Using Relational Database metadata to generate enhanced XML structure and document

Mokhtar Boshra - Sherif Sakr
Faculty of Computers and Information – Cairo University
mokhtarboshra@hotmail.com sherif.saker@ci-wi.com

Abstract
Relational databases get more and more employed in order to store the contents of a web site. At the same time, XML is fast emerging as the dominant standard at the hypertext level of web site management, describing pages and links between them. Thus, the integration of XML with relational database systems to enable the storage, retrieval and update of XML documents is of major importance. Also it has a great importance to be able to extract structured data from XML documents and store it in database as well as to generate XML data from data extracted from database. This paper describes a new technique of using relational database metadata to generate enhanced XML document and its structure using XML-schema language from data extracted from relational database

1. Introduction

Web-based information systems no longer aim at purely providing read-only access to their content, which is simply represented in terms of web pages stored in the web application server's directory. Nowadays, due to new requirements emerging from several areas such as electronic commerce, the employment of databases to store the content of a web site turns out to be worthwhile. This allows to easily handling both retrieval and update of large amounts of data in a consistent way on a large distributed scale. Besides using databases at the *content level*, XML is fast emerging as the dominant standard for representing the *hypertext level* of a web site.

An XML document consists of possibly nested *elements* rooted in a single element. Elements, whose boundaries are delimited by *start-tags* and *end-tags,* may comprise *attribute*s, whereby both are able to contain *value*s. User-defined *element type* and *attribute specifications,* allow describing the meaning of the content.

Because of the increasing importance of XML and database systems (DBS), the integration of them with respect to storage, retrieval, and update is a major need, regarding the kind of DBS used for the integration, one can distinguish four different approaches [7]. First, *special-purpose DBS* are particularly tailored to store, retrieve, and update XML documents. Examples for these are research prototypes such as Rufus and Lore as well as commercial systems such as eXcelon and Tamino. Second, because of the rich data modeling capabilities of *object-oriented DBS,* they are well suited for storing hyper-text documents. Object-oriented DBS and special-purpose DBS, however, are neither in widespread use nor mature enough to handle large-scale data in an efficient way. Third, *Object-relational DBS* would be also appropriate for mapping to and from XML documents since the nested structure of the object-relational model blends well with XML's nested document model. Similar arguments as above, however, hold against their short-term usage. Fourth, the more promising alternative to store XML documents is *relational database systems (RDBS)*. Such integration would provide several advantages

such as reusing a mature technology, seamlessly querying data represented in XML documents and relations, the wide spread of relational database applications and the possibility to make legacy data already stored within an RDBS available for the web.
Concerning the kind of storage within an RDBS, there exist three basic alternatives.
First, the most straightforward approach would be to *store XML documents as a whole* within a *single database attribut*e. Second approach is the possibility *to interpret XML documents as graph structures* and provide a relational schema allowing storing arbitrary graph structures. The third approach is that the *structure of XML documents is mapped to a corresponding relational schema wherein XML documents are stored according to the mapping*. Only the last of these alternatives allows to really exploiting the features of RDBS such as querying mechanisms, optimization, concurrency control and the like. Despite the benefits of the mapping approach, the problem is that when defining the mapping between an XML DTD and a relational schema, one has to cope with *data model heterogeneity* and *schema heterogeneit*y. Data model heterogeneity refers to the fact that there are fundamental differences between concepts provided by XML and those provided by RDBS, which have to be considered when defining a certain mapping. These differences concern, e.g., structuring, typing and identification issues, relationships, default declarations, and the order of stored instances. Schema heterogeneity in our context means that, even if the structure of XML document and the relational schema to which the XML document should be mapped represent the same part of the universe of discourse, the design of both is likely to be different. This could be because of different goals pursued during design like redundant representation of information versus normalization.
So according to this heterogeneity, generating XML document should benefit from the metadata information available about the relational schema and the features of XML data model to generate enhanced XML document structure that makes it easier to understand and use by different users and applications.
The remainder of the paper is organized as follows. Section 2 introduces basic concepts of XML. Section 3 describes the advantage of using XML as common data exchange format. Section 4 introduces the basics of XML data modeling. Section 5 introduces comparison between concepts of XML data and relational data. Section 6 describes simple approach of storing XML document as graph structure in relational database. Section 7 describes approaches of mapping XML document structure to Relational database. Section 8 is an introduction about generating XML documents from relational Data. Section 9 describes the using of relational schema metadata to generate enhanced XML document structure and data. Finally, Section 10 concludes the paper with a short summary and gives an outlook to future work.

2. Basic XML Concepts

XML is set of rules and conventions for designing text formats in a way that produces files that are easy to generate and read, unambiguous and platform independent [12].

XML looks a bit like HTML but isn't HTML

Like HTML, XML uses tags (words bracketed by '<' and '>') and attributes (of the form name="value"), but while HTML specifies what each tag & attribute mean (how the text between them will look like in the browser), XML uses the tags only to delimit pieces of data and leaves the interpretation of data completely to the application that reads it.

XML is text

XML documents are text files as the user can use any simple text editor to generate XML files. But the rules for XML files are much stricter than HTML. A forgotten tag or an attribute without quotes makes the file unusable, while in HTML such practice is often explicitly allowed

XML is license-free and platform-independent

Opting for XML is a bit like choosing SQL for databases. There is still need to build a database and programs that manipulate it.

XML is new, but not that new

Development of XML started in 1996 and it is a W3C standard since February 1998, which my lead to suspecting that it is rather immature technology. But in fact the technology is not very new as before XML there was SGML, which was an ISO standard since 1986 and widely used for large documentation projects. The designers of XML simply took the best parts of SGML, guided by the experience with HTML and produced something that is no less powerful than SGML but vastly more regular and simpler to use over the web.

3. XML as a common Data Exchange format

XML is subset of SGML defined for use on the web. It defines a meta-language for defining XML markup language as well as the rules that documents conforming to this language must follow. XML has several advantages over other data exchange formats. Its support of Unicode makes it highly portable and capable of representing most of the world's string data. Its use of tags to label data means that documents are self-describing, makes them readable by humans and allows them to be used by applications not initially intended as targets. Its predictable format means that any application can read a document, even if it doesn't understand the semantics of tags. Figure 1 and figure 2 compare between the status of data exchange before and after using of XML.

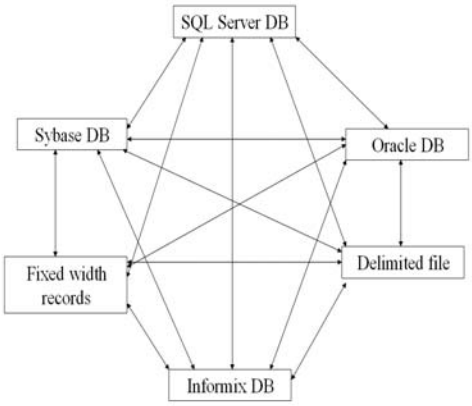

Figure 1: Data Exchange before XML

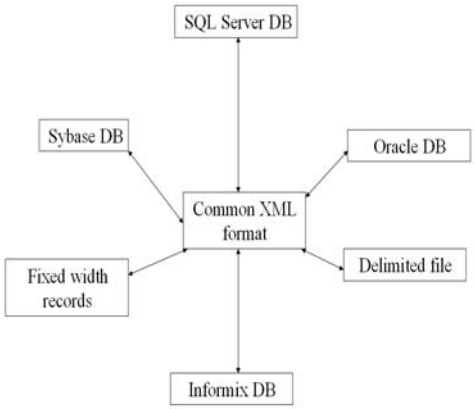

Figure 2: Data Exchange After XML

4. XML Data Modeling Basics

An XML document is essentially a medium for storing information, in order to assess the validity of an XML document you need to establish exactly to which structure the information within the document must adhere. This is accomplished with a schema which is a model used to describe the structure of information within an XML document. There

exist two alternatives for modeling XML data: - DTD (Document Type Definition) and XML-Schema. This section introduces the two techniques and compares them.

4.1 Modeling XML data with DTDs

DTDs are means of establishing schema for XML documents. DTDs were originated in SGML and serve as the standard schema mechanism for validating SGML documents. Since XML is subset of SGML, it makes sense to use the same schema approach and it is obvious that the big benefit of that is that existing SGML tools can be easily modified to support XML. But in fact DTDs has some limitations, which are serious enough to warrant considering another modeling approach. Figure 3 represents an example of modeling XML data using DTD.

```
<!ELEMENT addressbook (contact)+>
<!ELEMENT contact (name, address+, city, state, zip, phone, email, web,
company)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (voice, fax?)>
<!ELEMENT voice (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT web (#PCDATA)>
<!ELEMENT company (#PCDATA)>
```

Figure 3: Modeling XML data using DTD

4.2 Modeling XML Data with XML Schema

XML is a new approach to define the schema for XML documents that uses the same syntax of XML documents. An XML schema is very similar in purpose to DTD, in that it is used to establish the schema of a class of XML documents. Like DTDs, XML schemas describe elements and their content model so that documents can be validated. However XML schema goes several steps further than DTD by allowing you to associate data types with elements which allow XML processor to perform data content validation. . Figure 4 represents an example of modeling XML data using XML schema.

```
<?xml version="1.0"?>
<Schema name="AddressBookSchema" >
  <ElementType name="name" content="textOnly"/>
  <ElementType name="address" content="textOnly"/>
  <ElementType name="city" content="textOnly"/>
  <ElementType name="state" content="textOnly"/>
  <ElementType name="zip" content="textOnly" dt:type="int"/>
  <ElementType name="voice" content="textOnly" dt:type="int"/>
  <ElementType name="fax" content="textOnly" dt:type="int"/>
  <ElementType name="phone" content="eltOnly">
    <element type="voice" minOccurs="1" maxOccurs="1"/>
    <element type="fax" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="email" content="textOnly"/>
  <ElementType name="web" content="textOnly"/>
  <ElementType name="company" content="textOnly"/>
  <ElementType name="contact" content="eltOnly">
    <element type="name" minOccurs="1" maxOccurs="1"/>
    <element type="address" minOccurs="1" maxOccurs="2"/>
    <element type="city" minOccurs="1" maxOccurs="1"/>
    <element type="state" minOccurs="1" maxOccurs="1"/>
    <element type="zip" minOccurs="1" maxOccurs="1"/>
    <element type="phone" minOccurs="1" maxOccurs="1"/>
    <element type="email" minOccurs="0" maxOccurs="1"/>
    <element type="web" minOccurs="0" maxOccurs="1"/>
    <element type="company" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="addressbook" content="eltOnly">
    <element type="contact" minOccurs="1"/>
  </ElementType>
</Schema>
```

Figure 4: Modeling XML data using XML schema

4.3 comparing the two modeling approaches
DTD syntax is based on EBNF while XML schema is based on XML syntax.
DTD need special parser while XML schema is not.
DTD is more compact than XML schema.
There are many available tools for processing XML documents with DTD, while very few currently exist for XML schema.
XML schema supports a variety of data types while DTD treats all as strings.
XML schema supports some sort of inheritance in defining elements while DTD not.
According to the above comparison, our new technique is built to support generating XML document structure using XML schema as it is considered the most promising way for modeling XML data in the future. Also XML schema is considered the best way for representing XML document structure with making the best use of available metadata information about the relational schema.

5. XML and RDBS: Comparison of concepts

In this section, XML concepts and RDBS concepts are compared in detail, focusing on six different aspects of the data models, *comprising structuring and typing mechanisms, uniqueness of names, null values and default values, identification, relationships, and order* [1].

5.1 Structuring and Typing Mechanisms
The basic mechanisms used to specify the structure of XML documents and relational schemata are *element types* and *attributes* for XML versus *relations* and *attributes* for RDBS, respectively.
Concerning element types, it is useful to categorize them along two dimensions. The first dimension depicts whether the element type *contains an atomic domain or not* whereas the second dimension denotes whether the element type *contains a composite domain or no*t. This distinction results in four different kinds of element types. It has to be emphasized that this classification is applicable to both DTDs and XML Schema.

5.2 Uniqueness of Names
The name of a relation is required to be unique within the whole relational schema, similar to the name of an XML element type being unique throughout the DTD.
By means of so-called *namespaces,* XML allows element types having the same name by using different namespace prefixes. Namespaces, however, are not further considered in this paper. XML Schema is more flexible in this respect since the name of an XML element type has to be unique within a so-called *symbol space,* only. A symbol space is among others associated with each composite domain defined by a user. Thus, the same name may appear in composite element types being defined on the basis of different composite domains without conflict. For example, two composite domains may contain an element type address without conflict. The name of an XML attribute defined within a DTD or an XML Schema has to be unique within its element type, again similar to an RDBS attribute's name, which has to be unique within its relation.

5.3 Null Values and Default Values

Similar to RDBS, XML allows expressing null values as well as defaulting *value*s. In RDBS the concept of null values is defined for attributes, only. XML, however, supports null values for both attributes and elements. In DTDs, default values may be applied to XML attributes, only, whereas XML Schema supports default values for XML element types, too. Concerning XML attributes, the so-called *default declaration* within a DTD requires to specify for each attribute one of the following constraints:

#REQUIRED, meaning that a value is required in the sense of NOT NULL of RDBS.

#IMPLIED, denoting the optional nature of an attribute value, expressed by the omission of NOT NULL in RDBS. Note, that in case there is no value provided for such an XML attribute at the instance level, the attribute name is omitted within the XML document, too.

#FIXED <ConstValue>, defining a constant value which is not possible in RDBS.

<DefaultValue>, specifying a default value analogous to the DEFAULT clause in RDBS.

In XML Schema, there is an additional constraint for attributes named prohibited, which allows masking an inherited attribute for the actual element type. Concerning an *element*, whether it may be omitted or not is specified within both DTDs and XML Schema by means of *cardinality* constraints. The cardinality specifies how often the element of a certain element type occurs as component element of its composite element. Since element types may be components of more than one composite element type, each of its occurrences as component element type can exhibit cardinality. The cardinality symbols for DTDs are '?' (null or 1), '*' (null or more), '+' (1 or more), and no symbol (exactly 1). In XML Schema, the cardinality can be specified in more detail by using the attributes minOccurs and maxOccurs.

5.4 Identification

In RDBS, the unique identification of tuples is done by means of a *primary key,* which may be composed of one or more attributes of the corresponding relation. In DTDs, only a single attribute of an element type can be designated as identifying attribute by means of the special *attribute type* ID which may in

turn contain a string value. In addition to the DTD concept ID, XML Schema allows not just attributes, but also element types of an arbitrary atomic domain and combinations thereof to serve as keys. The *scope of identification* in RDBS is a single relation, i.e., the value of the primary key uniquely identifies each tuple within a relation. In DTDs, the scope of identification is broader in the sense that the value of an ID attribute is unique within the whole XML document. This allows the unique identification of an element not only with respect to other elements of the same element type but rather across all elements of any element type. XML Schema allows specifying the scope for each key by means of an XPath expression. In particular the Xpath expression denotes the element types and/or attributes serving as key.

In DTDs and XML Schema, element types are not required to contain an ID attribute or a key, respectively. This is similar to RDBS products, where relations need not

contain a primary key. Note, this is in contrast to the theory of the relational model, where primary keys are mandatory for each relation. Concerning DTDs, even in case that

an element type has an attribute of type ID, its usage may be optional by defining it as #IMPLIED. In contrast, keys in XML Schema must be always non-nullable. Since the identification of both tuples in RDBS and elements in XML is *value-base*d, it is not possible to distinguish between equality and identity, as it is possible in the object-oriented data model.

5.5 Relationships

In RDBS, relationships can be expressed *between relations* by means of *foreign key*s, i.e., arbitrary attributes that refer to the primary key of the same relation or of another relation. The number of tuples, which may participate in a relationship, can be constrained by defining the foreign key as NOT NULL and/or UNIQUE.

DTDs allow two alternative ways for specifying relationships between element types comprising IDREF(S) attributes and component element types. Attributes of type IDREF(S) represent some kind of foreign key referencing attributes of type ID. The distinction between IDREF attributes and IDREFS attributes concerns their cardinality, in that the former are single-valued and the latter are multi-valued. In contrast to RDBS, where the participating tuples are constrained to the participating relation, using IDREF(S) the participating elements cannot be constrained to be of a certain element type. XML Schema supports the concept of so-called *keyref*, which is similar to the RDBS concept of foreign keys, meaning that a certain element/attribute combination refers to the corresponding element/attribute combination building the key. Different to DTDs, the participants of a relationship are typed by the element type containing the key. Regarding relationships, which are realized by specifying component element types.

5.6 Order

In contrast to relations and tuples in RDBS, the element types and elements of an XML document adhere to both an *explicit* and *implicit orde*r. The order of element types can be explicitly defined within a DTD by using the sequence operator ‘,’ whereas XML Schema uses the element type sequence.

6. Storing XML document as graph structure in relational database

There exist three basic alternatives for storing XML document in relational database as described in the introduction. In this section we will describe simply the second of them which is *to interpret XML documents as graph structures* and provide a relational schema allowing storing arbitrary graph structures.

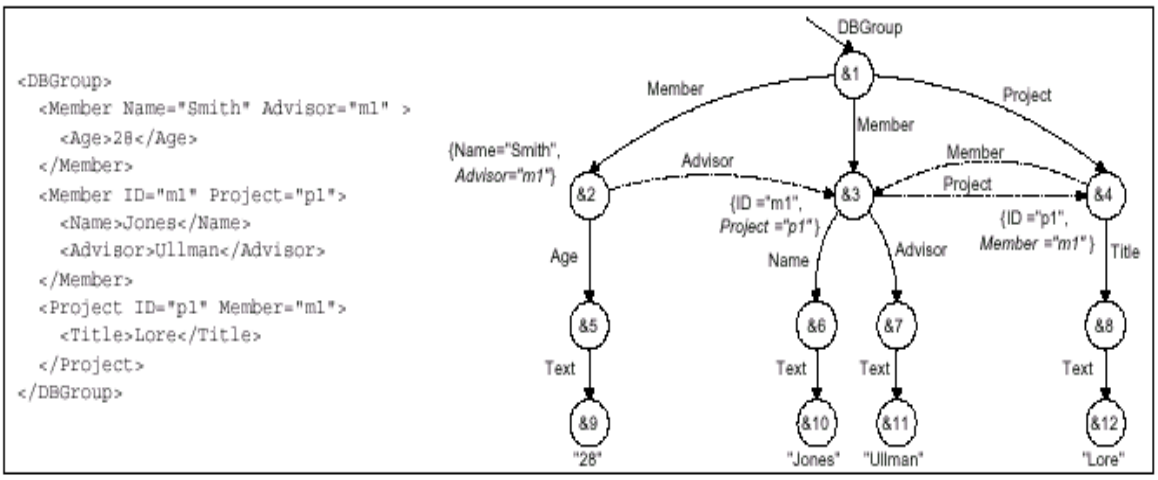

Figure 5: XML document sample and its graph representation

We assume here that an XML document can be represented as an ordered and labeled directed graph. Each XML element is represented by a node in the graph; the node is labeled with the object ID *(oid)* of the XML object. Element-subelement relationships are represented by edges in the graph and labeled by the name of the subelement. In order to represent the order of subelements of an XML object, we also order the outgoing edges of a node in the graph. Values (e.g., strings) of an XML document are represented as leaves in the graph [11].

Representing XML data as a graph is a simplification and some information can be lost in this process. The reason is that graph model does not differentiate between XML subelements and attributes, and it is does not differentiate between subelements and references (i.e., IDREFs). With the using of this technique, the original XML document cannot exactly be reconstructed from the relational data. Figure 5 represent example of XML document and its graph representation.

6.1 Mapping Edges

The simplest scheme is to store all edges of the graph that represents an XML document in a single table; let us call this table the *Edge* table, Table 1 shows an example of edge table. The *Edge* table records the oids of the source and target objects of each edge of the graph, the label of the edge, a flag that indicates whether the edge represents an inter-object reference (i.e., an internal node) or points to a value (i.e., a leaf), and an ordinal number because the edges are ordered, as mentioned above. The *Edge* table, therefore, has the indicated structure:

Source	Target	Ordinal	Name	Flag
1	2	1	Member	Ref
1	3	2	Member	Ref
1	4	3	Project	Ref
…	…	…	…	…
5	V1	1	Value	Int
…	…	…	…	…

Table 1: - Example of Edge Table

6.2 Mapping Values

We now turn to alternative ways to map the values of an XML document (e.g., strings like "Ullman" or "28"). We have two variants in this work: (a) storing values in separate *Value* tables (b) storing values together with edges.

6.2.1 Separate Value Tables

The first way to store values is to establish separate *Value* table for each conceivable data type. There could, for example, be separate *Value* tables storing all integers, dates, and all strings, table 2 shows an example of integer values table. The structure of each *Value* table is as follows, where the type of the *value* column depends on the *type* of the *Value* table: Vtype(vid, value). The *flag* column in the *Edge* table indicates in which *Value* table a value is stored; a *flag* can, therefore, take values such as *integer, date, strin*g, or *ref indicating* an inter-object reference.

VID	**Value**
V1	28
….	…

Table 2:- Example integer values Table

6.2.2 Inlining

The obvious alternative is to store values and attributes in the same tables, table 3 shows an example of the edge table with inlining. This corresponds to an outer join of the *Edge* table and the *Value* tables. Using this approach *flag* is not needed anymore, and a large number of *null* values occur.

Source	**Ordinal**	**Name**	**Vint**	**Vdate ….**	**Ref**
1	1	Member	Null	Null	2
…	…	…	…	…	…
5	1	Value	28	Null	Null

Table 3: Example of edge table with inlining

Using this approach causes loosing many of the advantage of integrating XML documents with database so it is not so common in use.

7. Mapping XML document structure to Relational database

In this section we introduce the third alternative of storing XML document in relational database, which is mapping XML document structure to a corresponding relational schema wherein XML documents are stored according to the mapping.

Two mappings are commonly used to map XML document schema to the database schema: - the *table-based mapping* and *object relational mapping* [2].

7.1 Table-Based mapping

It models XML document as single table or set of tables. That is the structure of an XML document must be as follows where the <database> element and additional <table> elements do not exist in the single table case:

```
<database>
    <table>
        <row>
            <col1> …</col1>
            <col2> …</col2>
            …….
        </row>
    </table>
    <table>
        ….
    </table>
</database>
```

Depending on the software, it may be possible to specify whether column data is stored as child element or attribute as well as the name to use for each element or attribute. An important note is that the term "table" is usually interpreted loosely, that is when transferring data from XML to the database, a "table" can be table, view or any result set returned from user query.

The table- based mapping is useful for serializing relational data, such as when transferring data between two relational databases. Its obvious drawback is that it cannot be used for any XML document that doesn't match the above format.

7.2 Object-relational mapping

It models the data in the XML document as tree of objects that are specific to the data in the document. In this model, element types with attributes, element content, or mixed content (complex element types) are generally modeled as classes. Element types with value content (simple element types) are modeled as properties. The model is mapped to relational database using traditional object-relational mapping techniques. Classes are mapped to tables, properties are mapped to columns and object-valued properties are mapped to primary key/foreign key pairs.

8. Generating XML documents from relational Data

Generating XML data from relational data consists of two steps: - generating the XML document structure and generating the XML data itself.

The most simple and common way to generate XML data from result set relational data generated by answering query over the relation database is to translate the result set to XML document.

8.1 Generating the XML document structure

The most common way for generating XML document structure is to use DTD to describe the XML document structure as follows:

```
<!ELEMENT result (row)+>
<!ELEMENT row (col1,col2,col3,…..)>
```

<!ELEMENT col1 (#PCDATA)>
<!ELEMENT col2 (#PCDATA)>
…..

8.2 Generating the XML Data

The XML data is generated in the following way:

```
<result>
        <row1>
                <column1>…</column1>
                <column2>…</column2>                .…….
        </row1>
        <row2>
                <column1>…</column1>
                <column2>…</column2>                .…….
        </row2>
</result>
```

This way of mapping considers the most simple and straightforward way of mapping relational data to XML documents but it causes loosing many of the rich features of hierarchical structure of XML documents and gives poor and flat structure description for the XML document.

9. New technique for using of relational schema metadata to generate enhanced XML document structure and data.

In this section we will describe our new technique for using the available metadata information about the relational database schema to generate enhanced XML document structure and enhanced XML document data.

Using metadata information to generate enhanced XML document is independent of using any specific tool or DBMS since the metadata information that we will consider in our technique is available in the different products of DBMS but there may be some differences in the way of storing it, so it will be the role of the tool that will implement this technique got the information from each different DBMS according to the way it stores the metadata.

Our new technique gives the user some flexibility features to choose from different alternatives of representation.

As described in section 4 there exist two ways to describe the structure of XML document:- DTD and XML schema. In our technique we choose to use XML schema to describe the structure of generated XML document because, as it was described in section 4, it has several advantages over DTD such as using the same syntax of XML, supporting a variety of data types and more flexibility in representing constraints. All of these advantages enable us to utilize the available metadata information to give more enhanced document structure. Also XML is considered the promising standard of describing XML documents in the future.

In the mapping of relational schema to XML schema, we will have an element type called the *result* which represents the root element for the generated document and each field in the result set can be mapped to a child element type tag for the *result* element type or an

attribute for the *result* element type, the default of the technique is to represent each field as child element type because the representation with attributes has some limitations such as:- attributes can not contain multiple values, can not describe structures and are not easily expandable for future changes. So, the default of the technique is to represent each field as child element type but the user can choose to represent some fields to be represented as attributes without violating the rules of XML documents.

In representing fields with element type tag we can utilize the metadata available about the field to generate enhanced element type tag about the fields. This metadata information available about the fields can be used as follows:-

1- For each field its data type can be used to describe the content of element type tag in XML-schema.

<ElementType name=”distance” dt:type=”float”>

<ElementType name=”birthdate” dt:type=”date”>

2- The order attribute for each element type can be generated automatically according to the default order in the result set but the user can make his explicit order of child tags of the *result* element type.

3- The metadata about the null ability of the field can be used to describe the minimum and maximum number of occurrences of each element type tag, for example if the field distance is allowed to have the null value then its definition will be as follows:-

<ElementType name=”distance” minoccurs=”0” maxoccurs=”1”>

on the other hand if the field does not allow to have the null value then its definition will be as follows:-

<ElementType name=”distance” minoccurs=”1” maxoccurs=”1”>

4- The metadata information available about whether each field is required or not can be used in the description of its mapping element type or attribute:

<ElementType name=”distance” required=”yes”>

5- The metadata information about the relationships between tables and the pair of PK/FK fields that participate in representing relationships can also be used to describe the cardinality of child elements within the parent elements.

<ElementType name=”dependent” minoccurs=”0” maxoccurs=”5”>.

6- The metadata information available about the key fields in the database relation can be used to describe the data type of the element type that represents key field to be *ID* data type. In the same way metadata information available about the foreign key fields can be used in representing the data types of their mapped element types to be *IDREF* data type.

7- XML schema provides for each element type an element *description*, which is used as a way of placing text description within the schema, so any information stored in the relational metadata about the description of the fields can be used in this *description* tag.

8- For the field that is represented as attribute instead of child element type, we can use the metadata information about its default value to be represented in its description (in XML schema default values are only supported with attributes not by element types as the element type can not have default value)

<attribute name=”state” default=”running”>

9- The definition of attribute in XML schema allows to define set of enumeration values for the attribute, so for more enrichment of the attribute description (also only with fields that are mapped to attributes not element types), the technique checks the distinct values for each attribute and if the number of distinct values < N (the default value of N is 5 but

the value of N can be changed according to the user preferences), a list of enumeration values is generated for the attribute description.

```
<attribute name=”state” dt:type=”enumeration”  Dt:values=”running cycling swimming”/>
```

10- The technique gives the user the ability to group more than one mapped element type using add-in parent attribute, for example if the result set has the three fields city, street and zip code he can group them to be arranged in parent/child structure with add-in parent element type (address).

```
<address>
    <city>…</city>
    <street>…</ street >
    < zipcode >…</ zipcode >
</address>
```

11- For each FK (foreign key) field in the result set, the technique replaces the element type of that field by hierarchical structure that maps this element type to parent/child structure where the parent is the field element type and the children are the element types that represent the fields of the referenced record by the foreign key value in the other table participating in the relationship.

```
<emp>
    <name>…</name>
    <age>…</age>
    <dept>1</dept>   //FK field
    …
<emp>
```

The previous document can be mapped using the metadata information and the application of the technique to the following structure:

```
<emp>
    <name>…</name>
    <age>…</age>
    <dept>
        <name>…</name>
        <Location>…</Location>
        ….
    </dept>
    ….
<emp>
```

12- The user can restructure the result set to the multi-occurrence of the value of one field of the result set according to the value of other field.

```
<result>
<row1>
        <name>john</name>
        <subject>math</subject>
        <points>90</points>
</row1>
<row2>
        <name>john</name>
        <subject>physics</subject>
        <points>95</points>
</row2>
<row3>
        <name>john</name>
        <subject>DB</subject>
        <points>97</points>
</row3>
</result>
```

The previous XML document can be restructured in the following way to group the tags of subject and points as marks according to the value of name. After applying this grouping the document will appear in the following way:

```
<result>
<row1>
        <name>john</name>
        <marks>
            <subject>math</subject>
            <points>90</points>
            <subject>physics</subject>
            <points>95</points>
            <subject>DB</subject>
            <points>97</points>
        </marks>
</row1>
</result>
```

10. Conclusion

XML is a fast emerging technology, which is the standard of data exchange over the web and due to the new requirements emerging from several applications there is a great need for the integration between XML and Database technology. A lot of work has been done in the directions of mapping and conversion between XML document and Relational database.

This paper focuses on generating an XML document from relational data. It uses the available metadata information about the relational schema to generate enhanced XML document structure using the XML schema mapping alternative, which is the most promising alternative for modeling XML documents. This supports a better mapping of the XML document to relational schema, which is to be considered for future research.

References

[1] Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, 2000.

[2] Bourret, R.: XML and Databases. Technical University of Darmstadt, http://www.informatik.Tudarmstadt.de/DVS1/staff/bourret/xml/XML And Databases.htm, November, 2000

[3] Bourret, R., Bornhvd, C., Buchmann, A.P.: A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. 2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS), San Jose, California, June, 2000.

[4] Deutsch, A., Fernandez, M., Suciu, D.: Storing Semi structured Data in Relations. Workshop on Query Processing for Semi structured Data and Non-Standard Data Formats, Jerusalem, Jan., 1999.

[5] Fernandez, M., Tan, W-C., Suciu, D.: SilkRoute: Trading between Relations and XML. 9thInt. World Wide Web Conf. (WWW), Amsterdam, May, 2000.

[6] Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World Wide Web: A Survey. ACM SIGMOD Record, Vol. 27, No. 3, September, 1998.

[7] Kappel, G., Kapsammer, E., Rausch-Schott, S., Retschitzegger, W.: X-Ray - Towards Integrating XML and Relational Database Systems. Proc. Of the 19th Int. Conf. on Conceptual Modeling (ER), LNCS 1920, Springer, Salt Lake City, USA, Oct.,2000 .

[8] Kappel, G., Kapsammer, E., Retschitzegger, W.: Architectural Issues for Integrating XML and Relational Database Systems – The X-Ray Approach. The XML Technologies and Software Engineering Workshop (XSE 2001) of the 23th Int. Conf. On Software Engineering, Toronto, Canada, May 2001.

[9] Lee, D., Chu, W. W.: Constraints-preserving Transformation from XML Document Type Definition to Relational Schema. Proc. of the 19 Th Int. Conf. on Conceptual Modeling (ER), LNCS 1920, Springer, Salt Lake City, USA, Oct., 2000.

[10] Schmidt, A. R., Kersten, M. L., Windhouwer, M. A., Waas, F.: Efficient Relational Storage and Retrieval of XML Documents. Workshop on the Web and Databases (WebDB), Dallas, May, 2000.

[11] Shanmugasundaram, J., et al.: Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. of the 25 Th Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, 1999.

[12] Widom, J.: Data Management for XML –Research Directions. IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999.

[13] World Wide Web Consortium (W3C): XML 1.0 Specification, http://www.w3.org/TR/2000/REC-xml