong, J. and San Kuen Cheung (2005),"Translating relational schema into XML schema definition with data semantic preservation and XSD graph", Information and Software Technology, Volume 47, pp.437-462

Fong, J. (2006), "Information Systems Reengineering and Integration", Springer Verlag, ISBN 1-84628-382-5, 370 pages.

Fong, J., Shiu, H., Wong, J. (2009), "Methodology for data conversion from XML documents to relations using Extensible Stylesheet Language Transformation"**, International Journal of Software Engineering and Knowledge Engineering,** Volume 19, Number 2, pp. 249-281

Fong, J and Shiu, H. (2012), "An interpreter approach for exporting relational data into XML documents with Structured Export Markup Language' Journal of Database Management, volume 23, issue 1.

Funderburk, J. E. et al. (2002), "XTABLES: Bridging Relational Technology and XML", IBM Systems Journal, Vol 41, No. 4, PP 616 –641.

Guardalben, G. (2004). Integrating XML and Relational Database Technologies: A Position Paper. Retrieved May 1st,2005, from http://www.hitsw.com/products_services/whitepapers/integrating_xml_rdb/integrating_xml_white_paper.pdf

Gilmore, W.J. (2000), Introduction to Database Normalization (MySQL database) http://www.devshed.com/c/a/mysql/an-introduction-to-database-normalization/

Harris, D. (2012), "cloud-databases-101-who-builds-em-and-what-they-do", GIGAOM, http://gigaom.com/2012/07/20/cloud-databases-101-who-builds-em-and-what-they-do/

Hsiao, D.K. and Kamel, M.N.(1989), "Heterogeneous Databases: Proliferations, Issues, and Solutions", IEEE Transactions on Knowledge and Data Engineering, Voumn 1, Np. 1. Pp.45-62.

Janssen, C., Entity-Relationship Diagram (2014), http://www.techopedia.com/definition/1200/entity-relationship-diagram-erd

Kanagaraj, S. and Sunitha, A. (2012),"Converting Relational Database Into Xml Document", International Journal of Computer Science Issues, Vol.9, Issue 2, No 1

Lee, D., Mani, M and Chu, W. W. (2002), "Effective Schema Conversions between XML and Relational Models" In: Proceedings of the European Conference on Artificial Intelligence and the Knowledge Transformation Workshop, Lyon, France

Lee, D., Mani, M and Chu, W. W. (2012), "Schema Conversion Methods between XML and Relational Models", International Journal of Computer Science Issues, Vol. 9, Issue 2, No 1, March 2012

Lum, V.Y., Shu, N.C. and Housel, B.C. (1976), "A General Methodology for Data Conversion and Restructuring", IBM Journal of research and development, Volume 20, Issue 5, pp.483-497.

Navathe, S. and Awong, A. (1998), Abstracting relational and hierarchical data with a semantic data model, Entity-Relationship Approach, p305-333.

Object-Oriented database users (2014)
http://www.objectivity.com/
http://www.gemstone.com/

ODBC(2014), http://en.wikipedia.org/wiki/Open_Database_Connectivity

Oracle (2014), Relational database software.

http://www.oracle.com/us/products/database/overview/index.html

Raima (2014), Network model database software. http://raima.com/

Raima users(2014), http://raima.com/customers/

Ramez,E., Shamkant, B.(2011), "Database Systems, Models, Languages, Design , and Application programming", Pearson, 6th edition, P.56

Relational database users (2014), https://www.oracle.com/search/customers/

Rhoton, J. and Haukioja. R., (2013), Cloud Computing Architected,
ISBN 978-0-9563556-1-4, Recursive Press, P163 - P183, 360 pages

Sellis, T., Lin, C.C. and Raschid, L. (1993), "Coupling Production Systems and Database Systems: A Homogeneous Approach", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 5, NO. 2.

Shoshani, A.(1975), "A Logical-Level Approach to Data Base Conversion", ACM SGMOD International Conference on Management of Data, pp.112-122.

Silverston, L. and Graziano, K. (2008)
www.360doc.com/content/08/0830/01/1032_1590731.shtml

XML as web service (2014), http://www.altova.com/downloadxmltools.html

XML export and import (2014),
http://wiki.servicenow.com/index.php?title=Exporting_and_Importing_XML_Files)

XML users (2014), https://tomcat.apache.org/tomcat-3.3-doc/serverxml.html

Wang, S.P., Ledley, R.S. (2013), Computer Architecture and Security Fundamentals of Designing Secure Computer Systems ISBN: 978-1-118-16881-3, 321 Pages, P203-208.

Appendix Schema of Prototype

The input Relational database schema is*:*
(primary keys are underlined, foreign keys are prefixed with "*")

PL_ INFORMATION (PL_INFORMATION_SEQNO, ISSUE_DATE, DATE_LAST_MODIFIED, LAST_MODIFIED_BY, PL_STATUS, PL_HEADER_REMARKS, SHIPMENT_DATE, EXPECTED_ARRIVAL_DATE)

PL_LINE_INFORMATION (*PL_INFORMATION_SEQNO, PL_LINE_INFORMATION_SEQNO, PACKAGE_TYPE, LENGTH_UNIT_OF_MEASURE, HEIGHT_UNIT_OF_MEASURE, WEIGHT_UNIT_OF_MEASSAGE)

PL_LINE_DETAIL (*PL_INFORMATION_SEQNO, *PL_LINE_INFORMATION_SEQNO, *ORDER_NUMBER, ITEM_NUMBER, TOTAL_PACKED_QTY, TOTAL_GROSS_WEIGHT, TOTAL_VOLUME, TOTAL_VOLUME_LENGTH, TOTAL_VOLUME_WIDTH, TOTAL_VOLUME_HEIGHT)

ORDER_INFORMATION (ORDER_NUMBER, BRAND, DIVISON, CUSTOMER_ORDER_NUMBER, CUSTOMER_NUMBER, ORDER_TYPE, MODEL_NUMBER, MODEL_DESCRIPTION, ORDER_DATE, ORDERED_QTY, RICE+PRE_UNIT, DISCOUNT)

BULKORDER(*ORDER_NUMBER, CUSTOMER_NAME, SIZE_INDEX, ORDERED_QTY, UNIT_PRICE)

TAILORMADEORDER(*ORDER_NUMBER, CUSTOMER_NAME, SIZE_INDEX, ORDERED_QTY , UNIT_PRICE)

The flattened XML document schema is:

```
<!ELEMENT db-casestudy (PL_INFORMATION, PL_LINE_INFORMATION,
PL_LINE_DETAIL, ORDER_INFORMATION, BULKORDER, TAILORMADEORDER)>
<!ELEMENT PL_INFORMATION EMPTY>
<!ATTLIST      PL_INFORMATION
      t-pl_information.c-PL_INFORMATION_SEQNO ID    #REQUIRED
      PL_INFORMATION_SEQNO CDATA #REQUIRED
          ISSUE_DATE CDATA #REQUIRED
        DATE_LAST_MODIFIED CDATA #REQUIRED
        LAST_MODIFIED_BY CDATA #REQUIRED
        PL_STATUS CDATA #REQUIRED
        PL_HEADER_REMARKS CDATA #REQUIRED
        SHIPMENT_DATE CDATA #REQUIRED
        EXPECTED_ARRIVAL_DATE CDATA #REQUIRED>
<!ELEMENT PL_LINE_INFORMATION EMPTY>
<!ATTLIST      PL_LINE_INFORMATION
      t-pl_information.c-PL_INFORMATION_SEQNO IDREF    IMPLIED
        t-pl_line_information.c-PL_INFORMATION_SEQNO.c-PL_LINE_INFORMATION
_SEQNO    ID    #REQUIRED
       PL_INFORMATION_SEQNO CDATA #REQUIRED
        PL_LINE_INFORMATION_SEQNO CDATA #REQUIRED
        PACKAGE_TYPE CDATA #REQUIRED
        LENGTH_UNIT_OF_MEASURE CDATA #REQUIRED
        WIDTH_UNIT_OF_MEASURE CDATA #REQUIRED
        HEIGHT_UNIT_OF_MEASURE CDATA #REQUIRED
        WEIGHT_UNIT_OF_MEASURE CDATA #REQUIRED>
<!ELEMENT PL_LINE_DETAIL EMPTY>
<!ATTLIST      PL_LINE_DETAIL
        t-pl_line_information.c-PL_INFORMATION_SEQNO.c-PL_LINE_INFORMATION
_SEQNO IDREF    IMPLIED
      t-order_information.c-ORDER_NUMBER IDREF    IMPLIED
      PL_INFORMATION_SEQNO CDATA #REQUIRED
        PL_LINE_INFORMATION_SEQNO CDATA #REQUIRED
        ORDER_NUMBER CDATA #REQUIRED
        ITEM_NUMBER CDATA #REQUIRED
        TOTAL_GROSS_WEIGHT CDATA #REQUIRED
        TOTAL_VOLUME_WIDTH CDATA #REQUIRED
        TOTAL_VOLUME_HEIGHT CDATA #REQUIRED>
```

```
<!ELEMENT ORDER_INFORMATION EMPTY>
<!ATTLIST     ORDER_INFORMATION
        t-order_information.c-ORDER_NUMBER ID   #REQUIRED
    ORDER_NUMBER CDATA #REQUIRED
      BRAND CDATA #REQUIRED
      DIVISION CDATA #REQUIRED
      ORDER_TYPE CDATA #REQUIRED
      MODEL_NUMBER CDATA #REQUIRED
      MODEL_DESCRIPTION CDATA #REQUIRED
    ORDER_DATE CDATA #REQUIRED
      ORDERED_QTY CDATA #REQUIRED
      PRICE_PRE_UNIT CDATA #REQUIRED
      DISCOUNT CDATA #REQUIRED>
<!ELEMENT BULKORDER EMPTY>
<!ATTLIST     BULKORDER
        t-order_information.c-ORDER_NUMBER IDREF   IMPLIED
    ORDER_INFORMATION_ID CDATA #REQUIRED
    CUSTOMER_NAME CDATA #REQUIRED
      SIZE_INDEX CDATA #REQUIRED
      ORDERED_QTY CDATA #REQUIRED
      UNIT_PRICE CDATA #REQUIRED>
<!ELEMENT TAILORMADEORDER EMPTY>
<!ATTLIST     TAILORMADEORDER
        t-order_information.c-ORDER_NUMBER IDREF   IMPLIED
    CUSTOMER_NAME CDATA #REQUIRED
      SIZE_INDEX CDATA #REQUIRED
      ORDERED_QTY CDATA #REQUIRED
      UNIT_PRICE CDATA #REQUIRED>
</ORDER>
```

The target XML database schema is:

```
<!ELEMENT db-casestudy (PL_INFORMATION, ORDER_INFORMATION)>
<!ELEMENT PL_INFORMATION (PL_LINE_INFORMATION*)>
<!ATTLIST    PL_INFORMATION
    PL_INFORMATION_SEQNO CDATA #REQUIRED
       ISSUE_DATE CDATA #REQUIRED
      DATE_LAST_MODIFIED CDATA #REQUIRED
      LAST_MODIFIED_BY CDATA #REQUIRED
      PL_STATUS CDATA #REQUIRED
      PL_HEADER_REMARKS CDATA #REQUIRED
      SHIPMENT_DATE CDATA #REQUIRED
      EXPECTED_ARRIVAL_DATE CDATA #REQUIRED>
<!ELEMENT PL_LINE_INFORMATION (PL_LINE_DETAIL)>
<!ATTLIST    PL_LINE_INFORMATION
    PL_INFORMATION_SEQNO CDATA #REQUIRED
      PL_LINE_INFORMATION_SEQNO CDATA #REQUIRED
      PACKAGE_TYPE CDATA #REQUIRED
      LENGTH_UNIT_OF_MEASURE CDATA #REQUIRED
      WIDTH_UNIT_OF_MEASURE CDATA #REQUIRED
      HEIGHT_UNIT_OF_MEASURE CDATA #REQUIRED
      WEIGHT_UNIT_OF_MEASURE CDATA #REQUIRED>
<!ELEMENT PL_LINE_DETAIL EMPTY>
<!ATTLIST    PL_LINE_DETAIL
      idref1   IDREF   IMPLIED
    PL_INFORMATION_SEQNO CDATA #REQUIRED
      PL_LINE_INFORMATION_SEQNO CDATA #REQUIRED
      ORDER_NUMBER CDATA #REQUIRED
      ITEM_NUMBER CDATA #REQUIRED
      TOTAL_GROSS_WEIGHT CDATA #REQUIRED
      TOTAL_VOLUME_WIDTH CDATA #REQUIRED
      TOTAL_VOLUME_HEIGHT CDATA #REQUIRED>
<!ELEMENT ORDER_INFORMATION (BULKORDER, TAILORMADEORDER)>
<!ATTLIST    ORDER_INFORMATION
      id1   ID   #REQUIRED
    ORDER_NUMBER CDATA #REQUIRED
      BRAND CDATA #REQUIRED
```

```
        DIVISION CDATA #REQUIRED
        ORDER_TYPE CDATA #REQUIRED
        MODEL_NUMBER CDATA #REQUIRED
        MODEL_DESCRIPTION CDATA #REQUIRED
    ORDER_DATE CDATA #REQUIRED
        ORDERED_QTY CDATA #REQUIRED
        PRICE_PRE_UNIT CDATA #REQUIRED
        DISCOUNT CDATA #REQUIRED>
<!ELEMENT BULKORDER EMPTY>
<!ATTLIST       BULKORDER
    CUSTOMER_NAME CDATA #REQUIRED
        SIZE_INDEX CDATA #REQUIRED
        ORDERED_QTY CDATA #REQUIRED
        UNIT_PRICE CDATA #REQUIRED>

<!ELEMENT TAILORMADEORDER EMPTY>

<!ATTLIST       TAILORMADEORDER
    CUSTOMER_NAME CDATA #REQUIRED
        SIZE_INDEX CDATA #REQUIRED
        ORDERED_QTY CDATA #REQUIRED
        UNIT_PRICE CDATA #REQUIRED>
</ORDER>
```

The target Object-Oriented database schema is:

```
create class PL_INFORMATION;
CREATE class PL_LINE_DETAIL;
Create class ORDER_INFORMATION;

create class PL_LINE_INFORMATION
(       PL_INFROAMATION_SEQNO varchar(20),
        PL_LINE_INFORMATOIN_SEQNO varchar(20),
        PACKAGE_TYPE varchar(20),
        LENGTH_UNIT_OF_MEASURE varchar(20),
        WIDTH_UNIT_OF_MEASURE varchar(20),
        HEIGHT_UNIT_OF_MEASURE varchar(20),
        WEIGHT_UNIT_OF_MESSAGE varchar(20),
        PL_LINE_ass2 set of (PL_LINE_DETAIL),
                PL_LINE_ass PL_INFORMATION);

alter class PL_INFORMATION
Add attribute PL_INFORMATION_SEQNO varchar(20),
        ISSUE_DATE varchar(20),
        DATA_LAST_MODIFIED varchar(20),
        LAST_MODIFIED_BY varchar(20),
        PL_STATUS varchar(2),
        PL_HEADER_REMARKS varchar(40),
        SHIPMENT_TYPE varchar(20),
        SHIPMENT_DATE   varchar(20),
        EXPERCTED_ARRIVAL_DATE   varchar(20),
         PL_as set of (PL_LINE_INFORMATION);

alter class PL_LINE_DETAIL
Add attribute   ORDER_NUMBER integer,
                PL_INFORMATION_SEQNO varchar(20),
              PL_LINE_INFORMATION_SEQNO varchar(20),
                ORDER_NUMBER varchar(20),
                ITEM_NUMBER varchar(20),
                TOTAL_PACKED_QTY varchar(20),
                TOTAL_GROSS_WEIGHT varchar(20),
                TOTAL_VOLUME_LENGTH varchar(20),
                TOTAL_VOLUME_WIDTH varchar(20),
                TOTAL_VOLUMEN_HEIGHT varchar(20),
                LINE_DETAIL_ass PL_LINE_INFORMATION,
                LINE_DETAIL_ass2 ORDER_INFORMATION;

alter class ORDER_INFORMATION
Add attribute ORDER_NUMBER varchar(20),
        BRAND varchar(20),
        DIVISION varchar(20),
        CUSTOMER_ORDER_NUMBER varchar(20),
        CUSTOMER_NUMBER varchar(20),
        ORDER_TPYE varchar(20),
        MODEL_NUMBER varchar(20),
        MODEL_DESCRIPTION varchar(40),
        ORDER_DATE varchar(20),
        ORDERD_QTY varchar(20),
        PRICE_PRE_UNIT varchar(20),
        DISCOUNT varchar(20),
        ORDER_INFO_as set of (PL_LINE_DETAIL);

create class BulkOrder as subclass of ORDER_INFORMATION
(       CUSTOMER_NAME varchar(20),
        SIZE_INDEX varchar(20),
        ORDERED_QTY varchar(20),
        UNIT_PRICE varchar(20),);

create class TailorMadeOrder as subclass of
ORDER_INFORMATION
(       CUSTOMER_NAME varchar(20),
        SIZE_INDEX varchar(20),
        ORDERED_QTY varchar(20),
        UNIT_PRICE varchar(20));
```

The target Network database schema is:

```
database NDB1 {
      data file "NDB1.000" contains Network_DBMS;
      data file "NDB1.001" contains PL_INFORMATION;
      data file "NDB1.002" contains
PL_LINE_INFORMATION;
      data file "NDB1.003" contains PL_LINE_DETAIL;
      data file "NDB1.004" contains
ORDER_INFORMATION;
      data file "NDB1.005" contains BULKORDER;
      data file "NDB1.006" contains TAILORMADEORDER;
      key file "NDB1.k01" contains Pl_information_seqno;
      key file "NDB1.k02" contains A;
      key file "NDB1.k03" contains B;
      key file "NDB1.k04" contains C;
      key file "NDB1.k05" contains D;
      key file "NDB1.k06" contains E;

      record Network_DBMS {      }
      record PL_INFORMATION {
char SHIPMENT_DATE[31];
char PL_STATUS[31];
char SHIPMENT_TYPE[31];
char ISSUE_DATE[31];
char DATE_LAST_MODIFIED[31];
char EXPECTED_ARRIVAL_DATE[31];
char LAST_MODIFIED_BY[31];
char PL_HEADER_REMARKS[31];
key char Pl_information_seqno[31];
      }
       record PL_LINE_INFORMATION {
char WIDTH_UNIT_OF_MEASURE[31];
char LENGTH_UNIT_OF_MEASURE[31];
char PACKAGE_TYPE[31];
char HEIGHT_UNIT_OF_MEASURE[31];
char PL_LINE_INFORMATOIN_SEQNO[31];
char WEIGHT_UNIT_OF_MESSAGE[31];
char PL_INFORMATOIN_SEQNO[31];
compound key A {
PL_INFORMATOIN_SEQNO;
PL_LINE_INFORMATOIN_SEQNO; }
}
  record PL_LINE_DETAIL {
char TOTAL_VOLUME_WIDTH[31];
char PL_LINE_INFORMATION_SEQNO[31];
char TOTAL_VOLUMEN_HEIGHT[31];
char TOTAL_PACKED_QTY[31];
char TOTAL_VOLUME_LENGTH[31];
char ITEM_NUMBER[31];
char ORDER_NUMBER[31];
char TOTAL_GROSS_WEIGHT[31];
char PL_INFORMATION_SEQNO[31];

compound key B {
PL_INFORMATION_SEQNO;
PL_LINE_INFORMATION_SEQNO;
ORDER_NUMBER;    }
}
record ORDER_INFORMATION{
char PRICE_PRE_UNIT[31];
char DIVISION[31];
char ORDER_DATE[31];
char CUSTOMER_ORDER_NUMBER[31];
char ORDER_TPYE[31];
char MODEL_NUMBER[31];
char MODEL_DESCRIPTION[31];
char CUSTOMER_NUMBER[31];
char BRAND[31];
char DISCOUNT[31];
char ORDER_NUMBER[31];
char ORDERD_QTY[31];

compound key C {
```

```
ORDER_NUMBER; }
}
record BULKORDER{
char PRICE_PRE_UNIT[31];
char ORDER_DATE[31];
char CUSTOMER_NAME[31];
char UNIT_PRICE[31];
char CUSTOMER_ORDER_NUMBER[31];
char ORDER_TPYE[31];
char ORDER_NUMBER[31];
char SIZE_INDEX[31];
char ORDERED_QTY[31];
char DIVISION[31];
char MODEL_NUMBER[31];
char MODEL_DESCRIPTION[31];
char BRAND[31];
char CUSTOMER_NUMBER[31];
char DISCOUNT[31];
char ORDERD_QTY[31];

compound key D {
ORDER_NUMBER;    }
}
record TAILORMADEORDER{
char PRICE_PRE_UNIT[31];
char ORDER_DATE[31];
char CUSTOMER_NAME[31];
char UNIT_PRICE[31];
char CUSTOMER_ORDER_NUMBER[31];
char ORDER_TPYE[31];
char ORDER_NUMBER[31];
char SIZE_INDEX[31];
char ORDERED_QTY[31];
char DIVISION[31];
char MODEL_NUMBER[31];
char MODEL_DESCRIPTION[31];
char BRAND[31];
char CUSTOMER_NUMBER[31];
char DISCOUNT[31];
char ORDERD_QTY[31];
compound key E {
ORDER_NUMBER;    }
}
        set pl_information {
        order last;
        owner Network_DBMS;
        member PL_INFORMATION;
        }
        set pl_line_information {
        order last;
        owner PL_INFORMATION;
        member PL_LINE_INFORMATION;
        }
        set pl_line_detail1 {
        order last;
        owner PL_LINE_INFORMATION;
        member PL_LINE_DETAIL;
        }
        set order_information {
        order last;
        owner Network_DBMS;
        member ORDER_INFORMATION;
        }
        set pl_line_detail2 {
        order last;
        owner ORDER_INFORMATION;
        member PL_LINE_DETAIL;
        }
        set BulkOrder {
        order last;
        owner ORDER_INFORMATION;
        member BULKORDER;
        }
        set TailorMadeOrder {
        order last;
        owner ORDER_INFORMATION;
```

```
member TAILORMADEORDER;                    }    }
```