An open universal database gateway for non-stop Internet Computing with Replicate Flattened XML Documents

Joseph Fong, and Kenneth Wong
Computer Science Department, City University of Hong Kong, Hong Kong, email: csjfong@cityu.edu.hk

Abstract
The DBMS of various data models have proliferated into many companies, and become their legacy databases. However, there is a need to access these legacy databases for mass information transmission on the Internet for e-commerce. Most XML database management systems can translate certain legacy databases into an XML(Extensible Model Language) document. However, the translation is without data semantics constraints consideration for all legacy databases, which may not be sufficient to meet users' requirements. Furthermore, the users may prefer to keep two production database systems such that a legacy database is for internal data processing and a replicate XML document is for external Internet computing. This paper presents an open universal database gateway (OUDG) which can capture the major data semantics of cardinality, isa, generalization and aggregation of a sender company's legacy database into a flattened XML document, and transmit and store it into a receiver company's legacy database on thea Internet. OUDG transforms legacy databases to flattened XML documents and then to receiver's legacy database. As a result, users can access each other's legacy database through flattened XML document, which let users apply their own familiar query language to access their partners' legacy databases. For example, an user can use SQL (relational database language) to access object-oriented, network, or XML databases. Similarly, an user can use OQL (object-oriented database language), IDMS (network database language) and XQuery (XML database language) to access other legacy databases. Furthermore, since flattened XML documents is a replication of sender's legacy database, therefore, the senders can use legacy database for internal data processing, and its replicate flattened XML documents for e-commerce Internet computing, in parallel non-stop processing.

Keywords: open universal database gateway, legacy databases, flattened XML Documents, e-commerce, data semantics, information highway

1 Introduction

The evolution of database technologies intend 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 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. Firstly, legacy database can be transformed into flattened XML documents which can be further transformed into four legacy databases 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 structure of the flattened XML document data file is a relational table structured XML document. It has XML document syntax with relational table structure. It replaces primary key with ID, and foreign key with 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 (tableN) determines its type name and the name of property (attributeN) 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 the ID in other tables in the same flattened XML document instance. 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. The OUDG can transform legacy databases into flattened XML document, and then further transform the flattened XML document into one of four receiver's legacy databases: relational, object-oriented, XML and network. The result is that OUDG allows users choose a legacy data model to receive the sender's legacy databases with data semantics preservation.

This paper offers flattened XML documents as 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 in this paper because they are all in tree structure. All legacy data models can be united into flattened XML document as shown in Figure 1.

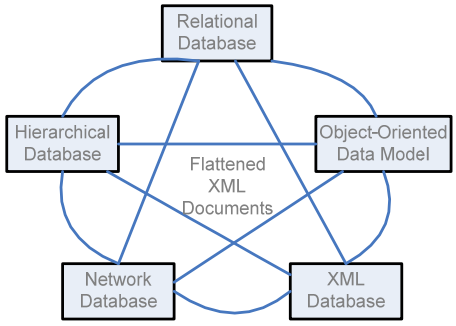


Figure 1 Legacy Databases Interoperability via Flattened XML documents

In Figure 1, proprietary legacy databases can be transformed into flattened XML documents as universal database. For example, in Figure 2, OUDG transforms a network database into a flattened XML document and then to a relational database for e-commerce on the Internet. In this case, receiver can access a network database as a relational database by using a Relational DBMS via OUDG.

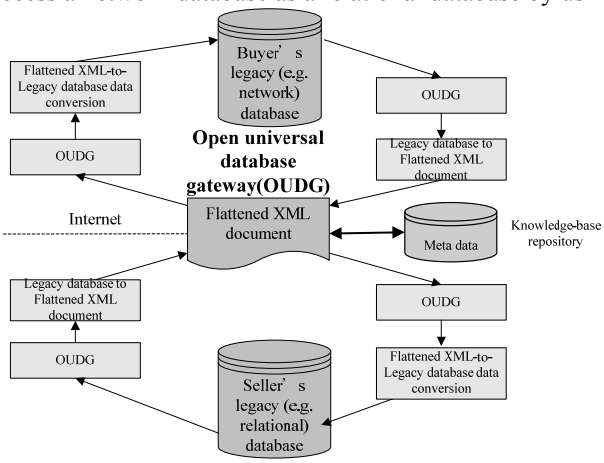


Figure 2 Replicate Flattened XML Documents for Information Highway on the Internet

An e-commerce company uses OUDG to convert its production legacy databases of relational, network, object-oriented and XML into flattened XML documents, and send them to its e-commerce partner on the Internet. The partner receives flattened XML document on the Internet and uses OUDG to convert the flattened XML documents into the receiver's legacy databases for storage. Then the

partner repeats the operations by converting its legacy databases into flattened XML documents and send them back to the company who uses OUDG to convert the flattened XML documents back into legacy databases for storage.

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 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.

Figure 3 shows the architecture of an open universal database gateway which transforms legacy databases into each other with different data models via flattened XML document:

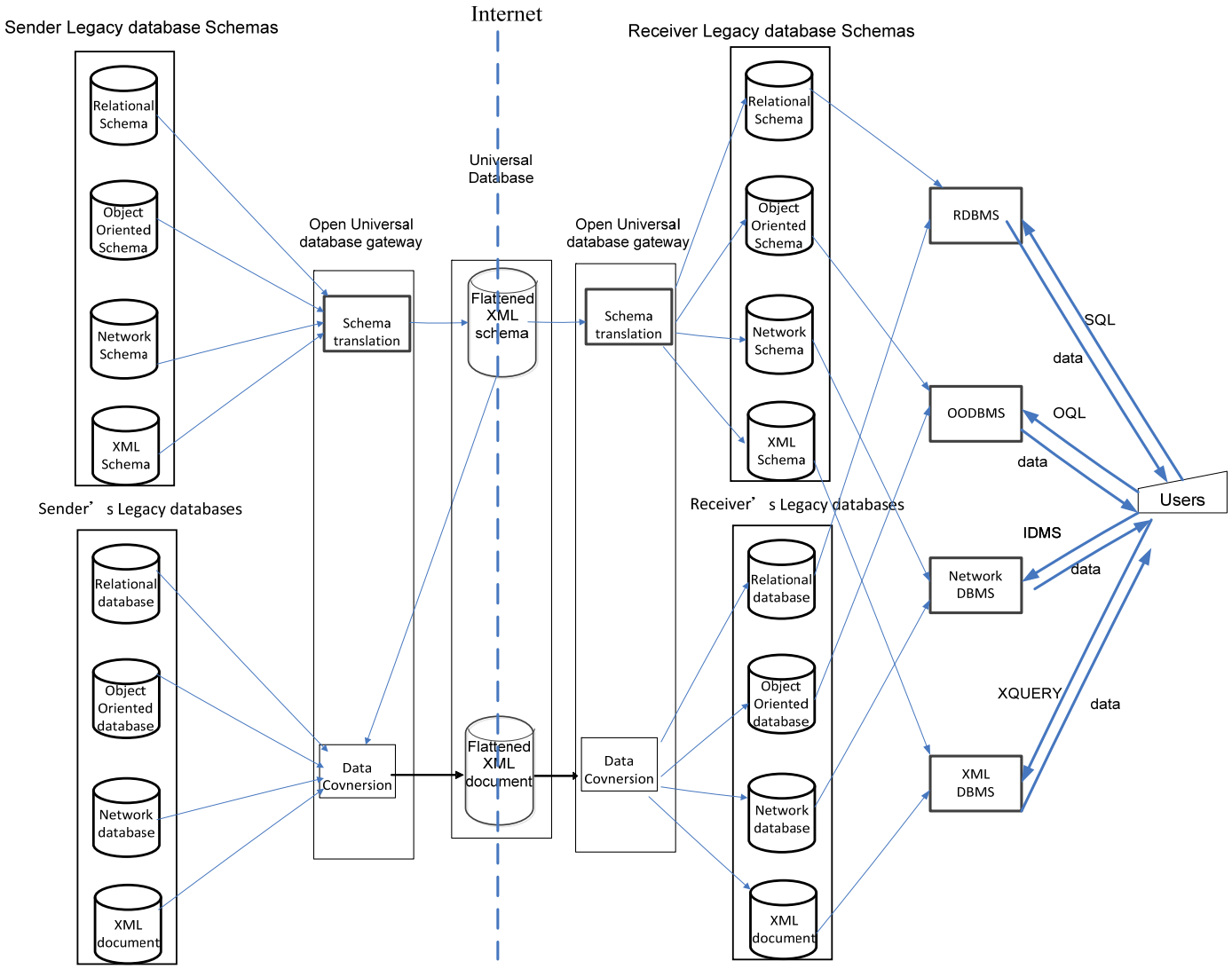


Figure 3 An open universal database gateway for legacy databases data materialization

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[7]. The result makes legacy databases not open to each each. On the other hand, 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 is much easier than before.

Problems:
(1) In logistics industry, many inter-companies processes such as purchase orders and invoices, are exchanged by using Electronic Data Interchange (EDI). These companies using EDI can reduce inventories, tighten supply chains and improve cash flow. However, not all companies use EDI because EDI is too expensive, complex and not worth the investment. 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.
(2) 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.
(3) All legacy database systems are proprietary. Database vendors do not facilitate tools to export their databases to other legacy databases.
(4) Most users cannot access all legacy databases because they do not know all legacy database languages.
(5) It is difficult to convert legacy databases in different data models because the data conversion of legacy database involves data models transformation.

Solution:
This paper proposes converting legacy databases into an information capacity equivalent and maintainable flattened XML document to achieve the interoperability among legacy databases. Therefore, users can use same database language access different legacy databases including relational, object-oriented, network and XML. Therefore, the operation can be more reliable and speedy because the same data can be concurrently processed by legacy database languages and their replicated flattened XML document rapidly on the web at the same time.

Academic merit:
XML document and legacy database data can be used in parallel processing for B2B and EDI in e-commerce on the Internet. Data can be transmitted rapidly on the web even if its XML database is down since its counterpart legacy database can help to recover it very fast. The result is a more reliable and concurrent Internet computing environment through flattened XML documents for the users. The benefit is that we can create replicate flattened XML documents which are compatible with legacy databases for rapid user friendly computing on the Internet.

Industrial merit:
The application of flattened XML document is for information highway on the Internet for data warehouse, decision support systems[10], e-commerce, and cloud computing. The benefits are information sharing among users for database interoperability.

Application:
(1) Flattened XML document is an united model of all data models of legacy databases.
(2) OUDG helps users access all legacy databases via flattened XML documents on the Internet for information highway.
(3) OUDG can transform a legacy database into another legacy database via flattened XML documents for database reengineering, recovery and integration.

Processing:
Pre-process: Reverse engineering legacy database schema into legacy database conceptual schema to recover data semantics. Schema translation between legacy database schemas and flattened XML schema can be performed before data transformation.

Step 1 Transform sender's legacy databases into flattened XML documents:
The OUDG transforms the legacy database into flattened XML documents according to the translated flattened XML schema in the pre-process.

Step 2 Transform flattened XML documents into receiver's legacy databases:
OUDG transforms the flattened XML documents into legacy database according to the translated legacy database schema in the pre-process as shown in Figure 2.

Meta data

In order to abstract data semantics from legacy database, meta data is needed to store the corresponding cardinality, isa, generalization and aggregation among various data models as follows:

Table 2 Meta data tables for universal database

Relational database meta data (classification table)

Table name	PR1	PR2	SR1	SR2	KAP	KAG	FKA	NKA	cardinality	Isa	Aggregation

Object-Oriented database meta data

Class name	Attributes	OID	Stored OID(s)	Cardinality	isa	Aggregation

XML database meta data

Element name	Attribute(s)	ID	IDREF(s)	Cardinality	isa	Aggregation

Network database meta data

Record Name	Attribute(s)	Key	Owner	Member	Cardinality	isa	Aggregation

Where

Data Type = primitive data format of each attribute in the data model

Table name = relation name.

PR1 = This is a relation whose primary key does not contain a key of another relation.

PR2 = This is a relation whose primary key contains a key of another relation.

SR1 = This is a relation whose primary key is fully formed by concatenation of primary keys of other relations.

SR2 = This is a relation whose primary key is partially formed by concatenation of primary keys of other relations.

KAP = This is an attribute in the primary key of SR1 that is also a key of some primary relations PR1 or SR1.

KAG = These are all the other primary key attributes in a secondary relation SR1 or SR2 that are not of the KAP type.

FKA = This is a foreign key attribute of a relation.

NKA = This is a non-key attribute.

OID = object identity in object-oriented schema

Stored OID = The OID value of association attribute in object-oriented schema.

ID = unique address of an element data occurrence in XML schema.

IDREF = pointer structure of an element referring to another element data occurrence.

Owner = the owner record linked by a SET to a member record in network schema.

Member = the member record linked by a SET to a member record in network schema.

Key = the key attribute of a record in network schema.

Data flows:

(1) The data semantics of legacy database schemas are captured into a meta data.
(2) Legacy database schemas are mapped into a flattened XML document schema.
(3) The data of legacy database are transformed into the flattened XML document.
(4) The flattened XML document schemas are mapped into the legacy database schemas.
(5) The flattened XML document are transformed into the legacy database according to the mapped legacy database schema.

Data Semantics preservation in legacy databases

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 in the 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; association attribute of a class object on "one" side points to another class objects on "many" side in object-oriented schema; owner record occurrence on "one" side and member record occurrences on "many" side in network schema; and 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; a subclass inherits its superclass's OID and attributes in object-oriented schema; an owner record has same key as its member record in network schema via SET linkage and 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 object-oriented schema, multiple subclasses objects contain the same OID as their superclass object. The subclasses' objects inherit their superclass's object attributes and methods. In network schema, one owner record links with multiple member records through a SET. In XML, multiple subclass elements and their superclass element are in 1:1 linkage with same key attribute.
Generalization can be implemented by multiple isa relationships with multiple subclasses generalized into one superclass.
(d) Aggregation is a method to form a composite object from its components.
It aggregates attribute values of an entity to form a whole entity. For example, if set B is part of set A, and set C is part of set A, then sets B and C can be aggregated into set A.
If aggregation relation is deleted, then component relations is also deleted in relational schema. Similarly, an aggregation class owns component classes such that the deletion of the owner class implies the deletion of its component classes in object-oriented schema. Similarly, an aggregation record links with component records such that the deletion of the aggregation implies the deletion of the component record in network schema. Also an aggregation element associates with component elements such that the deletion of the former implies the deletion of the latter in XML schema.

Before data transformation, OUDG maps major data semantics of cardinality, isa, generalization and aggregation into each legacy data model as shown in Table 3:

Table 3 Data semantics preservation 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 associates to another class's objects' OID(s) as a Stored OID.	An owner record data points to many member records data via SET linkage.	An element contains many sub-elements.	The IDREF(s) of a "many" side sibling element's data refer to an ID of "one" side sibling element data.
m:n cardinality	A relationship relation's composite key are foreign keys referring to 2 other relations' primary keys.	A class's association attribute refers to another class's objects' OID(s), and vice versa.	Two owner records data point to same member record data via 2 SETs linkages.	A sub-element links 1 element in element sub-element linkage and links another element by IDREF referring to ID.	A sibling element data contains 2 IDREF(s) referring to the ID of 2 other sibling elements data.
isa	Subclass relation's primary key is also a foreign key referring to its superclass relation's primary key.	A subclass inherit OID(s) and attributes of its superclass as its own additional attributes.	An owner record data links to a member record data in 1:1 with same key.	An element occurrence links a sub-element occurrence in 1:1 linkage.	The IDREF of a subclass sibling element data refers to the ID of a superclass sibling element with the same key value.
Generalization	2 subclass relations' primary keys are also foreign keys referring to same superclass	Two subclasses inherit OID(s) and attributes of same superclass as their	An owner record data points to two member records data with same key under 1	An element occurrence links with two sub-elements in 1:1 data	The IDREF(s) of 2 subclass sibling elements data refer to an ID of a superclass

	relation's primary keys.	own additional attributes.	SET linkage.	occurrence linkages.	sibling element data. All of them have same key value.
aggregation	An aggregation relation links with its component relations. Delete aggregation relation will delete its components relations.	An aggregation class owns components classes such that delete aggregation class will delete component classes.	An aggregation record points components classes such that delete aggregation record will delete component records.	A group element has component elements such that delete group element will delete component elements.	A group element has component elements such that delete group element will delete component elements.

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

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

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

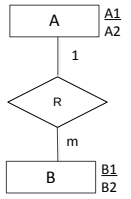
Definition of MVD

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 syntaxes of the three data semantics of cardinality, isa, generalization and aggregation representation of legacy databases and the flattened XML schema can be shown in Figure 4. Similarly, 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 5.

(a) one-to-many cardinality

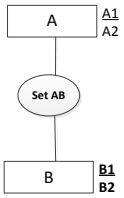
Relational conceptual schema in EER model



Relational Schema

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

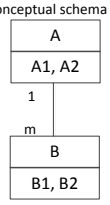
Network conceptual schema in Network Graph



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

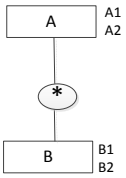
Object-Oriented Conceptual schema in UML



Object-Oriented Schema

Class A
Attribute A1 Char
Attribute A2 Char
Attribute A2B set (B)
End
Class B
Attribute B1 Char
Attribute B2 Char
Attribute B2A (A)
End

XML conceptual schema in DTD Graph



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>
```

Relations

R1

A1	A2
a11	a21
a12	a22

R2

B1	B2	*A1
b11	b21	a11
b21	b22	a12

FD: B → A

Records

A

A1	A2
a11	a21
a12	a22

B

B1	B2
b11	b21
b12	b22

FD: B → A

Classes

A

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

B

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

FD: B → A

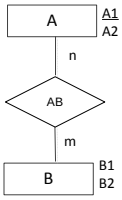
XML Docment

```
<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

(b) many-to-many cardinality

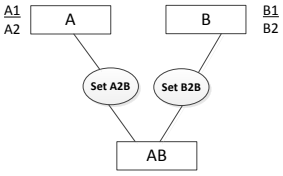
Relational conceptual schema in EER model



Relational Schema

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

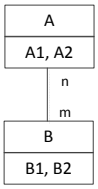
Network conceptual schema in Network Graph



Network Schema

Record Name is A
A1 Character
A2 Character
Record Name is B
B1 Character
B2 Character
Record Name is AB
Set A2B
Owner is A
Member is AB
Set B2B
Owner is B
Member is AB

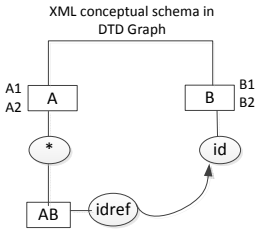
Object-Oriented Conceptual schema in UML



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

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>
```

Relations

A

A1	A2
a11	a21
a12	a22

B

B1	B2
b11	b12
b21	b22

AB

*A1	*B1
a11	b11
a12	b21

MVD: B →→ A
MVD: A →→ B

Records

A

A1	A2
a11	a21
a12	a22

B

B1	B2
b11	b21
b12	b22

AB

A1	B1
a11	b11
a12	b12

MVD: A →→ B
MVD: B →→ A

Classes

A

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

B

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

MVD: A →→ B
MVD: B →→ A

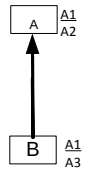
XML Docment

```
<A A1=" a11", A2=" a21">
   <AB idref=" 1"></AB>
   <AB idref=" 2"></AB>
</A>
<A A1=" a12", A2=" a22">
   <AB idref=" 1"></AB>
</A>
 <B B1="b11" B2=" b12" id=1"></B>
 <B B1="b12" B2=" b12" id=2"></B>
```

MVD: A →→ B
MVD: B →→ A

(c) Isa relationship

Relational conceptual schema in EER model



Relational Schema

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

Relations

A

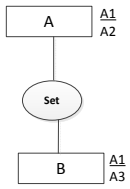
A1	A2
a11	a21
a12	a22

B

*A1	A3
a11	a31
a21	a32

ID: B ⊏ A

Network conceptual schema in Network Graph



Network Schema

Record Name is A
A1 Character
A2 Character
Record Name is B
A1 Character
A3 Character
Set AB
Owner is A
Member is B

Records

A

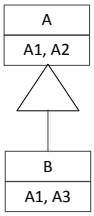
A1	A2
a11	a21
a12	a22

B

A1	A3
a11	a31
a12	a32

ID: B ⊑ A

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

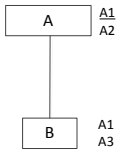
Classes

OID_A	A1	A2
#1	a11	a21
#2	a12	a22

OID_B	A1	A3	A2
#1	a11	a31	a21
#2	a12	a32	a22

ID: B ⊒ 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>

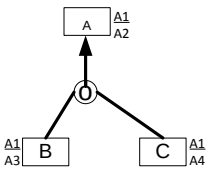
XML document

<A A1=" a11" A2=" a12"></A>
<B A1=" a11" A3="a31" ></B>
</A>
<A A1=" a21" A2=" a22"></A>
<B A1=" a21" A3="a32" ></B>
</A>

ID: B ⊑ A

(d) generalization

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

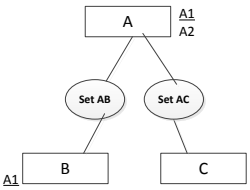
*A1	A3
a11	a31
a21	a32

C

*A1	A4
a11	a41
a21	a42

ID: (B , C) ⊑ A

Network conceptual schema in Network Graph



Network Schema

Record Name is A
A1 Character
A2 Character
Record Name is B
A1 Character
A3 Character
Record Name is C
A1 Character
A4 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

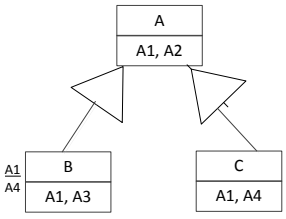
*A1	A3
a11	a31
a12	a32

C

A1	A4
a11	a41
a12	a42

ID: (B , C) ⊒ A

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

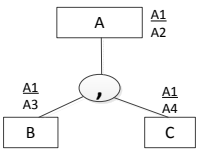
OID_B	A1	A2
#1	a11	a21

C

OID_B	A1	A3
#2	a12	a31

ID: (B , C) ⊒ 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 , C) ⊒ A

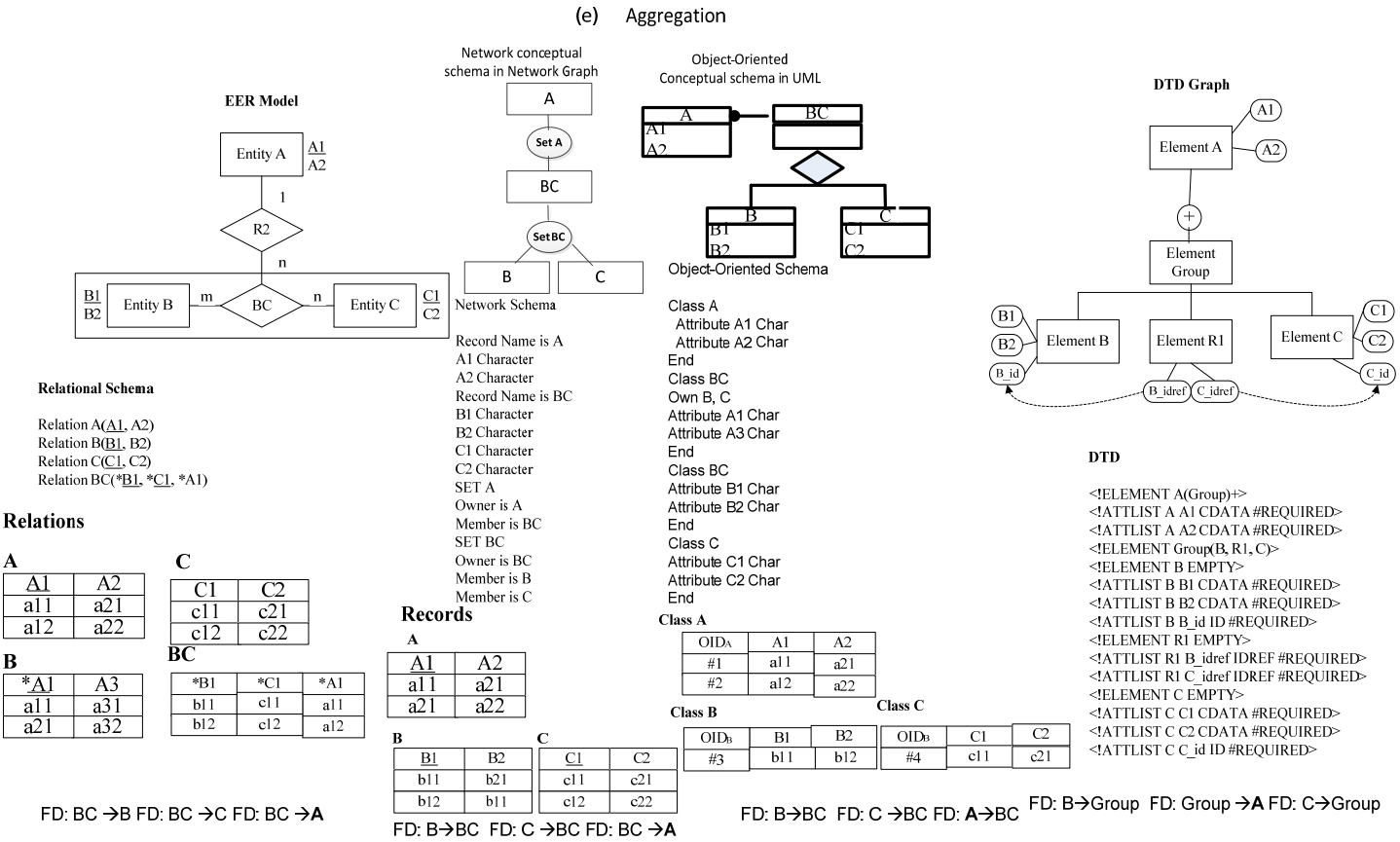
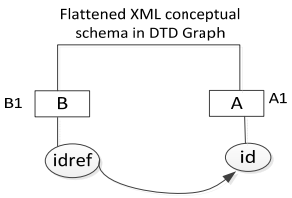


Figure 4 Data semantics preservation in equivalent legacy databases

(a) one-to-many cardinality



Flattened XML Document schema in DTD

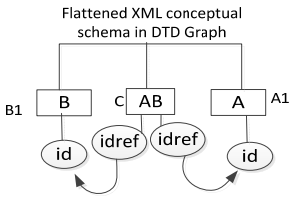
```
<!ELEMENT ROOT(A, B)>
<!ELEMENT A EMPTY>
<!ATTLIST A id ID #REQUIRED>
<!ATTLIST A A1 CDATA #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B idref IDREF #REQUIRED>
<!ATTLIST B B1 CDATA #REQUIRED>
```

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 → A

(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 A1 CDATA #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B id ID #REQUIRED>
<!ATTLIST B B1 CDATA #REQUIRED>
<!ELEMENT AB EMPTY>
<!ATTLIST AB idref1 IDREF #REQUIRED>
<!ATTLIST AB idref2 IDREF #REQUIRED>
<!ATTLIST AB C CDATA #REQUIRED>
```

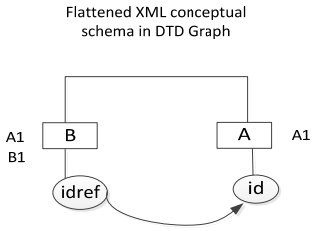
Flattened XML Document Data

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

MVD: A →→ B

MVD: B →→ A

(c) isa relationship



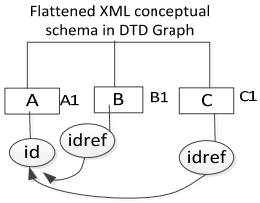
Flattened XML Document schema in DTD

```
<!ELEMENT ROOT(A, B)>
<!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>
<!ATTLIST B B1 CDATA #REQUIRED>
```

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

ID: B ⊆ A

(d) generalization



Flattened XML Document schema in DTD

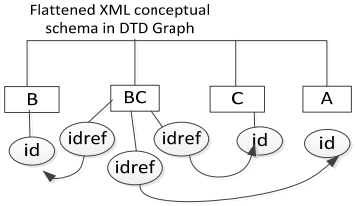
```
<!ELEMENT ROOT(A,B,C)>
<!ELEMENT A EMPTY>
<!ATTLIST A id ID #REQUIRED>
<!ATTLIST A A1 CDATA #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B idref IDREF #REQUIRED>
<!ATTLIST B B1 CDATA #REQUIRED>
<!ELEMENT C EMPTY>
<!ATTLIST C idref IDREF #REQUIRED>
<!ATTLIST C C1 CDATA #REQUIRED>
```

Flattened XML Document Data

```
<ROOT>
    <A A1="a11" id="1"></A>
    <A A1="a12" id="2"></A>
    <B A1="a11" B1="b11" idref=1"></B>
     <C A1="a12" C1="c11" idref=2"></C>
</ROOT>
```

ID: (B | C) ⊆ A

(e) aggregation



Flattened XML Document schema in DTD

```
<!ELEMENT ROOT(A,B,C, BC)>
<!ELEMENT A EMPTY>
<!ATTLIST A id ID #REQUIRED>
<!ELEMENT B EMPTY>
<!ATTLIST B id ID #REQUIRED>
<!ELEMENT C EMPTY>
<!ATTLIST C id ID #REQUIRED>
<!ELEMENT BC EMPTY>
<!ATTLIST BC idref1 IDREF #REQUIRED>
<!ATTLIST BC idref2 IDREF #REQUIRED>
<!ATTLIST BC idref3 IDREF #REQUIRED>
```

Flattened XML Document Data

```
<ROOT>
    <A id="1"></A>
    <B id="2"></B>
    <C id="3"></C>
    <BC idref1="  1", idref="  2"idref1="  3" ></BC>
</ROOT>
```

FD: BC→A FD: BC→B FD: BC→C

Figure 5 Data semantics preservation in flattened XML documents

2 Related Works

On data transformation

Shoshani[1] defined logical level approach data conversion by using source and target schemas to perform data type conversion instead of physical data type conversion. He provided a general methodology of using logical level approach of downloading source legacy database into sequential file and uploading them into target legacy database for data transformation.

Lum et al[2] showed how to construct data conversion languages SDDL and TDL to extract and restrict data from source legacy database into target legacy database.

Fong and Bloor [3] described mapping navigational semantics of the network schema into a relational schema before converting data from network database to relational database.

Fong[4] presented a methodology of transforming object-oriented database objects into Relational database by using SQL Insert statements.

Fong and Shiu[5] designed a Semantic Export Markup Language as a data conversion language to export component of relational database into XML database.

Fong et al.[6] applied logical level approach for data materialization between relational database and object-oriented database using sequential file as medium.

On Heterogeneous database

Given huge investment for a company put into heterogeneous databases, it is difficult for company 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[9]. For instance, we can provide an relational interface to non-relational database such as Hierarchical, Network, Object-Oriented and XML[16].

Hsiao and Kamel[7] offered a solution of multiple-models-and-languages-to-multiple-models-and–languages mapping to access heterogeneous databases.

Fong[8] suggested translating heterogeneous database schemas into Extended Entity Relationship Model as a conceptual schema for information retrieval. There are tools that perform transaction translation from SQL to other DBMS's data manipulation language.

On Universal database

Fong[9] proposed using a frame model metadata to unite different data models of various databases as an universal database, which can perform data operations in a table.

Fong et al.[10] applied universal database system to access universal data warehousing for the integration of both relational databases and object-oriented databases with star schema and OLAP functions.

Silverston and Graziano[11] used a universal data model in a diagram to design the conceptual schema of different legacy data models of any legacy database.

On Homogeneous database

Timos Sellis, Chih-Chen Lin, and Louiqa Raschid [12] 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.

On schema translation

Navathe et. al. [13] 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.

Funderburk et al. [14] proposed DTD Graph as XML conceptual schema which is identical to DTD, but in a graph format.

On Cloud Database
[15] defines cloud database as databases in virtual machines.

On Relational Interface
Fong[16] applied an relational API (application program interface) to access hierarchical and network databases by SQL, schema translation pre-processing and online transaction translation.

On Flattened XML document
Fong et al.[17] converted an XML document into Relational database by transforming XML document into flattened XML document with relational table structure by Extensible Stylesheet Language Transformation.

This paper extends the work of universal database into an "open" universal database gateway. The limitation of an 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 paper 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, OUDG differs from ODBC because ODBC requires programming solution to access different 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 as the trend of IT technology.

3 Methodology of open universal database gateway(OUDG)

This paper proposes OUDG as a database middleware to access legacy databases via flattened XML documents as an open database as follows:

Sender Legacy databases → Flattened XML documents (universal database) →Receiver Legacy databases

The major functions of OUDG are:

(1) Pre-process of recovering data semantics from legacy databases:

In general, the data semantics are presented in the legacy database conceptual schemas. However, if the legacy database conceptual schemas do not exist or become obsolete, it is necessary to recover their data semantics by reverse engineering legacy database logical schema into legacy database conceptual schema.

For example, to recover an EER model from a relational schema, a classification table can be used to define the relationship between keys and attributes in all relations, and data semantics can be recover 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 etc.

Similarly, we can recover 1:n association between two associated objects with a Stored OID in a class referring to many OID(s) in another associated class in OODB. We can also recover 1:n cardinality from owner and members records of Network schema into Network database conceptual schema, and also element and sub-elements in XML schema into XML conceptual schema as shown in table 2.

For example, we can apply classification table to abstract cardinality into EER model from relational schema[13][23]. We can also extract XML schema from XML document using reverse engineering[22].

(2) Transform sender's legacy databases into flattened XML documents
Any legacy database can be transformed into flattened XML document as follows:

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 conversion. 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 table structured sibling XML elements linked with id and idref. In many-to-many data semantic, it will post 2 relations and their relationship relation into table structured 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 as shown in Figure 4 and Figure 5.

```
Algorithm: Map relational schema into flattened XML schema
Begin
        Select an artifact root element for flattened XML schema;
        Map all parent only (PR1 in classification table) entities into sibling elements under root element;
          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;
                /*A is one and B is many */
                 map relation A into sibling element A with ID;
                 map relation B into sibling element B with IDREF refer to the above ID;
                end;
         If relation B has a primary key which is also a foreign key refers to relation A primary key
          Then begin
                 /* Map subclass relation A isa superclass relation B into sibling element A isa sibling element B */
                 map relation A into sibling element A with ID value of relation key value;
                 map relation B into sibling element B with IDREF value of the same relation key value;
                end;
          If (relation A and relation B is in 1:n cardinality) And (relation C and relation B is in 1:n)
          Then relation A and relation C are in m:n cardinality;
          If (relation A isa relation B) and (relation C isa relation B)
          Then subclass relation A and subclass relation C are generalized into superclass relation B;
          If a composite key of a relationship relation BC refer primary keys of relations B and C
          Then begin
                    /* Map aggregation of relations BC,B,C into aggregation of sibling elements BC,B,C */
                    map relation B into sibling element B with ID value;
                    map relation C into sibling element C with ID value;
                    map relation BC into sibling element BC with IDREF(s) refer to the above ID(s);
                end;
End;

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 conversion. The Input is an XML database and the output is a flattened XML document with relational table structure. 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 XML 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 XML sibling elements linked with id(s) and idref(s). In isa data semantic, it will post superclass and subclass elements into XML sibling elements linked with id and idref with the same key as shown in Figure 4 and Figure 5.

```
Algorithm:  Map XML schema into flattened XML schema:
Begin
   If element A and its sub-element B have same attribute a1 in XML schema
   Then begin
          /* Map subclass element A isa superclass element B into sibling elements A isa B */
          map element A with key a1 into sibling element A with key a1 and an ID value into flattened XML schema;
          map element B with key a1 into sibling element B with key a1 and IDREF with 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 */
           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 */
   If a group element BC has component elements B and C
   Then begin
          /* Map aggregation with elements BC,B,C into aggregation with sibling elements BC,B,C */
          map element B into sibling element B with ID value;
          map element C into sibling element C with ID value;
          map element BC into sibling element BC with IDREF(s) refer to the above ID(s);
      end;
End;

Algorithm: Transform an XML document to a flattened XML document
Input: an XML document
Output: a flattened XML document
Begin
      Perform depth first search on the input XML document instance;
         For each XML element do
         begin
            Add an ID attribute id with value "entity:sequence_number";
            Add an IDREF attribute idref with value  "parent_element_name:seqeuence number of parent;
            Create a new blank XML document with root as root element;
            Perform depth first search (top-to-bottom, left-to-right and front-to-back) on the input XML document instance;
            For each element in the input XML document instance do
           Add a sibling XML element with attributes id and idref to the new XML document;
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 man-to-many data semantic, it will post 2 sets of associated objects with a common object into 3 XML sibling elements such that a sibling element with 2 IDREF(s) referring 2 sibling elements with 2 ID(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 as shown in Figure 4 and Figure 5.

```
Algorithm: Map object-oriented schema into flattened XML schema:
Begin
   If B is subclass of class A
   Then begin
            /* Map subclass class B isa superclass class A into sibling element A isa sibling element B */
            map class A with OID into sibling element A with ID value same as OID of element A;
            map class B with same OID as above into sibling element B with IDREF referring to the above ID value;
           end;
     If class A has association attribute (A2B) referring to class B's multiple objects
     Then begin
              /* Map Classes A and B in 1:n cardinality into sibling elements B and C in 1:n cardinality */
             Map class A with OID into sibling element A with ID value same as OID;
             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 subclass to superclass B*/
     If a class BC owns classes B and C
     Then begin
             /* Map aggregation with classes BC,B,C into aggregation with sibling elements BC,B,C */
              map class B into sibling element B with ID value;
              map class C into sibling element C with ID value;
              map class BC into sibling element BC with IDREF(s) refer to the above ID(s);
             end;
End;

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 child element of root element of flattened XML document with OID
            as ID type attribute;
       End
     For each association attribute of obj do
           Begin
              For each stored referred obj do
             Begin
                    Locate the corresponding sibling XML element e in flattened XML document:
                    Create an IDREF attribute for element e:
             End
           End
   End
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 as shown in Figure 4 and Figure 5.

Preprocess algorithm: Map Network schema into flattened XML schema:

```
Begin
   If (owner record A has a key value attribute a1) and (member record B under owner record A has same key value a1)
   Then begin
            /* Map subclass Record B isa superclass record A into sibling element A isa sibling element B */
            Map record A into sibling element A with an attribute a1 and with ID value into flattened XML schema;
            Map record B into sibling element B with as above and an IDREF referring to element A's ID value  into flattened XML
            schema;
          End;
    If member record B under owner record A
    Then begin
             /* Map Records A and B in 1:n cardinality into sibling elements A and B in 1:n cardinality */
             Map record A into sibling element A with ID value into flattened XML schema;
             Map record B into sibling element B with IDREF referring to the above ID value into flattened XML schema;
          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 subclass sibling element A and subclass sibling element C are generalized into superclass sibling element B;
     If an owner record BC has member records B and C
     Then begin
              /* Map aggregation with sibling elements BC,B,C into aggregation with elements BC,B,C */
              map record B into sibling element B with ID value;
              map record C into sibling element C with ID value;
              map record BC into sibling element BC with IDREF(s) refer to the above ID(s);
           end;
      End;
```

Algorithm of transforming NDB to flattened XML documents

Input: A NDB instance

Output: A flattened XML document

```
Begin
  Create a flattened XML document with a root element
  For each record type t in NDB do
  begin
     For each record r of type t do
     Begin
          Locate the key for type t for record r;
          Create a sibling XML element for record r as child element of root element of flattened XML document with key as
          ID type attribute;
      End
  end
  For each set s in NDB do
  Begin
    For each i in s do
    Begin
          For the non-owner record type t defined by s do
          Begin
                    Locate the corresponding sibling XML element e in flattened XML documents;
                    Create an IDREF attribute for sibling element e for the owner record defined by I;
```

```
        End
    End
End
```

Step 2: Transform flattened XML documents into receiver's legacy databases

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. Therefore, we can post the flattened relational structured XML document into a legacy database of relational, object-oriented, network or XML.

Case 5: Transform flattened XML documents into relational databases

Firstly, we perform the preprocess of mapping flattened XML schema into flattened XML 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 as shown in Figure 4 and Figure 5.

```
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 subclass Sibling element B isa superclass sibling element A into relation A isa relation B */
          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 */
          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 subclass sibling A and subclass sibling element C are generalized into superclass sibling element B;
    If an aggregation exists with component sibling elements BC, B and C
    Then begin
          /* Map aggregation with sibling elements BC, B and C into aggregation with relations BC, B and C */
          map sibling element B with ID into relation B with ID as primary key value;
          map sibling element C with ID into relation C with ID as primary key value;
          map sibling element BC with 2 IDREF(s) into sibling element BC with IDREF(s) as composite key refer to the
          above ID(s);
        end;
End;

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 sub element c of e do      /* extract attributes from the sub-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 sub 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 6: 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 as shown in Figure 4 and Figure 5.

```
Algorithm: Map flattened XML schema into object-oriented 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 class B isa class A */
           map sibling element A into class A with attribute a1 into object-oriented schema;
           map sibling element B into subclass B of class A in object-oriented 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 classes A and B in 1:n cardinality */
           map sibling element A into class A with association attribute A2B referring to class B's multiple objects in OODB
           schema;
           map sibling element B into class B with association attribute B2A referring to class A's object in OODB schema;
          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 subclass & B is their superclass */
    If an aggregation exists with component sibling elements BC, B and C
    Then begin
           /* Map aggregation with sibling elements BC, B and C into aggregation with relations BC, B and C */
           map sibling element B with ID into relation B with ID as primary key value;
           map sibling element C with ID into relation C with ID as primary key value;
           map sibling element BC with 2 IDREF(s) into sibling element BC with IDREF(s) as composite key refer to the
           above ID(s);
          end;
     end;
Algorithm: Create OODB statements from flattened XML documents
Input: flattened XML document
```

```
Output: A sequence of OODB OQL statements
Begin
  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
          If       (sibling element Ai ID name = sibling element Ai IDREF name)
             and (sibling element Aj ID name = sibling element Ai IDREF name)
          Then sibling element Ai isa sibling element Aj; /* subclass element Ai and superclass element Aj */
          If sibling element Ai ID name = sibling element Aj IDREF name
          Then sibling element Ai and sibling element Aj are in 1:n cardinality; /* element Ai links many element Aj */
          If sibling element Ai IDREF name = sibling element Aj ID name
          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 value with the same key;
          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 with associated Stored OID(s) in text file T;
   n:1:   Output update statement of Aj to replace null value with associated Stored OID in text file T;
          case end;
   end;
   Execute OODB DML (data manipulation statements) in the text file T by OODB DBMS (database management systems);
end
```

Case 7 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.

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 man-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 as shown in Figure 4 and Figure 5.

Algorithm: Map flattened XML schema into network 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 record B isa record A */
           map sibling element A with attribute a1 into owner record A with key attribute a1 into network schema;
           map sibling element B with attribute a1 into member record B under record A with same attribute a1 into network
            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 records A and B in 1:n cardinality */
           map sibling element A with attribute ID value a1 into owner record A with key attribute a1 into network schema;
           map sibling element B into member record B under record A into network schema;
           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 subclass, & B is their superclass */
    If an aggregation exists with component records BC, B and C
   Then begin
           /* Map aggregation with sibling elements BC, B and C into aggregation with records BC, B and C */
           map sibling element B with ID into record B with ID as primary key value;
           map sibling element C with ID into record C with ID as primary key value;
           map sibling element BC with 2 IDREF(s) into owner record BC with member records B and C via a SET;
          end;
end;
```

Algorithm Transform flattened XML document into Network database:

```
//* Create CSV file from flattened XML document */
begin     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 =’val’;
                                                  Append “,” to vals;
                              End;
```

```
                    Add vals to the CSV file;
                Begin
                Export the CSV file;
        End
End
```

Case 8: 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 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 as shown in Figure 4 and Figure 5.

```
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 subclass sibling element B isa superclass sibling element A into element B isa element A */
        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 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 */
        map sibling element A into element A in XML schema;
        map sibling element B into element B under element A in XML schema;
        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 subclass & B is their superclass */
    If an aggregation exists with component sibling elements BC, B and C
   Then begin
        /* Map aggregation with sibling elements BC, B and C into aggregation with elements BC, B and C */
         map sibling element B with ID into element B with same ID value;
         map sibling element C with ID into element C with same ID value;
        map sibling element BC with 2 IDREF(s) into a group element BC with member elements B and C;
       end;
end;

Algorithm: Post flattened XML document into an XML document
Input: A flattened XML document
Output: An XML document
Begin
  Let xml = replicate of flattened XML document;
  Call Restructure XML with xml;
  Return xml
End
Function: Restructure XML
Begin
    For each sibling XML element e with one IDREF attribute idref do
    begin
      Locate sibling element e' with ID referred by idref;
```

Move e as child element of e';
Remove attribute idref from element e;
End
End

3 Case Study:

A logistic system records the customer shipment information including which orders are being packed and what the packing information is. Based on the XML 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[18]). The strategy for resolving many-to-many relationship[s] is to replace the relationship with two one-to-many cardinality with an association entity and then relate the two original entities to the association entity[19]. 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 ORDER_INFORMATION can be divided into BulkOrder and CustomerOrder in generalization as shown in Figure 5.

Step 1: Data transformation from relational database into flattened XML document:

The input relational conceptual schema in Extended Entity Relationship model

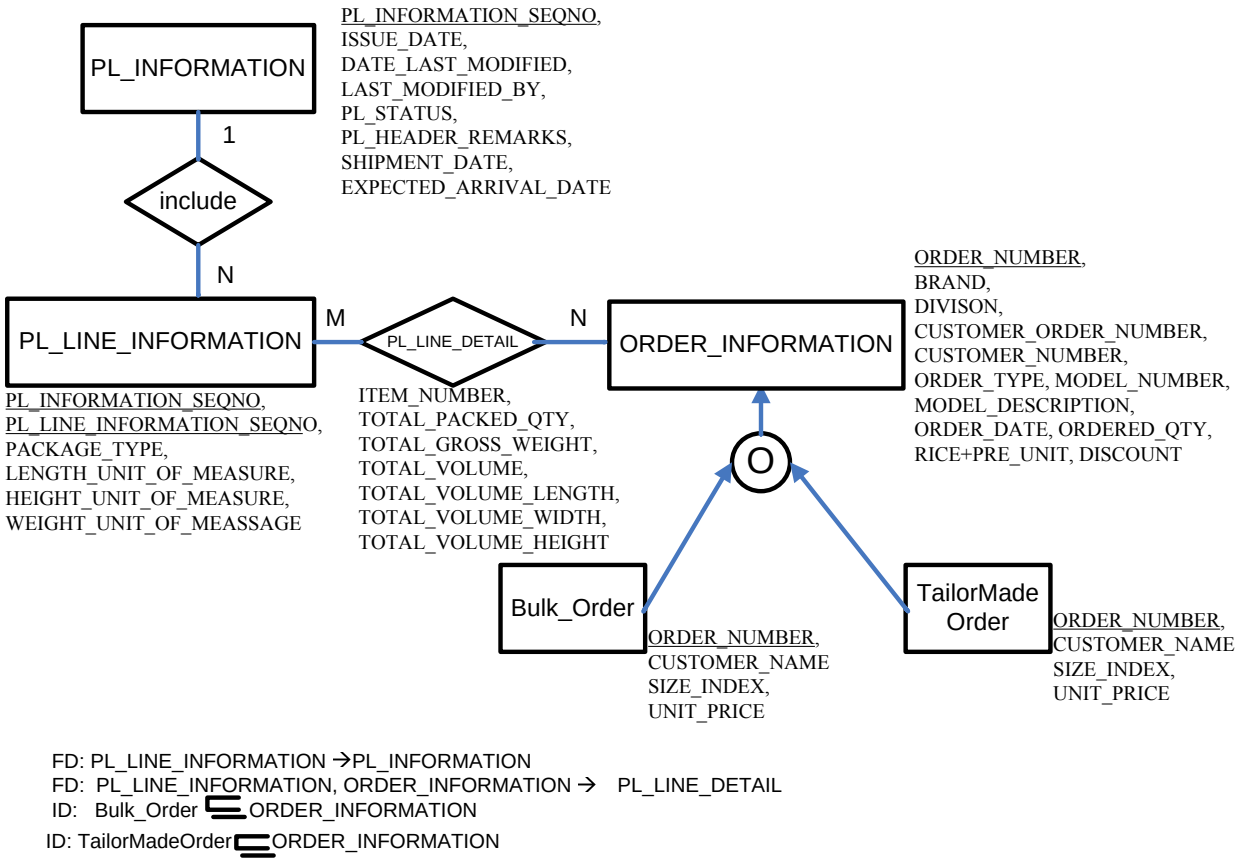


Figure 5 Input relational database conceptual schema in Extended Entity Relationship model

There are six tables. Each table has its primary key in italic, and foreign key prefixed with "*". Their data dependencies 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 shown in Figure 5.

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_VOLUMN_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	UNIT_PRICE
135792468	AMAZON	S	12.1

Table TailorMadeOrder

*ORDER_NUMBER	CUSTOMER_NAME	SIZE_INDEX	UNIT_PRICE
123469999	PETER CHAN	L	12.3

We map input relational schema into an flattened XML OUDG schema with relational structure in two levels tree only as shown in Figure 6. Notice that the second level elements (under root elements) are linked together using idref referring to id, which is similar to foreign key referring to primary key.

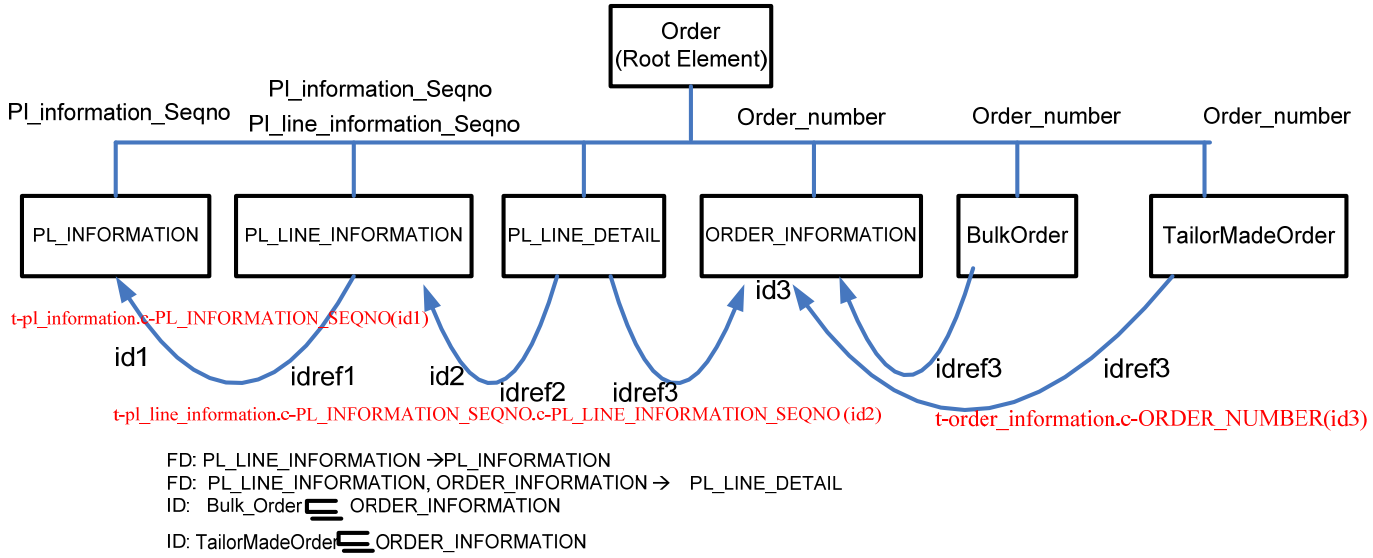


Figure 6 Transformed flattened XML document conceptual schema in DTD Graph

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 shown in Figure 6.

The flattened XML document schema is:

```
<?xml version="1.0"?>
<!DOCTYPE db-prototype [
<!ELEMENT db-prototype (t-bulk_order*,t-order_information*,t-
pl_information*,t-pl_line_detail*,t-pl_line_information*,t-
tailor_made_order*)>
<!ELEMENT t-bulk_order (c-ORDER_NUMBER,c-
CUSTOMER_NAME,c-SIZE_INDEX,c-UNIT_PRICE)>
<!ELEMENT t-order_information (c-ORDER_NUMBER, c-
BRAND,c-DIVISION, c-CUSTOMER_ORDER_NUMBER, c-
CUSTOMER_NUMBER, c-ORDER_TPYE, c-MODEL_NUMBER,
c-MODEL_DESCRIPTION, c-ORDER_DATE, c-ORDERD_QTY,
c-PRICE_PER_UNIT,c-DISCOUNT)>
<!ELEMENT t-pl_information (c-PL_INFORMATION_SEQNO, c-
ISSUE_DATE,c-DATE_LAST_MODIFIED, c-
LAST_MODIFIED_BY, c-PL_STATUS, c-
PL_HEADER_REMARKS, c-SHIPMENT_TYPE, c-
SHIPMENT_DATE, c-EXPECTED_ARRIVAL_DATE)>
<!ELEMENT t-pl_line_detail (c-PL_INFORMATION_SEQNO, c-
PL_LINE_INFORMATION_SEQNO, c-ORDER_NUMBER, c-
ITEM_NUMBER, c-TOTAL_PACKED_QTY, c-
TOTAL_GROSS_WEIGHT, c-TOTAL_VOLUME_LENGTH, c-
TOTAL_VOLUME_WIDTH, c-TOTAL_VOLUMEN_HEIGHT)>
<!ELEMENT t-pl_line_information (c-
PL_INFORMATION_SEQNO, c-
PL_LINE_INFORMATION_SEQNO, c-PACKAGE_TYPE, c-
LENGTH_UNIT_OF_MEASURE, c-
WIDTH_UNIT_OF_MEASURE, c-
HEIGHT_UNIT_OF_MEASURE, c-
WEIGHT_UNIT_OF_MESSAGE)>
<!ELEMENT t-tailor_made_order (c-ORDER_NUMBER,c-
CUSTOMER_NAME,c-SIZE_INDEX,c-UNIT_PRICE)>
<!ELEMENT c-BRAND (#PCDATA)>
<!ELEMENT c-CUSTOMER_NAME (#PCDATA)>
<!ELEMENT c-CUSTOMER_NUMBER (#PCDATA)>
<!ELEMENT c-CUSTOMER_ORDER_NUMBER (#PCDATA)>
<!ELEMENT c-DATE_LAST_MODIFIED (#PCDATA)>
<!ELEMENT c-DISCOUNT (#PCDATA)>
<!ELEMENT c-DIVISION (#PCDATA)>
<!ELEMENT c-EXPECTED_ARRIVAL_DATE (#PCDATA)>
<!ELEMENT c-HEIGHT_UNIT_OF_MEASURE (#PCDATA)>
<!ELEMENT c-ISSUE_DATE (#PCDATA)>
<!ELEMENT c-ITEM_NUMBER (#PCDATA)>
<!ELEMENT c-LAST_MODIFIED_BY (#PCDATA)>
<!ELEMENT c-LENGTH_UNIT_OF_MEASURE (#PCDATA)>
<!ELEMENT c-MODEL_DESCRIPTION (#PCDATA)>
<!ELEMENT c-MODEL_NUMBER (#PCDATA)>
<!ELEMENT c-ORDERD_QTY (#PCDATA)>
<!ELEMENT c-ORDER_DATE (#PCDATA)>
<!ELEMENT c-ORDER_NUMBER (#PCDATA)>
<!ELEMENT c-ORDER_TPYE (#PCDATA)>
<!ELEMENT c-PACKAGE_TYPE (#PCDATA)>
<!ELEMENT c-PL_HEADER_REMARKS (#PCDATA)>
<!ELEMENT c-PL_INFORMATION_SEQNO (#PCDATA)>
<!ELEMENT c-PL_LINE_INFORMATION_SEQNO (#PCDATA)>
<!ELEMENT c-PL_STATUS (#PCDATA)>
<!ELEMENT c-PRICE_PER_UNIT (#PCDATA)>
<!ELEMENT c-SHIPMENT_DATE (#PCDATA)>
<!ELEMENT c-SHIPMENT_TYPE (#PCDATA)>
<!ELEMENT c-SIZE_INDEX (#PCDATA)>
<!ELEMENT c-TOTAL_GROSS_WEIGHT (#PCDATA)>
<!ELEMENT c-TOTAL_PACKED_QTY (#PCDATA)>
<!ELEMENT c-TOTAL_VOLUMEN_HEIGHT (#PCDATA)>
<!ELEMENT c-TOTAL_VOLUME_LENGTH (#PCDATA)>
<!ELEMENT c-TOTAL_VOLUME_WIDTH (#PCDATA)>
<!ELEMENT c-UNIT_PRICE (#PCDATA)>
<!ELEMENT c-WEIGHT_UNIT_OF_MESSAGE (#PCDATA)>
<!ELEMENT c-WIDTH_UNIT_OF_MEASURE (#PCDATA)>
<!ATTLIST t-bulk_order t-bulk_order.c-ORDER_NUMBER ID
#REQUIRED>
<!ATTLIST t-bulk_order t-order_information.c-ORDER_NUMBER
IDREF #IMPLIED>
<!ATTLIST t-order_information t-order_information.c-
ORDER_NUMBER ID #REQUIRED>
<!ATTLIST t-pl_information t-pl_information.c-
PL_INFORMATION_SEQNO ID #REQUIRED>
<!ATTLIST t-pl_line_detail t-order_information.c-
ORDER_NUMBER IDREF #IMPLIED>
<!ATTLIST t-pl_line_detail t-pl_line_detail.c-
PL_INFORMATION_SEQNO.c-
PL_LINE_INFORMATION_SEQNO.c-ORDER_NUMBER ID
#REQUIRED>
<!ATTLIST t-pl_line_detail t-pl_line_information.c-
PL_INFORMATION_SEQNO.c-
PL_LINE_INFORMATION_SEQNO IDREF #IMPLIED>
<!ATTLIST t-pl_line_information t-pl_information.c-
PL_INFORMATION_SEQNO IDREF #IMPLIED>
<!ATTLIST t-pl_line_information t-pl_line_information.c-
PL_INFORMATION_SEQNO.c-
PL_LINE_INFORMATION_SEQNO ID #REQUIRED>
<!ATTLIST t-tailor_made_order t-order_information.c-
ORDER_NUMBER IDREF #IMPLIED>
<!ATTLIST t-tailor_made_order t-tailor_made_order.c-
ORDER_NUMBER ID #REQUIRED>
]>
```

The flattened XML document is shown in Figure 7.

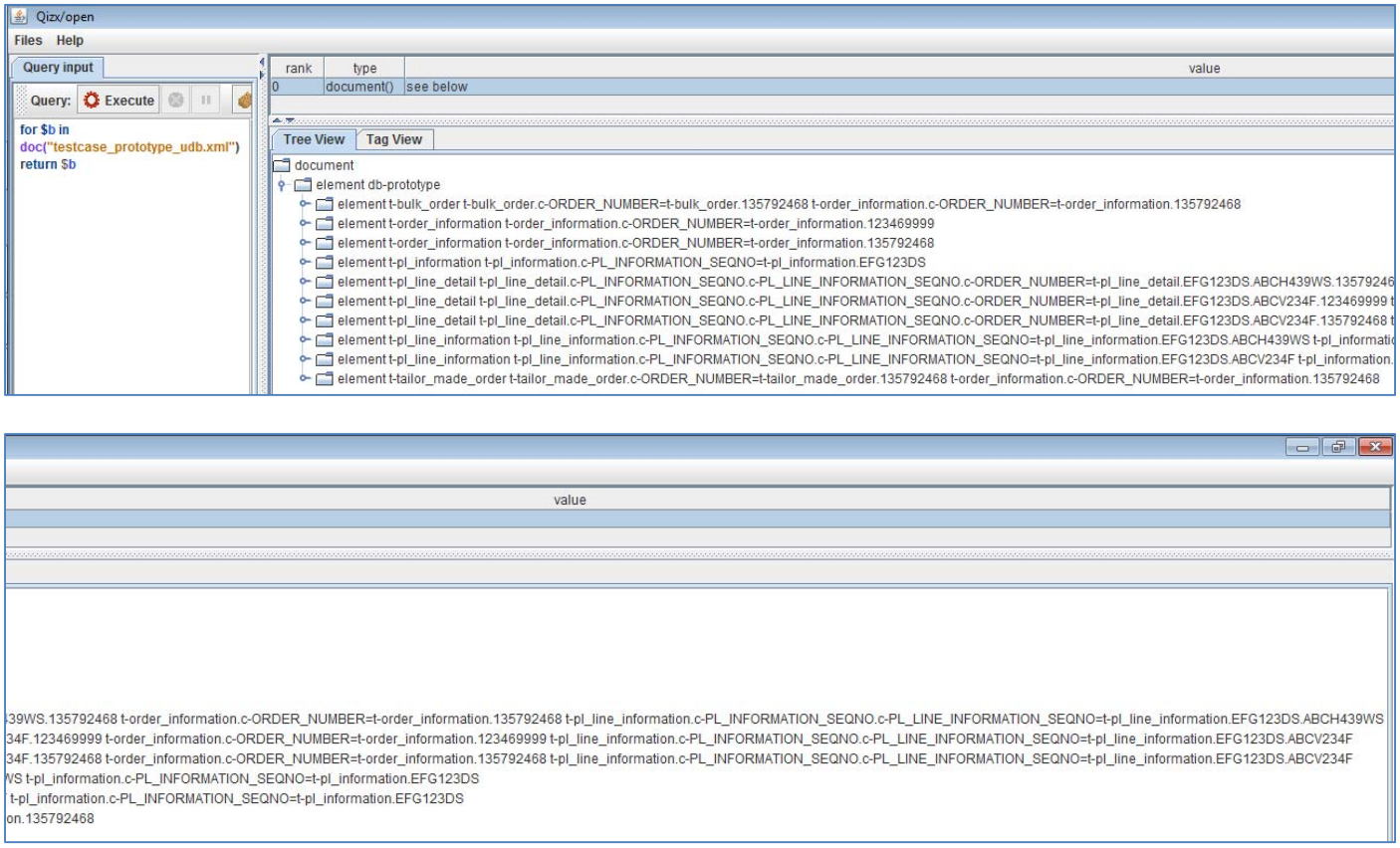


Figure 7 Transformed flattened XML document

Phase II: Data transformation 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 database

We can map the open universal database schema into XML database schema as shown in Figure 5. Figure 8 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 1:1 association. Elements Order_information and TailorMadeOrder are also in 1:1 association. This XML structure has multiple levels of elements which is different from the 2 levels elements in Open universal database schema.

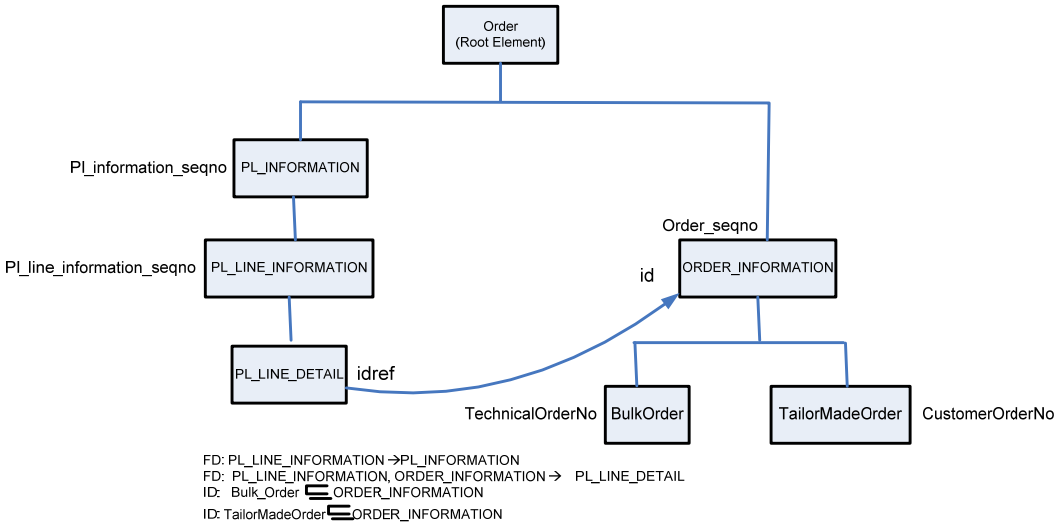


Figure 8 Translated XML document schema in DTD Graph

For example, Figure 8 shows the mapping into UML for object-oriented database schema. 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. Classes BulkOrder and TailorMadeOrder are in disjoint generalization under their superclass Order_information such that the two subclasses data are mutually exclusive.

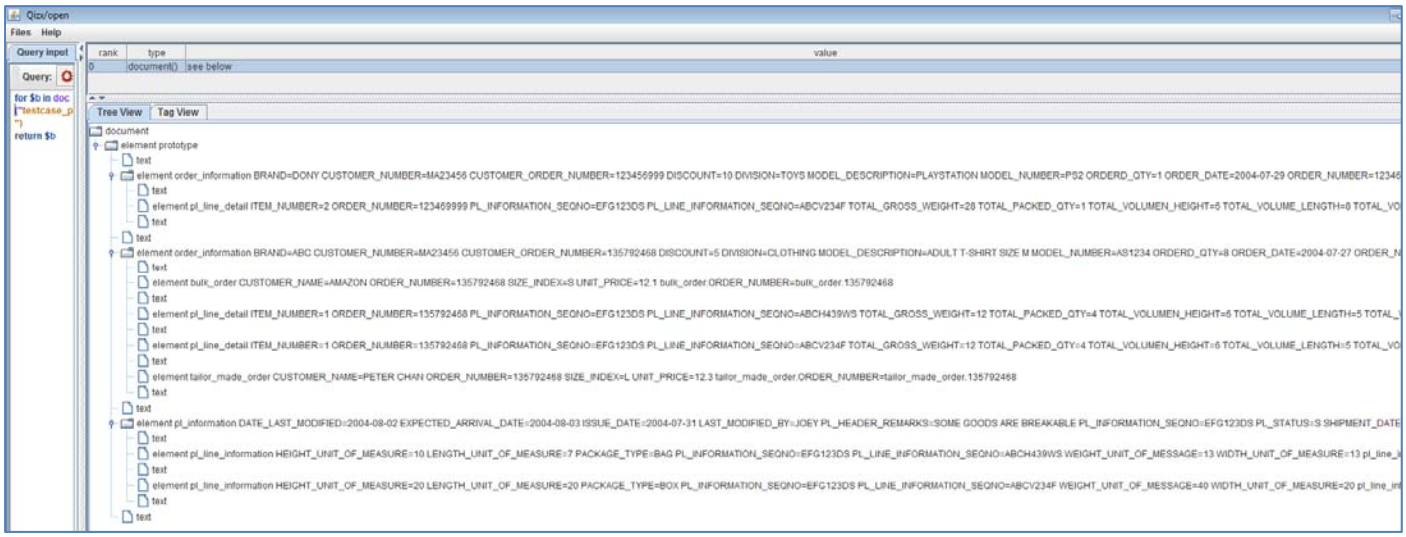
There are seven elements. The sub-element key determines its element key in FD. The idref can determine its referred id in FD. The subclass element key is a subset of its superclass key in ID as shown in Figure 8.

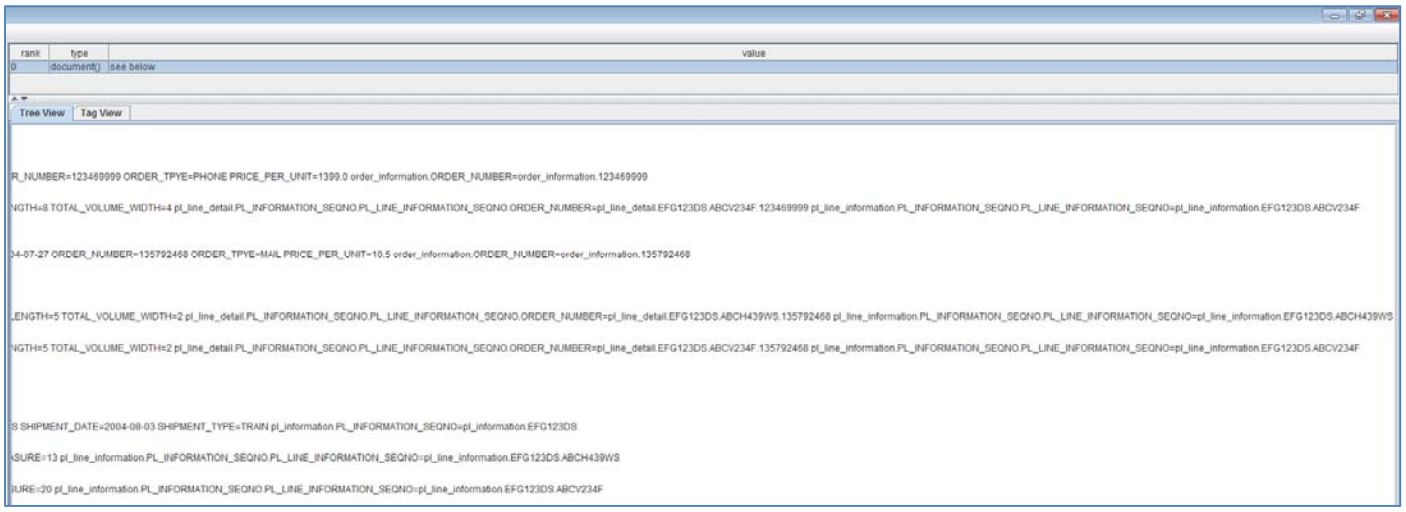
The output XML database schema is:

The output XML database schema is:

```
<!ELEMENT prototype (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
          UNIT_PRICE CDATA #REQUIRED>
<!ELEMENT TAILORMADEORDER EMPTY>
<!ATTLIST    TAILORMADEORDER
          CUSTOMER_NAME CDATA #REQUIRED
          SIZE_INDEX CDATA #REQUIRED
          UNIT_PRICE CDATA #REQUIRED>
</ORDER
```

The output XML database document is:





(b) Data transformation from flattened XML documents into Object-Oriented databases

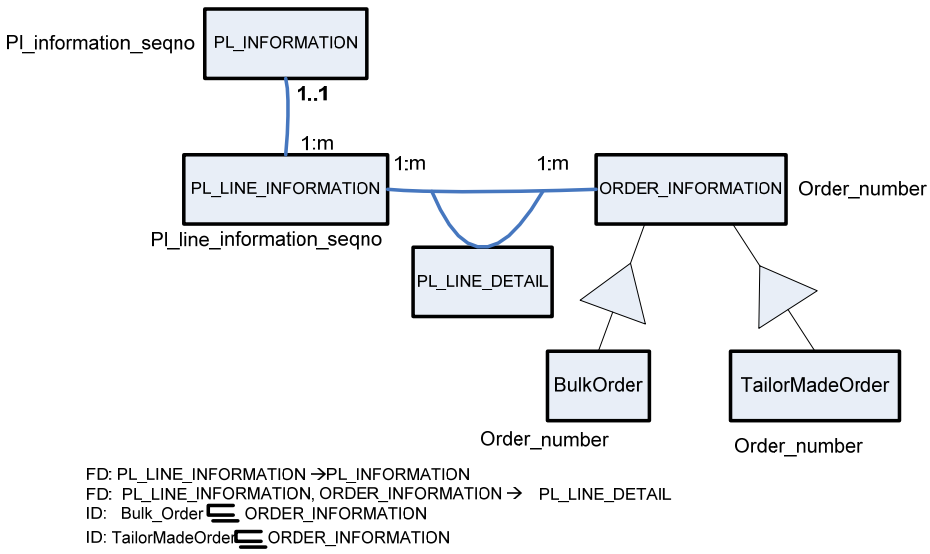


Figure 9 Translated legacy object-oriented database schema in UML

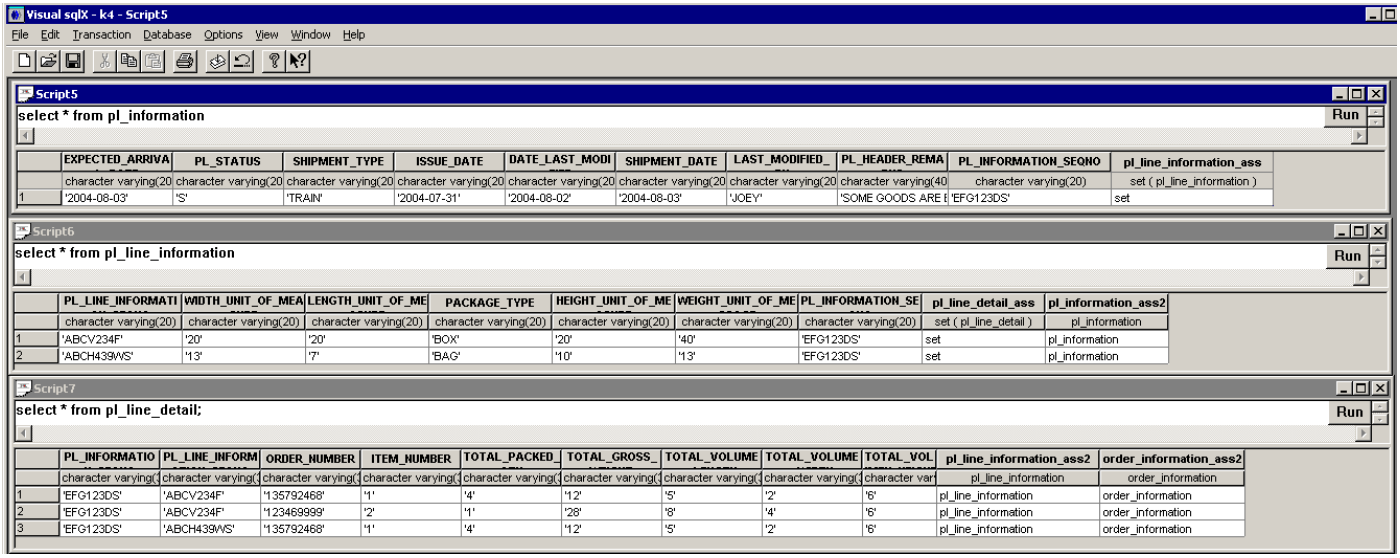
In figure 9, record Pl_informations and Order_information are under network DBMS as first records for database navigation access path. The path can go from record Pl_information to Pl_line_information in owner and member record in 1:n relationship. Records Pl_line_information (owner), Order_information(owner) 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.

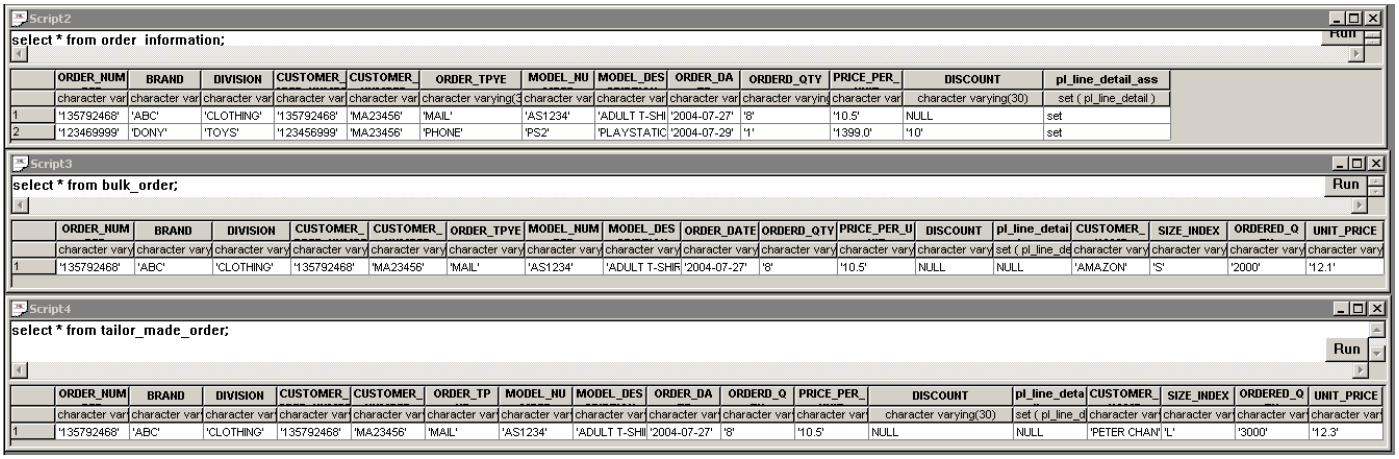
There are six classes. Each class has its OID, and Stored OID. Their data dependencies are such that each Stored OID key determines its referred OID in FD, and each subclass OID is a subset of its superclass OID in ID as shown in Figure 9.

The output Object-Oriented database schema is:

```
CREATE class PL_INFORMATION;

CREATE class PL_LINE_DETAIL;

Create class ORDER_INFORMATION;


create class PL_LINE_INFORMATION
(       PL_LINE_INFORMATION_SEQNO varchar(20),
        WIDTH_UNIT_OF_MEASURE varchar(20),
        LENGTH_UNIT_OF_MEASURE varchar(20),
        PACKAGE_TYPE varchar(20),
        HEIGHT_UNIT_OF_MEASURE varchar(20),
        WEIGHT_UNIT_OF_MESSAGE varchar(20),
        PL_INFORMATION_SEQNO varchar(20),
        pl_line_detail_ass set of (PL_LINE_DETAIL),
         pl_information_ass2 PL_INFORMATION);

alter class PL_INFORMATION
Add attribute
EXPECTED_ARRIVAL_DATE varchar(20),
PL_STATUS varchar(20),
SHIPMENT_TYPE varchar(20),
ISSUE_DATE varchar(20),
DATE_LAST_MODIFIED varchar(20),
SHIPMENT_DATE varchar(20),
LAST_MODIFIED_BY varchar(20),
PL_HEADER_REMARKS varchar(40),
PL_INFORMATION_SEQNO varchar(20),
pl_line_information_ass set of (PL_LINE_INFORMATION);

alter class PL_LINE_DETAIL
Add attribute   PL_INFORMATION_SEQNO varchar(30),
                PL_LINE_INFORMATION_SEQNO
                varchar(30),
                ORDER_NUMBER varchar(30),
                ITEM_NUMBER varchar(30),
                TOTAL_PACKED_QTY varchar(30),
                TOTAL_GROSS_WEIGHT varchar(30),
                TOTAL_VOLUME_LENGTH varchar(30),
                TOTAL_VOLUME_WIDTH varchar(30),
                TOTAL_VOLUMEN_HEIGHT varchar(30),
                pl_line_information_ass2
                PL_LINE_INFORMATION,
                order_information_ass2
                ORDER_INFORMATION;

alter class ORDER_INFORMATION
Add attribute   ORDER_NUMBER varchar(30),
                BRAND varchar(30),
                DIVISION varchar(30),
                CUSTOMER_ORDER_NUMBER varchar(30),
                CUSTOMER_NUMBER varchar(30),
                ORDER_TPYE varchar(30),
                MODEL_NUMBER varchar(30),
                MODEL_DESCRIPTION varchar(40),
                ORDER_DATE varchar(30),
                ORDERD_QTY varchar(30),
                PRICE_Per_UNIT varchar(30),
                DISCOUNT varchar(30),
                pl_line_detail_ass set of (PL_LINE_DETAIL);

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

The output Object-Oriented Database Base is:





(d) Data transformation from flattened XML OUDG into Network database

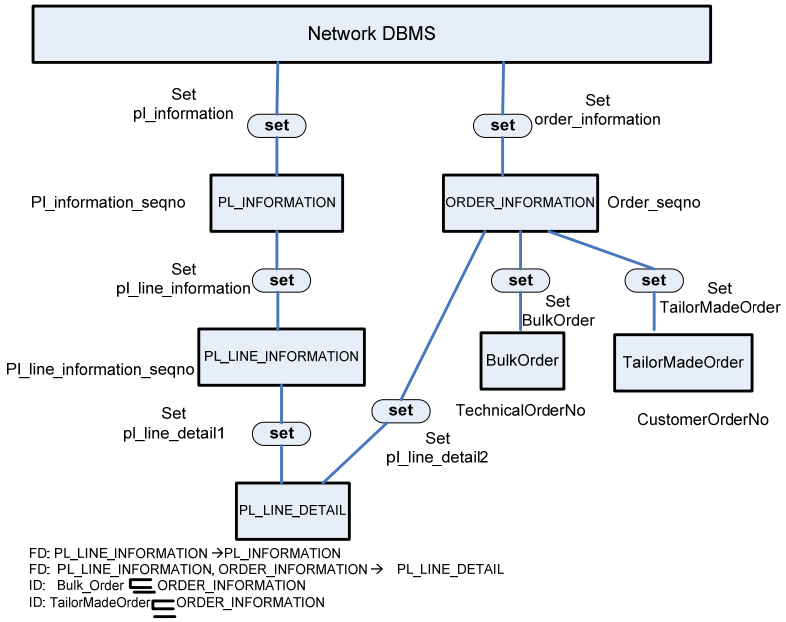


Figure 10 Translated legacy network database schema in Network Graph

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 shown in Figure 10.

The output 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 CUSTOMER_NAME[31];
        char UNIT_PRICE[31];
        char ORDER_NUMBER[31];
        char SIZE_INDEX[31];
compound key D {
ORDER_NUMBER;
}
}
record TAILORMADEORDER{
        char CUSTOMER_NAME[31];
        char UNIT_PRICE[31];
        char ORDER_NUMBER[31];
        char SIZE_INDEX[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;
    }
}
```

```
C:\Raima\RDM\11.0\win32\bin>ddlp -d NDB1.ddl

Database Definition Language Processor 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\


0 errors detected

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 BREAKABLE','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','MA23456','ABC','5','135792468','8'
[bulk_order.csv:000001] 'AMAZON','12.1','135792468','S'
[tailor_made_order.csv:000001] 'PETER CHAN','12.3','135792468','L'
Successful import

C:\Raima\RDM\11.0\win32\bin>_
```

(e) Data transformation from flattened XML documents to relational database

RDB (output) schema

```
CREATE TABLE bulk_order (
CUSTOMER_NAME VARCHAR(10),
UNIT_PRICE VARCHAR(10),
ORDER_NUMBER VARCHAR(10),
SIZE_INDEX VARCHAR(10),
PRIMARY KEY (ORDER_NUMBER));
CREATE TABLE tailor_made_order (
CUSTOMER_NAME VARCHAR(10),
UNIT_PRICE VARCHAR(10),
ORDER_NUMBER VARCHAR(10),
SIZE_INDEX VARCHAR(10),
PRIMARY KEY (ORDER_NUMBER));
CREATE TABLE order_information (
CUSTOMER_ORDER_NUMBER VARCHAR(10),
DISCOUNT VARCHAR(10),
CUSTOMER_NUMBER VARCHAR(10),
ORDER_TPYE VARCHAR(10),
BRAND VARCHAR(10),
MODEL_DESCRIPTION VARCHAR(10),
MODEL_NUMBER VARCHAR(10),
ORDER_NUMBER VARCHAR(10),
ORDERD_QTY VARCHAR(10),
DIVISION VARCHAR(10),
ORDER_DATE VARCHAR(10),
PRICE_PER_UNIT VARCHAR(10),
PRIMARY KEY (ORDER_NUMBER));
CREATE TABLE pl_line_information (
PL_INFORMATION_SEQNO VARCHAR(10),
PL_LINE_INFORMATION_SEQNO VARCHAR(10),
WIDTH_UNIT_OF_MEASURE VARCHAR(10),
HEIGHT_UNIT_OF_MEASURE VARCHAR(10),
PACKAGE_TYPE VARCHAR(10),
WEIGHT_UNIT_OF_MESSAGE VARCHAR(10),
LENGTH_UNIT_OF_MEASURE VARCHAR(10),
```

```
PRIMARY KEY
(PL_INFORMATION_SEQNO,PL_LINE_INFORMATION_SEQN
O));
CREATE TABLE pl_line_detail (
TOTAL_VOLUME_LENGTH VARCHAR(10),
PL_INFORMATION_SEQNO VARCHAR(10),
TOTAL_GROSS_WEIGHT VARCHAR(10),
TOTAL_VOLUME_WIDTH VARCHAR(10),
TOTAL_VOLUMEN_HEIGHT VARCHAR(10),
ORDER_NUMBER VARCHAR(10),
TOTAL_PACKED_QTY VARCHAR(10),
PL_LINE_INFORMATION_SEQNO VARCHAR(10),
ITEM_NUMBER VARCHAR(10),
PRIMARY KEY
(PL_INFORMATION_SEQNO,PL_LINE_INFORMATION_SEQN
O,ORDER_NUMBER));
CREATE TABLE pl_information (
SHIPMENT_DATE VARCHAR(10),
ISSUE_DATE VARCHAR(10),
DATE_LAST_MODIFIED VARCHAR(10),
PL_INFORMATION_SEQNO VARCHAR(10),
PL_STATUS VARCHAR(10),
SHIPMENT_TYPE VARCHAR(10),
PL_HEADER_REMARKS VARCHAR(10),
EXPECTED_ARRIVAL_DATE VARCHAR(10),
LAST_MODIFIED_BY VARCHAR(10),
PRIMARY KEY (PL_INFORMATION_SEQNO));
ALTER TABLE bulk_order ADD FOREIGN KEY
(ORDER_NUMBER) REFERENCES
order_information(ORDER_NUMBER);
ALTER TABLE tailor_made_order ADD FOREIGN KEY
(ORDER_NUMBER) REFERENCES
order_information(ORDER_NUMBER);
ALTER TABLE pl_line_information ADD FOREIGN KEY
(PL_INFORMATION_SEQNO) REFERENCES
pl_information(PL_INFORMATION_SEQNO);
ALTER TABLE pl_line_detail ADD FOREIGN KEY
(ORDER_NUMBER) REFERENCES
order_information(ORDER_NUMBER);
ALTER TABLE pl_line_detail ADD FOREIGN KEY
(PL_INFORMATION_SEQNO,PL_LINE_INFORMATION_SEQN
O) REFERENCES
pl_line_information(PL_INFORMATION_SEQNO,PL_LINE_INFO
RMATION_SEQNO);
```

```
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)
```

4 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 2. 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

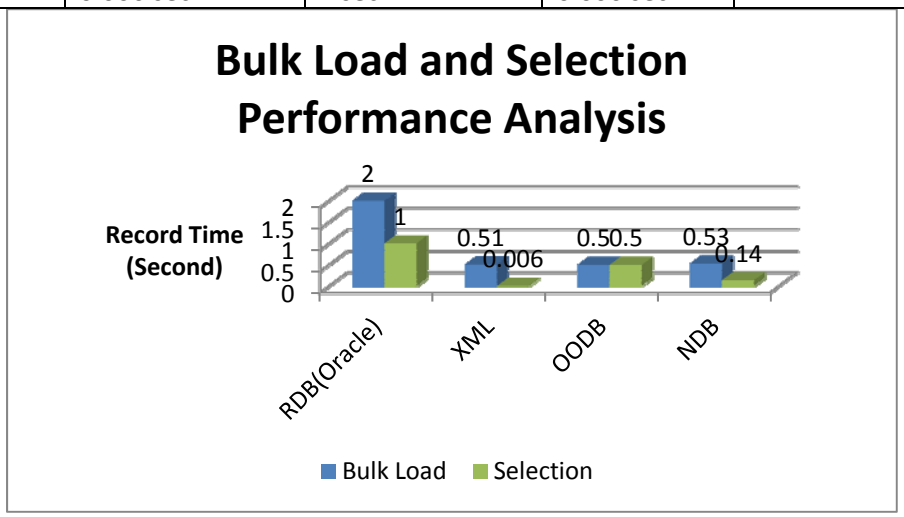
First, we bulk load 400 record of Relational database source of the prototype OUDG. Then we measure the time for these 4 output database legacy for this bulk load (Table 5).
Second, we query the data from one table from each database legacy. We measure the time and recorded in table 6.

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

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



The above figure was compared the bulk load and selection performance analysis in 4 output database legacy. The X axis represents the record time(second) while the Y axis represents 4 output database legacy from the UDB. From the figure above, RDB is poorest in performance in both bulk load and selection while XML is the best for selection.

Summary

In theory, the performance of OODB, NDB, XML are better than RDB in bulk load, in the sequence of selection performance are XML > NDB > OODB> RDB. It is because XML are in Dom Tree structure which is the best for selection. RDB requires value match, therefore it's performance is poorest. NDB is pointer structure, so it is better than OODB which requires table format and pointer structure.

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

5 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 project, we propose a Flattened XML database as universal database such that it is most user friendly and portable as a database middleware for all legacy database.

The uniqueness of the paper are:

(1) Openness of an universal database:

The reason we choose flattened XML document is due to 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 by 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 a replicate legacy database, it can be used to recover any legacy database whenever the production legacy database is down. As a result, replicate XML document can be parallel processing with legacy database in non-stop computing.

(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 allows users of any legacy database access other legacy database. The proposed OUDG unites all legacy databases 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.

In summary, the data conversion methodology of this OUDG is to download the raw data of source database into sequential file using source database schema, and upload the sequential file into target database using translated target database schema, which is a logical level approach, and which can 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 programs for the data conversion; otherwise, we need 4 * 4 = 16 programs, instead of the current 4 + 4 = 8 programs to do the 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 computer network for information retrieval.

References:

1. Shoshani, A., "A Logical-Level Approach to Data Base Conversion", ACM SGMOD International Conference on Management of Data, pp.112-122, 1975.
2. V.Y.Lum, N.C. Shu and B.C. Housel, "A General Methodology for Data Conversion and Restructuring", IBM Journal of research and development, Volume 20, Issue 5, pp.483-497, 1976.
3. Joseph Fong and Chris Bloor, "Data Conversion Rules from Network to Relational Databases", Information and Software Technology, Volume. 36 No. 3, pp. 141-154, 1994.
4. Joseph Fong, "Converting Relational to Object-Oriented Databases", SIGMOD RECORD, Volume 26, Number 1, pp53-58, 1997.
5. Joseph Fong and Herbert Shiu "An interpreter approach for exporting relational data into XML documents with Structured Export Markup Language' Journal of Database Management, volume 23, issue 1, 2012.
6. Joseph Fong, Ringo Pang, Anthony Fong, Francis Pang, and Kenny Poon, "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, 2003.
7. David K. Hsiao, and Magdi N. Kamel, "Heterogeneous Databases: Proliferations, Issues, and Solutions", IEEE Transactions on Knowledge and Data Engineering, Voumn 1, Np. 1. Pp.45-62., 1989.
8. Joseph Fong, "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.

9. Joseph Fong and Shi-Ming Huang, "Architecture of a Universal Database: A Frame Model Approach", International Journal of Cooperative Information Systems, Volume 8, Number. 1, pp. 47-82, 1999.
10. Joseph Fong, Qing Li and Shi-Ming Huang, "Universal Data Warehousing Based on a Meta-Data Modeling Approach", International Journal of Cooperative Information Systems, Volume 12, Number 3, pp.325-363, 2003.
11. Len Silverston and Kent Graziano, www.360doc.com/content/08/0830/01/1032_1590731.shtml
12.. Timos Sellis, Chih-Chen Lin, and Louiqa Raschid, "Coupling Production Systems and Database Systems: A Homogeneous Approach", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 5, NO. 2, APRIL 1993
13. Navathe, S. and Awong, A., Abstracting relational and hierarchical data with a semantic data model, Entity-Relationship Approach, p305-333, (1988).
14. J. E. Funderburk et al., "XTABLES: Bridging Relational Technology and XML", IBM Systems Journal, Vol 41, No. 4, PP 616 –641, 2002.
15. Derrick Harris, "cloud-databases-101-who-builds-em-and-what-they-do", GIGAOM, July 2012. http://gigaom.com/cloud/cloud-databases-101-who-builds-em-and-what-they-do/
16. Joseph Fong, "Adding Relational Interface to Non-relational Database", **IEEE Software**, pp. 89-97, 1996.
17. Joseph Fong, Herbert Shiu and Jenny Wong "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, 2009..pp. 249-281
18. MySQL AB:: An Introduction to Database Normalization, http://dev.mysql.com/tech-resources/articles/intro-to-normalization.html
19. Data Modeling: Refining The Entity-Relationship Diagram, http://www.utexas.edu/its/windows/database/datamodeling/dm/refining.html
20. Joseph Fong, "Information Systems Reengineering and Integration", Springer Verlag, ISBN 1-84628-382-5, 370 pages, July 2006.
21. Joseph Fong, "Methodology for Schema Translation from Hierarchical or Network into Relational", ***Information and Software Technology***, *Volume 34, Number 3, pp. 159-174, 1992.*
22. Herbert Shiu and Joseph Fong "Reverse Engineering from an XML Document into an Extended DTD Graph"**,** *Journal of Database Management, Volume 19, No 4, pp.62-80, 2008.*
23. Joseph Fong, "Methodology for Schema Translation from Hierarchical or Network into Relational", *Information and Software Technology*, *Volume 34, Number 3, pp. 159-174, 1992. (impact factor: 1.20)*