FROM XML SCHEMA TO OBJECT-RELATIONAL DATABASE – AN XML SCHEMA-DRIVEN MAPPING ALGORITHM

Irena Mlynkova, Jaroslav Pokorny
Charles University, Faculty of Mathematics and Physics, Department of Software Engineering
Malostranske nam. 25, 118 00 Prague 1, Czech Republic
{mlynkova,pokorny}@ksi.ms.mff.cuni.cz

ABSTRACT

Since XML becomes a crucial format for representing information, it is necessary to establish techniques for managing XML documents. A possible solution can be found in storing XML data in (object-)relational databases. For this purpose most of the existing techniques often exploit an XML schema of the stored XML data, usually expressed in DTD. But the more complex today's applications are, the more insufficient the DTD becomes and the necessity to use XML Schema language becomes more essential.

The paper proposes an algorithm for mapping XML Schema structures to an object-relational database schema (defined by the SQL:1999 standard) using a (modified) DOM interface and an algorithm for storing the valid XML data into relations of the resulting schema. The main aim is to exploit object-oriented features XML Schema has and the advantages of object-relational databases and to preserve the structure as well as semantic constraints of the source schema in the target schema.

KEYWORDS

XML Schema transformation, object-relational database, mapping algorithm, DOM graph

1. INTRODUCTION

Recently XML (Bray et al., 2000) quickly becomes a key format for representing and exchanging information. But the growing usage of XML technologies brings an essential demand for effective management of XML documents and querying the data. A possible solution can be found in storing XML data in (object-)relational ((O)R) database systems. This idea results especially from many database features (e.g. query languages, schemes, programming interfaces, etc.) XML technologies include. In addition, connecting XML and (O)RDBMS enables to provide XML with missing database mechanisms (e.g. indexes, transactions, multi-user access, etc.).

Nowadays, there are several techniques for managing XML data using (O)RDBMS, whereas most of them exploit an existing XML schema[1] of the stored XML documents, usually expressed in DTD. The popularity of DTD results from its simplicity and from (usually) satisfactory expressive power. But as the requirements for exact expression of the allowed structure of XML documents grow, the DTD becomes more insufficient and the necessity to use W3C recommended XML Schema (Thompson et al., 2001; Biron & Malhotra, 2001) becomes more essential. Although XML Schema is much more complex and thus can seem to be difficult for learning, it contains many useful features (e.g. simple and complex data types, user-defined data types, precisely defined numbers of occurrences, etc.) that DTD lacks and thus gives the users more powerful tool.

This paper proposes an algorithm for mapping XML Schema structures to OR database schema defined by the SQL:1999[2] standard (Melton & Simon, 2002; Melton, 2003). The choice of the type of the target

[1] It is necessary to distinguish between words "XML schema" (i.e. schema of XML documents expressed in any language, e.g. DTD, XML Schema, etc.) and "XML Schema" (i.e. one of the languages).
[2] Latterly the SQL:2003 standard is at disposal. Except for few side points it does not differ from SQL:1999 in considered features.

schema results from many object-oriented features XML Schema has (e.g. user-defined data types, inheritance, substitutability, etc.) and the existence of corresponding OR elements for most of them. The main aim of the algorithm is to preserve the structure as well as semantic constraints of the source XML schema in the target OR schema and to exploit the advantages of OR features (i.e. user-defined data types, typed tables, references, nesting, etc.). The proposed algorithm also exploits another XML technology – the DOM programming interface (Wood et al., 1998). The idea is based on the fact, that XML schema expressed in XML Schema language is at the same time an XML document and thus can be processed using the DOM interface as well. For the purpose of the mapping algorithm a modification of the DOM tree, so-called DOM graph, is defined.

The paper is structured as follows: Section 2 briefly sums up the related works and the basic characteristics of the proposed algorithm. Section 3 describes the algorithm itself in detail. Section 4 introduces its prototype implementation called XMLSchemaStore and presents an example of the storage strategy. Finally, conclusions and future works are provided in Section 5.

2. RELATED WORKS

As mentioned above, there is a significant amount of techniques for managing XML data using (O)R databases. Generally these techniques can be classified according to several different criteria.

The basic classification is obviously related to dividing XML documents according to their content, structure, and supposed use into *data-centric* and *document-centric* (Bourret, 2003). The methods can be then classified according to the type of documents for which they were primarily designed. Most of the existing ones focus on data-centric documents with few document-centric extensions (e.g. preserving the sibling order of elements, preserving mixed-content elements, etc.).

Another classification (Amer-Yahia & Fernandez, 2001) results from the basic ideas of the mapping methods and consists of three classes – *generic*, *schema-driven* and *user-defined*. Generic methods do not use any schema of the stored documents and enable to store any XML document regardless its structure. On the contrary, schema-driven methods are based on an existing XML schema of the stored documents, that is mapped to an (O)R database schema, into whose relations the data from valid XML documents are then stored. Finally, user-defined methods, which are used mostly in commercial systems, are based on user-defined mapping.

Schema-driven mapping methods can be further divided either according to the source schema (i.e. DTD, XML Schema, etc.) or the target schema (i.e. relational or OR). From another point of view, these methods can be divided into so-called *fixed* and *flexible* (Amer-Yahia & Fernandez, 2001). Fixed methods do not use any other information than the source schema itself and the mapping is straightforward. On the other hand, flexible methods use additional information (e.g. query statistics, element statistics, etc.) and focus on creating an optimal schema for a certain application.

A more comprehensive discussion and overview of the existing methods can be found in Mlynkova & Pokorny, 2003.

The proposed algorithm belongs to the class of fixed schema-driven mapping algorithms. As was already mentioned the source schema is expressed in XML Schema, the target schema is OR. Much like the most of existing methods, the algorithm is primarily designed for data-centric XML documents, but also includes several document-centric extensions. Despite the fact, that the number of existing techniques is high, the proposed algorithm brings several not very common ideas – the focus on complex XML Schema structures (especially their object-oriented features and the semantic constraints), the exploitation of OR features (in this case user-defined types, nesting and references) and modelling XML schema by the DOM graph.

3. FROM XML SCHEMA TO OBJECT-RELATIONAL SCHEMA

This section describes the whole mapping algorithm in detail. First, the mapping rules for XML Schema structures are summed up and briefly discussed. Second, the auxiliary modification of the DOM tree – the DOM graph – is formally defined and its features are discussed. Finally, the mapping algorithm based on the DOM graph and the algorithm for storing XML data are described.

3.1 Mapping rules

The mapping rules for particular XML Schema structures are established with respect to preserving the structure and semantic constraints of the source XML schema in the target OR schema. They are summed up in Table 1, the algorithm for creating the OR schema, which follows the rules, is described in Section 3.3.

Table 1. Overview of the mapping rules

XML Schema item		Object-relational mapping
Built-in simple type	Single-valued	A corresponding SQL simple type (e.g. `TIMESTAMP` for `dateTime`, `VARCHAR` for `string`, etc.), eventually with corresponding integrity constraint (e.g. `CHECK (Column > 0)` for `positiveInteger`, etc.)
	Multi-valued	An array of corresponding SQL simple types (e.g. `VARCHAR ARRAY[N]` for `NMTOKENS`, etc.)
User-defined simple type derived by	Restriction	An SQL simple type together with corresponding integrity constraint (e.g. `CHECK (LENGTH(Column) >= N)` for `minLength`, `CHECK (Column IN [E1,...,En])` for `enumeration`, etc.)
	List	An array of corresponding SQL simple types
	Union	A sufficiently general SQL simple type, e. g. `VARCHAR`
Complex type		An SQL user-defined type (UDT)[3] *Com* consisting of attributes, which correspond to complex type's XML attributes and to its content type
Attribute		An attribute of *Com* with corresponding simple type
Content type of complex type	Simple content	An attribute of *Com* with corresponding simple type
	Extension	Inheritance of UDTs
	Restriction	Ignored – currently there is no possible mapping of this feature
	Sequence of elements	An auxiliary UDT *Seq*: Each item in the sequence is also mapped to its UDT *Item*. The sequence-item relationship is mapped to an attribute of *Seq*, whose type is determined according to the type of the item: • Globally defined → according to the maximum occurrence of the item either a reference or an array of references to *Item* (Instances of *Item* are stored into a common typed table.) • Locally defined with maximum occurrence of 1 → *Item* (Instances of *Item* are stored into the typed column of *Seq*.) • Locally defined with maximum occurrence > 1 → like globally defined items with maximum occurrence > 1 The element-sequence relationship is mapped like the sequence-item relationship, i.e. depending on the maximum occurrence and the type of the whole sequence.
	Set of elements	Like in the previous case together with additional attributes for storing the ordinal numbers of the elements
	Choice of elements	Using inheritance and substitutability: All UDTs of the OR schema are derived from a common ancestor *Anc*. The choice of elements is mapped to a UDT *Choi* having one attribute, whose type is a reference to *Anc* (without the `SCOPE` constraint) and thus can refer to any UDT.
	Model group	Like its content (The difference is, that model groups are defined globally and thus are always stored into common typed tables.)
Element		A UDT corresponding to its type (Instances of the UDT are stored according to the type of the element and its maximum occurrence either into a common typed table or into a typed column of its parent element.)
Root element		Like an element having a complex type without attributes, whose content type corresponds to a choice of (globally defined) elements

[3] The idea to use UDTs (in combination with user-defined mapping) was already exploited in DBMS Oracle9i Release 2. But the default schema used there is relatively simple and more complex structures must be created just using the user-defined mapping.

As mentioned above, the proposed algorithm focuses on data-centric XML documents, but also include two document-centric extensions – preserving the order of sibling elements and the mixed content of elements. The sibling order is preserved partly naturally using arrays and their indexes[4] (in case of multiple-occurrence elements) and partly unnaturally using additional attributes(s) keeping the order information (in case of set of elements). On the other hand, the mixed content is preserved by storing the mixed-content elements as other elements and their text parts in a separate multi-valued property.

There are also XML Schema structures, which cannot be mapped at all or could be mapped just in special cases. They include:

Identity constraints. Identity constraints (i.e. `unique`, `key` and `keyref` elements) can be considered as an XML Schema generalization of (originally DTD) simple types `ID`, `IDREF` and `IDREFS`. Most of the present DTD-driven algorithms map these types (or strictly speaking their features) to SQL keys and foreign keys. But this mapping is not precise, since the DTD uniqueness refers to the whole XML document while the SQL uniqueness just to one table. Thus an enough suitable mapping even of their generalization is impossible.

Wildcards. The idea of wildcards enables to store any kind of element or attribute at a certain place. Thus this feature can be mapped only to an enough general SQL type (e.g. `VARCHAR`). But in this case a further exploitation of the stored data is quite small or must be solved using any XML-aware text enhancements.

Substitution groups. Substitution groups can be considered as probably one of the most powerful tool among all XML Schema structures. They can be mapped analogous to a choice of elements, but this mapping does not express their features precisely.

Furthermore, there are few XML Schema structures, which have no significance for the structure of the schema (i.e. notations and annotations) or whose mapping is not quite essential (i.e. external schemes).

3.2 DOM graph

As mentioned before, the proposed mapping algorithm, that follows the previously defined mapping rules, is based on a modification of the DOM tree called *DOM graph*. The idea results from the fact, that XML schema expressed in XML Schema language is at the same time an XML document and thus can be processed using the DOM interface as well. This section contains a formal definition of the graph and a discussion of its properties.

Definition 1. Let $T_{DOM} = (V_T, E_T)$ be an (undirected) DOM tree of the given XML schema. Let $Glob^T_{type} \subseteq V_T$ be a set of globally defined simple and complex types (i.e. `simpleType` and `complexType` element nodes, which are direct subelements of `schema` root element node). Let $Glob^T_{item} \subseteq V_T$ be a set of globally defined elements, model groups, attributes, and attribute groups (i.e. `element`, `group`, `attribute` and `attributeGroup` element nodes, which are direct subelements of `schema` root element node). *DOM graph* corresponding to T_{DOM} is a directed graph $G_{DOM} = (V_G, E_G)$, where

$V_G = V_T$ and

$E_G = \{(v_x, v_y) \mid \{v_x, v_y\} \in E_T \wedge$ (element node v_y is a subelement of element node v_x)$\} \cup$
$\{(v_x, v_y) \mid \{v_x, v_y\} \in E_T \wedge$ (attribute node v_y represents an attribute of element node v_x)$\} \cup$
$\{(v_x, v_y) \mid v_x$ is `base`, `type` or `itemType` attribute node referencing to a node $v_y \in Glob^T_{type}\} \cup$
$\{(v_x, v_y) \mid v_x$ is `ref` attribute node referencing to a node $v_y \in Glob^T_{item}\}$

The DOM graph hence consists of nodes of the original DOM tree and two kinds of edges – directed edges of the original DOM tree and additional edges expressing the "direction" of the usage of globally defined items or data types. An example of an XML Schema file is depicted in Figure 1, its DOM graph in Figure 2. The solid lines (without ordering) correspond to edges of the original DOM tree; dash-and-dot lines are the additional ones. The letters in parentheses express element (E) or attribute (A) nodes.

[4] If the preserving of sibling order is not important, the new SQL:2003 type `MULTISET` can be used for unordered XML data.

```
<xs:schema xmlns:xs="http://w w w .w 3.org/2001/XMLSchema">

  <xs:element name="Staff">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Person"  maxOccurs="1000">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Name"/>
            </xs:sequence>
            <xs:attribute name="OnHoliday" type="YesNo"
                          use="required"/>
            <xs:attribute name="Note"  type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Name">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ForeName"  type="xs:string"
                    maxOccurs="5"/>
        <xs:element name="Surname"  type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="YesNo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Yes"/>
      <xs:enumeration value="No"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Figure 1. An example of an XML Schema file

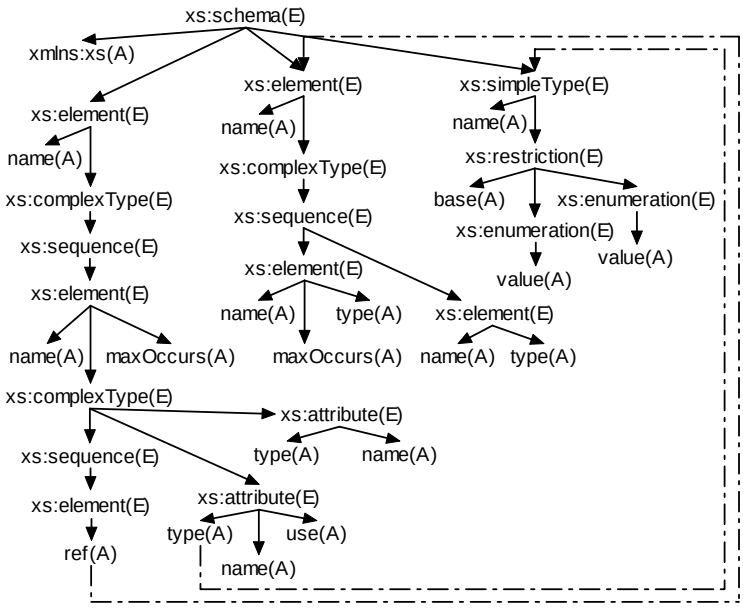

Figure 2. An example of a DOM graph of the XML schema depicted in Figure 1

3.2.1 Cycles in the graph

The W3C XML Schema recommendation allows several kinds of cyclic definitions of elements. The cycles rise from mutual usage of globally defined elements or complex types and result in cycles in the DOM graph.

So far, the SQL:1999 standard allows no cyclic definitions of UDTs. Nevertheless, certain kinds of cycles would be probably included in future versions of the standard. On the other hand, some of the existing (O)RDBMS support "natural" cyclic definitions of UDTs. For instance, DBMS Oracle9i Release 2 enables to use so-called *incomplete types*, which can be considered as a forward declaration. An incomplete type is defined without its content and can be used instead of the corresponding complete type (except for the inheritance ancestor). Its definition is completed only when all necessary types exist.

Using incomplete types it is possible to process cycles in the DOM graph. Before processing the DOM graph, an incomplete type is created for each globally defined complex type/element. The incomplete types are then used instead of the corresponding complete types and their definitions are completed after all necessary types exist. Since W3C XML Schema recommendation does not allow cycles among complex type extensions, there is always at least one edge in each cycle, which does not represent the extension of complex types and just this edge enables to "break" the cycle using an incomplete type.

3.3 Mapping algorithm

Using the above-defined structure of the DOM graph the mapping process becomes relatively simple and obvious. It has two parts – creating the DOM graph and its processing. The result of the processing is an OR schema, which consists of a set of typed tables "interconnected" using references. The algorithm can be summed as follows:

Creating the DOM graph $G_{DOM} = (V_G, E_G)$:
- Create a DOM tree $T_{DOM} = (V_T, E_T)$ of the given XML schema;
- **For each** node $v_T \in V_T$ **do** create the corresponding node $v_G \in V_G$;
- **For each** (unordered) edge $e_T \in E_T$ **do** create the corresponding (ordered) edge $e_G \in E_G$;
- Let $Glob^T_{type} \subseteq V_T$ and $Glob^T_{item} \subseteq V_T$ be defined like in Definition 1;
- Let $Glob^G_{type} = \{v_G \mid v_G \in V_G \wedge v_G$ corresponds to $v_T \in Glob^T_{type}\}$ and $Glob^G_{item} = \{v_G \mid v_G \in V_G \wedge v_G$ corresponds to $v_T \in Glob^T_{item}\}$;
- **For each** `base`, `type`, `itemType` or `ref` attribute node $a \in V_G$ referencing to node $g \in Glob^G_{type} \cup Glob^G_{item}$ **do** create the corresponding (ordered) $e = (a, g) \in E_G$;

Processing the DOM graph G_{DOM}:
- Mark each $v \in V_G$ as unprocessed;
- Let $Glob^G_{elem} \in V_G$ be the set of globally defined elements;
- **For each** $v \in Glob^G_{type} \cup Glob^G_{elem}$ **do begin**
  - Create its incomplete UDT;
  - Mark v as processed;
- **end**
- *ProcessNode*(`schema` root node $\in V_G$);
- **For each** $v \in Glob^G_{type} \cup Glob^G_{elem}$ with an incomplete UDT **do** create its complete UDT;
- **For each** element, which is not mapped to a typed column **do** create a typed table (including corresponding column integrity constraints);
- **For each** reference (except for those corresponding to a choice of elements) or array of references **do** create a corresponding `SCOPE` integrity constraint;

procedure *ProcessNode*($v \in V_G$)
- Let V_{child} be the set of child nodes of v;
- **For each** $v_{ch} \in V_{child}$ **do begin**
  - **If** v_{ch} is marked unprocessed **then** *ProcessNode*(v_{ch});
  - **Else if** $(v, v_{ch}) \in E_G$ expresses an extension of $v_{ch} \in Glob^G_{type}$ and v_{ch} has an incomplete UDT **then begin**
    - *ProcessNode*(v_{ch});
    - Create its complete UDT;
  - **end**
  - **Else if** v is a `schema` root node and v_{ch} has an incomplete UDT **then** *ProcessNode*(v_{ch});
- **end**
- **If** v is marked unprocessed **then** process v according to the established mapping rules;
- Mark v as processed;

The above-described order in which the SQL items of the target schema (i.e. UDTs, typed tables, references and their constraints) are created is established so, that it follows the SQL rules and considers possible cycles in the graph. If the DOM graph contains no cycles, the algorithm could be simplified.

3.4 Storing XML documents

This section describes the algorithm for storing the data from XML documents valid against the source XML schema into relations of the target OR schema. It is based on traversing the DOM tree of the XML document and creating the corresponding SQL constructors of the given data.

The algorithm can be summed as follows:

Create a DOM tree $T_{DOM} = (V_T, E_T)$ of the given XML file;
c_v = *StoreNode*(root node $v \in V_T$);
Use c_v for storing the data into corresponding typed table;

function *StoreNode*($v \in V_T$):string
Let V_{child} be the set of child nodes of v;
Let $C = \{c_i \mid c_i = StoreNode(v_i); v_i \in V_{child} \wedge i = 1,...,|V_{child}|\}$ be a set of constructors of V_{child};
Create the constructor c_v of the node v from $\forall\ c \in C$;
If v is mapped to a reference or an array of references **then begin**
Use c_v for storing the data into corresponding typed table;
Create a constructor c_{ref} of a reference to the stored data;
$c_v = c_{ref}$;
end
Return c_v;

The whole storage process is driven not only by the stored XML document, but also by auxiliary information about the current OR schema (i.e. types of parent-child mapping, names of the tables, names of the UDTs, structure of UDTs determining the constructors, etc.). This information is stored into auxiliary tables during the mapping process.

4. PROTOTYPE IMPLEMENTATION

A prototype implementation of the proposed algorithm is called XMLSchemaStore (Mlynkova, 2003). It is based on DBMS Oracle9i Release 2, which supports a lot of SQL:1999 OR features, especially those, which were exploited in the proposed algorithm. As there are few syntactical differences between Oracle9i SQL and SQL:1999 standard, the algorithm was slightly adapted to Oracle9i features. The features, which are not supported in the system yet, were in XMLSchemaStore omitted.

The implementation enables:

- to create an OR schema according to a given XML schema using the proposed algorithm,
- to store any valid XML document into relations of the corresponding OR schema, and
- to process (a subset of) XML path queries over the stored XML data.

The last mentioned feature is not discussed here for the paper length. In short, the implemented algorithm enables to map path queries over XML documents to SQL queries over the OR schema and to convert the SQL result back to an XML document.

An example of an XML document valid against XML schema depicted in Figure 1 and its storage in the resulting OR schema is depicted in Figure 3. The arrow keys represent references; the square brackets represent arrays; the dot notation represents nested attributes.

5. CONCLUSION

This paper proposed an algorithm for mapping XML Schema structures to an object-relational database schema and an algorithm for storing the valid XML data into the resulting relations. For the purpose of the algorithm a modification of the DOM tree, so-called DOM graph, was defined to determine and at the same time to simplify the mapping procedure. In contrast to currently existing methods, the algorithm focused on object-oriented features XML Schema has and the advantages of object-relational databases.

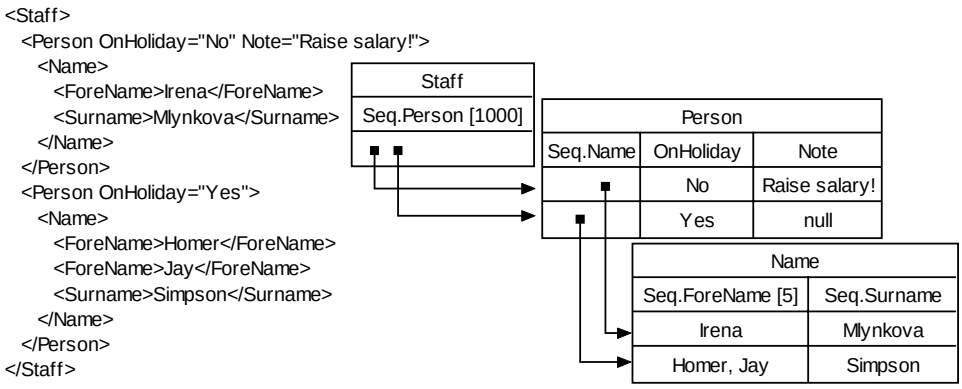

Figure 3. An example of an XML document and its storage in resulting OR schema

The algorithm is considered as a basis for future work, which should focus on optimalizations of the created schema. First possible approach can be based on so-called flexible mapping methods (Bohannon et al., 2002; Klettke & Meyer, 2000), which try to establish an optimal schema for a certain application. Second possible approach can result from the necessity to determine a definition of a "good" XML schema (such as e.g. normal forms for relations) and ways how to establish it. The reason is, that as there are no rules, which define a "good" XML schema, a fixed mapping of a "bad" one can result in a "bad" relational schema as well.

ACKNOWLEDGEMENTS

This work was supported in part by the National programme of research (Information society project 1ET100300419).

REFERENCES

Amer-Yahia, S. and Fernandez, M., 2001. *Overview of Existing XML Storage Techniques.* AT&T Labs, Cambridge, UK.

Biron, P. V. and Malhotra, A., 2001. *XML Schema Part 2: Datatypes*. W3C Recommendation, http://www.w3.org/TR/xmlschema-2/

Bohannon, P. et al., 2002. From XML Schema to Relations: A Cost-Based Approach to XML Storage. *Proceedings of ICDE Conference*, San Jose, California, p. 64.

Bourret, R., 2003. *XML and Databases*. www.rpbourret.com

Bray, T. et al., 2004. *Extensible Markup Language (XML) 1.0 (Third Edition).* W3C Recommendation, http://www.w3.org/TR/REC-xml/

Klettke, M. and Meyer, H., 2000. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. *Informal Proceedings of WebDB Workshop*, Dallas, Texas, pp 151 - 170.

Melton, J., 2003. *Advanced SQL: 1999 – Understanding Object-Relational and Other Advanced Features*. Morgan Kaufmann Publishers, San Francisco, USA.

Melton, J. and Simon, A. R., 2002. *SQL: 1999 – Understanding Relational Language Components*. Morgan Kaufmann Publishers, San Francisco, USA.

Mlynkova, I., 2003. *XML Schema and its Implementation in Relational Databases*. Master thesis, Charles University, Prague, Czech Republic, http://kocour.ms.mff.cuni.cz/~mlynkova/doc/dip2003.pdf (In Czech)

Mlynkova, I. and Pokorny, J., 2003. *XML in the World of (Object-) Relational Database Systems*. Technical report 2003/8, Charles University, Prague, Czech Republic, http://kocour.ms.mff.cuni.cz/~mlynkova/doc/tr2003-8.pdf

Thompson, H. S. et al., 2001. *XML Schema Part 1: Structures*. W3C Recommendation, http://www.w3.org/TR/xmlschema-1/

Wood, L. et al., 1998. *DOM Level 1 Specification.* W3C Recommendation, http://www.w3.org/TR/REC-DOM-Level-1/