Proceedings of Informing Science & IT Education Conference (InSITE) 2012

Implementation of the Tree Structure in the XML and Relational Database

Aleksandar Bulajic
Faculty of Information Technology,
Metropolitan University,
Belgrade, Serbia

aleksandar.bulajic.1145@fit.edu.rs

Nenad Filipovic
Faculty of Engineering Science,
University of Kragujevac,
Kragujevac, Serbia

nfilipov@hsph.harvard.edu

Abstract

A tree structure is a powerful tool for storing and manipulating all kind of data, without differences is it used for natural language analysis, compiling computer languages or storing and analyzing scientific or business data. Algorithms for searching and browsing tree structures are well known, and for free are available numbers of tools for manipulating tree structures implemented in different computer languages. Even is impossible to avoid these algorithms and tools, primary focus in this paper is XML and XML tools. Storing and retrieving, as well as validation of XML tree structures in relational database can be a challenge. This document describes these challenges and currently available solutions.

Keywords: Tree structure, XQuey, XML, XML Parser, XSLT, XML Schema, DOM, SAX, Database, SQL.

Introduction

Tree structure in computer science is a data structure used for representing hierarchical data structures. Oxford Dictionary (Oxford Advanced Learner's Dictionary 2011) describes hierarchy as "*a system, especially in a society or an organization, in which people are organized into different levels of importance from highest to lowest*" and Wikipedia describes hierarchy as "*Greek: hierarchia (ἱεραρχία), from hierarches, "leader of sacred rites") is an arrangement of items (objects, names, values, categories, etc.) in which the items are represented as being "above," "below," or "at the same level as" one another*." These definitions are not sufficient because a Tree structure requires also that each node has only one parent and can have zero or more children, and is an acyclic connected graph. Wikipedia defines a node as "*a structure which may contain a value, a condition, or represent a separate data structure (which could be a tree of its own)*".

Definition can be developed further and each definition would contain a new terms that shall be further defined. That is out of scope of this paper. Focus in this paper is on the Extensible Markup Language (XML) documents and storing/retrieving XML documents to/from relational database.

The “Terminology and Examples of Tree Structures” section defines basic terms and concepts and presents different kind of tree structures.

The “XML” and “XML Parsers” section introduce basic XML concept and tools for XML document validation and manipulation.

The “XML Working Groups” and “XML Schema Working Group” sections introduce World Wide Web Consortium (W3C) international groups that are responsible for developing XML standards as well as developing of Web standards.

The “XSLT Working Group” introduces W3C group for standardization of transformation of XML content to other formats.

The “XQuery Working Group” section introduces XQuery language for combining unstructured, semi-structured and structured data that are very important for Online Analytical Processing (OLAP), Data warehousing and generally speaking Business Intelligence (BI) that analyses past, current and future of business operations and business performances.

The “XML and database” section introduces solutions that are implemented by leading database vendors such as Oracle, Microsoft and IBM.

The “Conclusion” section contains short discussion and summarizes authors’ opinions about presented technologies. Language and Grammar

All papers are to be written in English. If you have questions on English grammar or punctuation, an excellent guide can be found at http://grammar.ccc.commnet.edu/grammar

While U.S. spelling is preferable, other versions of English are acceptable.

Terminology and Examples of Tree Structure

Following is a widely accepted tree structure terminology:

1. Root node – the topmost node in a tree without a parent,
2. Parent node – a node that has a child or children,
3. Child – a node that has a parent node
4. Sibling – nodes that share the same parent node.
5. Leaf node – a node that has not any child node.

Figure 1 illustrates some examples of the Tree structure:

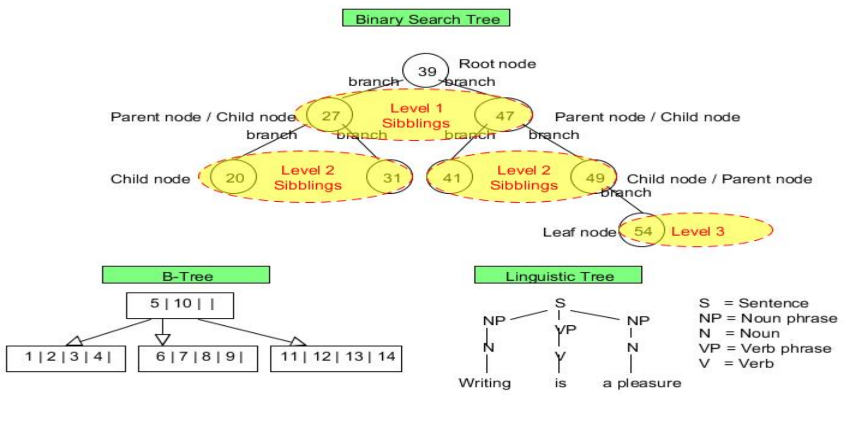


Figure 1. Tree structure examples

These pictures show all important elements and illustrate terms that are used in this document. The Root node is only node that does not have a parent. Each other sub node has a parent. Nodes that are called the Parent nodes are the Child nodes of the Root node. Leaf node is a node that contains some data and should not be treated differently than any other Child node.

Figure 1 contains common interpretations of Tree structure. However, different problems can require different representation and notation. One example is the New Hampshire format, also referred as Newick format, that uses commas and parentheses to define pairs of nodes to be displayed as connected. The length of a branch can be incorporated next to a node name followed by a colon. (Pavlopoulus et al. 2010) Following is an Newick format example:

`(B,(A,C,E),D);`- all nodes are named

(A:0.1,B:0.2,(C:0.3,D:0.4):0.5); - *distances and leaf names*

This format is used for tools like PHYLogeny Inference Package (PHYLIP), *a free package of programs for inferring evolutionary trees* (Department of Genome Sciences and Department of Biology 2011) and handling data types such as molecular sequences (proteins, DNA and RNA), *gene frequencies, restriction sites and fragments, distance matrices, and discrete characters*. (Department of Genome Sciences and Department of Biology 2011)

Other examples of Tree format notation is a DOT-language (GRAPHWIZ.ORG 2011) used by Graph Visualization (GRAPHWIZ) software at AT&T Labs Research for visualization of undirected and directed graphs. Currently very popular and most used language for describing Tree structures is XML. Documents that are described by XML language are by default machine and human beings readable.

The most common operations on a Tree structure are:

1. Searching,
2. Inserting,
3. Deleting,
4. Enumerating,
5. Moving (changing parent node).

Implementation of efficient search algorithms requires an ordered Tree structure. This means that all nodes are sorted according to search criteria. Traversing a Tree structure can be executed as:

1. Pre-ordered – when parent node is visited before their child, nodes,
2. Post-ordered – when children are visited before parent node,
3. In-order - when first left child node is visited and then parent node and then right child node is visited.

Tree structure Insert and Delete operations are damaging balanced Tree structure and can increase search time. Reorganizing a Tree structure by choosing another root node and moving all other nodes accordingly makes a Tree structure balanced and improves algorithm performances. Figure 2 illustrates this example of tree structure where balance has been damaged by inserting new nodes and rebalanced tree structure by choosing another root node:

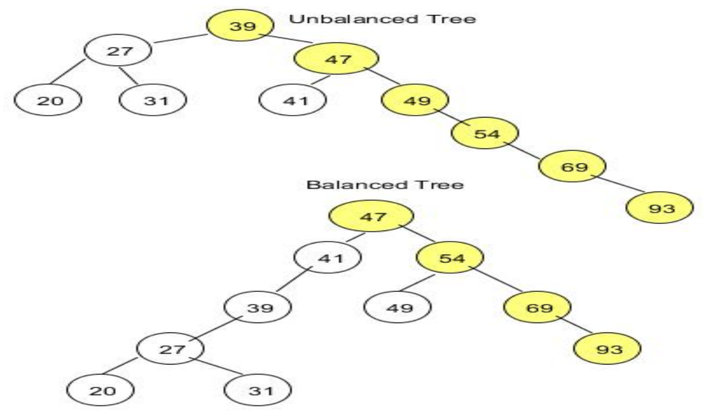


Figure 2. Unbalanced and balanced Tree structure

While self-balanced trees, such as AVL tree, Red-black tree or AA tree are doing balancing incrementally, Day/Stout/Warren (DSW) algorithm is balancing a binary search tree periodically, when it is not expected more insert operations, and reduces costs related to balancing operations.

Often balancing of tree structure will cause slower insertion and removal operations, but faster search and retrieval. One example is AVL tree that is constantly balanced and provide guaranteed worst-case running time that is proportional to O(log n) time for search, insert and delete operations (Sedgewick 2008).

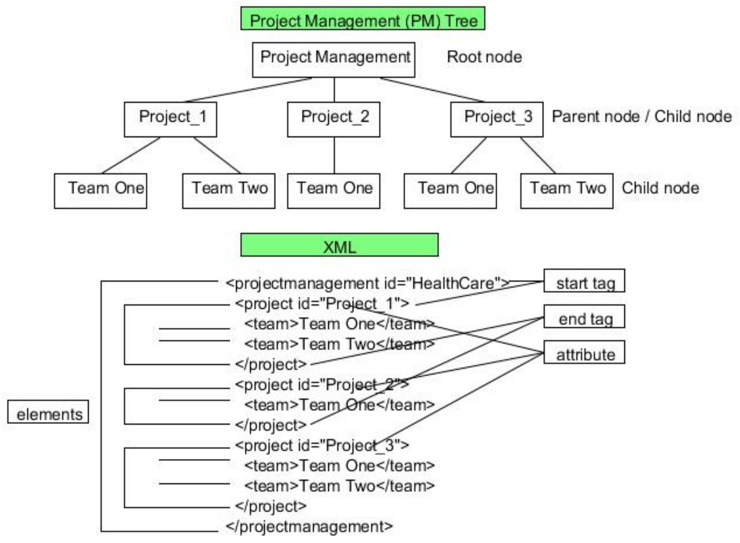
Other tree structures, as for example Splay tree, are optimized to provide quick access to recently accessed nodes. Splay tree structure implements operation called splaying that brings recently accessed elements close to the root node. During each operation is applied a simple restructuring rule that simultaneously search and reorganize a tree structure and improve the efficiency of future operations The restructuring can be done by applying tree rotation operation or move to root algorithm or bottom-up algorithm, where accessed item is moved bottom-up all the way to the root.(Sleator and Tarjan 1985). There are many practical implementation where quick access to frequently used nodes is required, as for example for cashes and garbage collections algorithms, and data compression and lexicographic and multidimensional analysis algorithms. The splay tree structure is used when the total time of sequences of operations is important, but not the individual times of the operations (Sleator and Tarjan 1985).

Scapegoat tree is a binary search tree structure optimized to avoid per-memory node overhead and do not require keeping of extra data. Each node contains only a key value and pointers to child nodes. The two extra values, the number of nodes in the whole tree and the max number of nodes in the tree since the tree was last completely rebuilt, are associated with the root of the whole tree (Galperin and Rivest 1998).

Databases and file systems are using B-tree. B-tree structure allows that each node can have more than two child nodes and this is most notable difference from a binary tree structure. Figure 1 illustrates one example of B-tree. B-tree structure is always sorted and keeps depth of tree structure at minimum, to enable quick search operation. A B-tree is always sorted and can be used for sequential access to nodes

While in case of a Binary Search Tree or a B-Tree, each node usually contains the same kind of data, data structure and data type, in case of other Tree structures content of each node can be different.

The next example demonstrates such case. This example structure is used as an example for the rest of this paper. Figure.. 3 illustrates an organizational schema of Project Management (PM) Tree structure:



```
<projectmanagement id="HealthCare">
<project id="Project_1">
  <team>Team One</team>
  <team>Team Two</team>
</project>
<project id="Project_2">
  <team>Team One</team>
</project>
<project id="Project_3">
  <team>Team One</team>
  <team>Team Two</team>
</project>
</projectmanagement>
```

Figure 3. Project Management (PM) Tree and XML

Project Management (PM) Tree structure describes organizational structure of Project Management (PM) and involved parties that are described in sub nodes. A Tree structure is not enough to explain purpose of this structure, and usually is supported by a textual description. For example PM Tree structure in Figure 3 could contain only project names, or names of project managers and team leaders, but also the whole project team names and responsibilities. What can be concluded from above picture are Project Management hierarchy and a scope of responsibility.

Each node in this example can contain different data or data structure, but this cannot be very practical because each node in that case should contain at least a type of node indicator. Convenient solution is to implement the same data or data structure and type at each level. This means that all nodes on the Project level will have the same data, data structure and data type, and all nodes at the Team level will have the same data, data structure and data type. Data, data structure and data type at these two levels can differ, as well as data, data structure and data type on the Project Management, root level.

Moving a node is very important operation even it is not recognized as a separate operation in current literature or better to say it is assumed to be a part of balancing algorithm, as already mentioned rotation tree operation or top-down algorithm. In case of binary search tree, moving a node by changing a parent relation is acceptable and improves tree structure. In case of tree structure that contains business data, such as for example a tree structure illustrated in Figure 3, changing a parent node would damage whole structure and corrupt data quality and in some cases make tree structure useless. This of course depends of an example. In other cases, as for example in case of organizational structure, changing a parent would move all sub-nodes and responsibilities and reporting chains. This is one of the best tree structure characteristics. Never mind how large is sub-node structure; the whole structure can be moved by simple changing a parent of sub-node. This is quick and simple operation that can attach large number of nodes to another node.

While sorting operation in case of Binary Search Tree would improve search and insert, and delete performances, sorting of entire tree structure that contains business related data would probably damage tree structure irrevocably. In case of business related data, sorting operation could be implemented only on the nodes that are belonging to the same parent node, Sibling nodes or nodes that are sharing the same parent node.

XML

"*Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere*" (W3C XML 2003).

The very first what you can see in the Figure 3 XML example is that transformation from PM Tree to XML is not 1 to 1. Transformation requires identification of common elements and naming convention that uniquely identifies these elements. For example Project Management has been replaced by a tag called <projectmanagement>, and multiply projects by general tag named <project>, as well as multiply teams have been replaced by tag named <team>. This structure is complex enough for illustrating relations between parents and children. In this case Project Management has multiply Project children and each Project child has multiply Teams.

Each level in the tree structure when represented as XML needs to be generalized. For example second level in Figure 2 is tagged as <project> and level tree is tagged as <team>. Differences between projects and teams are described by Id's and XML text data.

XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML language is a subset of restricted form of the Standard Generalized Markup Language (SGML) (W3C XML 1.0 2008).

"*XML documents are made up of storage units called* entities, *which contain either parsed or unparsed data. Parsed data is made up of* characters, *some of which form* character data, *and some of which form* markup. *Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure*" (W3C XML 1.0 2008).

XML document represents a tree structure with additional information about XML version and applied characters encodings. This information is stored at the document header. Following examples illustrates XML document header:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
```

The simple XML tree structure is defined by tags, elements and attributes. Tag is a markup structure that starts with "<" and ends with ">"sign. XML makes differences between start-tag and end-tag. End-tag starts with "</" and ends with ">". Empty-tag starts with "<" and ends with "/>". Element is a logical structure that starts with start-tag and ends with end-tag. Attributes is a name and value pair that exists within a start-tag or empty-tag.

One of the most important XML terminology terms are:

1. Well-formed XML document,
2. Valid XML document.

Well-formed constraints can be described as only one root and each start tag has corresponding end tag and these are correctly nested without overlapping with others start and end tag pairs. XML specification at World Wide Web Consortium (W3C) contains a full list of other conditions that well-formed XML document shall satisfy. (W3C XML 1.0 2008) More about W3C consortium is available atW3C home page at http://www.w3.org/Consortium/.

Figure 3 illustrates well-formed XML document where is only one root element represented by a start-tag <projectmanagement> and a corresponding end-tag </projectmanagement>. Note that end-tag, even has the same name as start-tag, starts with backslash ("/") character that denotes end-tag. Each start-tag in this XML document example contains end-tag and these are not overlapping.

While well-formedness can be tested by reading and parsing an XML document, validity checking is not possible without additional information that is part of another document. For validating an XML document are defined different standards:

1. Document Type Definition (DTD),
2. XML Schema,
3. RELAX NG.

The first two standards are developed by World Wide Web Consortium (W3C). The third standard is developed by Organization for the Advancement of Structured Information Standards (OASIS). More about OASIS can be find at http://www.oasis-open.org/.

All of these three standards are used for XML documents validations. These standards describe valid XML document elements, elements structure and data types. While DTD is an old standard and offers a very limited data types, XML Schema, newer standard, offers extended data types and Object Oriented approach. While XML Schema syntax can be very complex, RELAX NG claims that is much simpler and easy to learn.

What are differences between a Tree structures used for business related data storing and Tree structures that are used for searching and sorting data? The first one very important is that a business trees are by nature self balanced. Moving nodes is not applicable because it would corrupt a structure and damage data quality. Sorting of these tree structures is not applicable too, because such operation would require moving nodes and damage data quality. Missing of optimal searching and sorting algorithms is a weak component of such tree. Effective searching needs to know about a full path to a data or traversing over a whole tree to find a correct data set. Different data nodes can contain the same data names and in this case is impossible to find out what nodes belongs to a correct data set if there is not provided a unique path to a target data set. Adding and removing nodes in Business Tree structure is quick and easy and is limited to changing pointer to a parent node.

XML Parsers

XML resources are very huge, and besides standards such as XML Schema, for XML document validation, XSL Transformation (XSLT), for XML document transformation, and XML Path (XPath) language for addressing a part of XML document, there are available XML parsers created in different computer languages for parsing an XML document and other kind of software for manipulating XML document, as well as tutorials and documentations for learning about this widely accepted standard.

XML Parsers are most important for programmatically manipulation of XML documents. XML Parser parsing a XML document elements to data structures that enables easy in memory manipulation. Memory data structures are not independent from programming languages.

There are two approaches to XML document parsing available as:

1. Simple API for XML (SAX),
2. Document Object Model (DOM).

SAX is an event-based public standard that reads an XML Document sequentially and search for a particular element. When element is located, SAX triggers an event and calls back a functions created by developer to process element data.

History of development of SAX parser starts in 1997, when Peter Murray-Rast proposed that all parsers should support a common Java event-based Application Programming Interface (API). (SAXProject 2011) API is a set of common interfaces that enable external application communications and software to software communications. After almost a year of discussion, and a lot of suggestions from different people, this initiative produced a first draft of SAX interface proposal. Jon Bosak, the founder of XML allowed using of his domain xml.org for SAX Java package that is named org.xml..sax. (SAXProject 2011) SAX parser is today available for many other programming languages and environments.

Following comments are describing an example of SAX parser implemented in Java language and parsing of a XML document described in the Figure 3. The `SAXParserImp` class extends `DefaultHandler` class. This class "*provides default implementation for all the callbacks*" (Apache Software Foundation 2011):

```
public class SAXParserImp extends DefaultHandler
```

Next important steps are creating `XMLReader` and define default content handler that are usually created inside of the implementation class constructor:

```
XMLReader r = XMLReaderFac-
tory.createXMLReader("org.apache.xerces.parsers.SAXParser");

r.setContentHandler(this);  r.setErrorHandler(this);

r.parse(new InputSource(new StringReader(xmlString)));
```

`XMLReader` is an interface for reading an XML document by using callbacks. Reading an XML document is different than reading text files and "*reader must wait for an event-handler callback to return before reporting then next event*". (Apache Software Foundation 2011)

Following callback methods are used to parse XML document described in Figure 3:

```
public void startElement(String namespaceURI, String localName, String
qName, Attributes atts) throws SAXException

public void endElement(String namespaceURI, String localName, String
qName) throws SAXException

public void characters(char[] chars, int start, int len) throws SAXEx-
ception

public void startDocument() throws SAXException

public void endDocument() throws SAXException
```

Although only first three call-backs methods are sufficient, the `startDocument()` method and `endDocument()` method can be used for parameters initialization or clean up procedures after document parsing is completed.

The `startElement` method call-back occurs when parser discovers start-tag of next element. In this method is important to note that attributes, if there is any in the XML document, should be processed here.

The real processing of element value or values occurs when call-back to `endElement` method occurs. Element values are concatenated in the `characters` method.

An XML document can be very complex and contain namespaces, processing data and entities, and links to external XML documents. Apache Xerces parser provides default implementations for each call-back. However, developer task is to override these methods and provide appropriate implementation code.

Even many are using expression "DOM Parser" or "XML DOM Parser", DOM is not a parser. DOM is a W3C standard and specification that describes an object model that provides standard set of names and functions that are used for hierarchical model description and access to nodes and node manipulation, as well as it is a standard programming interface for accessing to DOM objects programmatically by using programming language.

"*The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.*" (W3C DOM 2005)

Before XML document can be processed via DOM interface, first shall be parsed and make available for processing via DOM interface. DOM standard DOM standard has three parts:

1. DOM Core,
2. DOM XML,
3. DOM Hyper Text Markup Language (HTML).

Even DOM Working Group has been closed in 2004, after completion of DOM Level 3 Recommendations, other W3C groups overtakes maintenance and further development of standard APIs. (W3C DOM 2005)

Following code example describes XML document parsing and creating of a DOM tree:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(xmlFileName);
} catch (Exception e) {}
```

`DocumentBuilderFactory` creates an instance of DOM APIs that is able to obtain a parser for creating a Dom document tree. (Apache Software Foundation 2011)

`DocumentBuilder` parses an XML document received from different sources, such as, for example, files, input streams, or SAX input sources or URLs into an DOM document. (Apache Software Foundation 2011)

`Document` is entire XML document represented as DOM tree and a root element that is an entry point to access document root, elements, text nodes, processing instructions, attributes or namespaces and comments, or any other DOM Document elements that cannot exists outside of a DOM Document context. (Apache Software Foundation 2011)

While SAX parser parses XML document sequentially and requires minimal quantity of internal computer memory, DOM requires that entire tree is stored in the computer internal memory. In case of a huge XML document that contains large number of elements and sub-elements, this requirement can be challenging.

However, DOM tree and DOM APIs provide a lot of advantages in case of XML document querying, modifying and generally speaking, in case of any kind of XML document processing.

"*W3C's Document Object Model (DOM) is a standard Application Programming Interface (API) to the structure of documents. The DOM aims to make it easy for programmers to access components and to delete, add, or edit their content, attributes and style. In essence, the DOM makes it possible for programmers to write applications that work properly on all browsers and servers* *and on all platforms. While programmers may need to use different programming languages, they do not need to change their programming model*." (W3C DOM 2005)

XML Working Groups

W3C XML Working Groups are responsible for development of new XML recommendations and standards, as well as for specifications and maintenance of existing. Following are some of the currently existing XML Working groups (W3C XML 1.0 2000):

1. XML Core Working Group "*develop and maintain the specifications for XML itself and closely related specifications such as Namespaces in XML, the XML Information Set, and XInclude*." (W3C XML 1.0 2000)
2. XML Processing Model Working Group "*defining a scripting language for XML: that is, a way to specify what operations should be performed on an XML document and in what order*." (W3C XML 1.0 2000)
3. XML Schema Working Group "*provide mechanisms to define and describe the structure, content, and to some extent semantics of XML documents*." (W3C XML 1.0 2000)
4. Extensible Stylesheet Language Transformation (XSLT) Working Group "*responsible for XSL Transformations (XSLT) and a number of supporting specifications*." (W3C XML 1.0 2000)
5. XML Query Working Group "*working on the XML Query Language, a way to provide flexible query facilities to extract data from real and virtual XML documents on the Web. This includes publication of XQuery and also XPath, in conjunction with the XSLT Working Group (part of the Style Activity)*." (W3C XML 1.0 2000)

Before making a final specification, these groups make recommendations available for public discussions and suggestions. For example XSLT Working Group public mailing list is available to everyone at http://www.mulberrytech.com/xsl/xsl-list/ and comments to specification can be sent directly to mail address xsl-editors@w3.org..

XML Schema Working Group

XML Schema replaced a DTD, an old standard for XML document validation. Following is an example of DTD document used for validation of XML document described in Figure 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMLPMv1 [
<!ELEMENT projectmanagement (project+)>
<!ELEMENT project (team+)>
<!ELEMENT team (#PCDATA)>
<!ATTLIST projectmanagement id CDATA #REQUIRED>
<!ATTLIST project id CDATA #REQUIRED>
]>
```

This DTD is simple and easy to understand. Plus sign means that at least one or more elements of `project and team` type shall exist. `PCDATA` describes that contents of element `team` are text data and `ATTLIST` describes which element has an attribute and attribute name and data type, in this case `CDATA` data type, as well as attribute is mandatory, `REQUIRED`.

DTD can be more complex and describe choice and default values.

While DTD defines limited set of basic data types, such as PCDATA and CDATA, XML Schema defines a long list of basic types such as byte, integer, floating point, string, date, time, as well as atomic types, list types and union types or user defined types and data type validation rules. Full list of basic data types and other XML Schema data types is available at http://www.w3.org/TR/xmlschema-#simpleTypesTable at W3C Internet site.

XML Schema defines data types that support data types from relational databases, such as relational data types and objects. User-defined data types are derived from existing data. In case of derived data, it is possible to restricted derived data by range, precision, length or format. XML Schema validation constrains depends of data type. For example string data type validation can be length, max length, min length, enumeration, pattern or whitespace, while Boolean data type can be only validate on pattern and whitespace. Pattern can contain a regular expression for data type validation. Following is an example of XML Schema that validates XML document described in Figure 3:

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema attributeFormDefault="unqualified" elementFormDe-
fault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="projectmanagement">

    <xs:complexType> <xs:sequence>

        <xs:element maxOccurs="unbounded" name="project">

          <xs:complexType> <xs:sequence>

              <xs:element minOccurs="0" maxOccurs="unbounded"
name="team" type="xs:string" />

            </xs:sequence>

            <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType> </xs:element>

      </xs:sequence>

      <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType> </xs:element> </xs:schema>
```

This XML Schema is almost self-explained. `<xs:element name="projectmanagement">` defines an `projectmanagement` as a `complexType`, as well as `project` is defined as `complexType` and means that max occurrences of project elements is unlimited.

XML Schema specification has three different parts:

1. XML Schema Part 0 Primer Second Edition – an easy readable overview for learning about concepts,
2. XML Schema Part 1: Structures – full specification of XML Schema language,
3. XML Schema Part 2: Datatypes - full specification of XML Schema language.

Full text of current XML Schema W3C Recommendations and examples from October 28, 2004 are available at W3) at Internet address http://www.w3.org/TR/xmlschema-0/.

New W3C Candidate Recommendations from July 21, 2011 are available at Internet address http://www.w3.org/XML/Schema.

For this paper is important to understand another very important XML Schema design approach. XML Schema is usually stored in multiply files. When new XML Schema is designed, new types definitions are based on existing data types. Including an existing XML Schema is accomplished by `import` or `include` elements:

```
<xs:import namespace="http://www.bulajic.net/CS610" />

<xs:include schemaLocation="XMLPMv1.xsd" />
```

Import element allow including XML Schema from different target namespace, but include element requires that included XML Schema has the same target namespace. Target namespace qualified element name by using target namespace prefix defined in the XML Schema header and enables distinguishing between elements that have the same name but are defined in different target namespaces. In above XML Schema example, following line

```
xmlns:xs=http://www.w3.org/2001/XMLSchema
```

defines target namespace prefix `xs:` that is used to qualify all XML Schema elements. For example `projectmanagement` element definition:

```
<xs:element name="projectmanagement">
```

Storing XML Schema in multiply files encourages reusability. Other reasons are easy maintenance and access control. Instead to define the same elements again and again in each new XML Schema, element can be defined in a separate file and included or imported when necessary. If such element is changed, changes are done in one place and would affect all places where that element is used and all data types derived from modified element.

While this approach is working well and is supported very well in case when an XML document and XML Schema are stored in flat files, in case when XML document and XML Schema are stored in database this approach increase complexity and create a number of issues. These issues and challenges are discussed in section “XML AND DATABASES”.

XSLT Working Group

XSLT Working Group is responsible for:

1. XSL Transformations (XSLT) “*a language for transforming XML*” (W3C XSLT 2011)
2. XML Path (XPath) “*an expression language used by XSLT (and many other languages) to access or refer to parts of an XML document*” (W3C XSLT 2011),
3. XSL Formating Objects (XSL-FO) “*an XML vocabulary for specifying formatting semantics*.” (W3C XSLT 2011).

Following is an example of XSLT language stylesheet for transforming XML document from Figure 3 to HTML:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html> <head> <title>Projects and teams</title> </head>

  <body> <table border="1">

    <tr> <th>Project</th>

      <th>Team</th>

      <th>Team</th> </tr>

    <xsl:for-each select="projectmanagement/project">

        <xsl:sort select="id" data-type="text"                  or-
der="ascending"/>

     <tr> <td><i><xsl:value-of select="@id"/></i>      </td>

      <xsl:for-each select="team">

        <td> <xsl:apply-templates/> </td>
        </xsl:for-each> </tr>
      </xsl:for-each> </table> </body> </html>
  </xsl:template> </xsl:stylesheet>
```

This XSL stylesheet applied to the XML file described in Figure 3, will generate following HTML file:

```
  <html>
  <head> <title>Projects and teams</title>  </head>
    <body>   <table border="1">
       <tr> <th>Project</th>
         <th>Team</th>
         <th>Team</th></tr>
       <tr> <td><i>Project_1</i></td>
          <td>Team One</td>
          <td>Team Two</td></tr>
       <tr> <td><i>Project_2</i></td>
          <td>Team One</td>></tr>
       <tr> <td><i>Project_3</i></td>
          <td>Team One</td>
          <td>Team Two</td></tr>
     </table> </body> </html>
```

When this file is opened inside of a Web Browser it will create following output:

Project	Team	Team
Project_1	Team One	Team Two
Project_2	Team One	
Project_3	Team One	Team Two

The XSLT 2.0 is able to work with multiply input files in different formats such as relational database tables, file systems or geographical information systems (W3C Transformation 2011), and generate XML files, text files or HTML files.

The XSL stylesheet used for transformation in this example is also a Tree structure. All XSLT elements are properly nested and might not corrupt well-formed document structure. The namespace declaration at the top of the document refers to the XSLT elements namespace at URI *http://www.w3.org/1999/XSL/Transform.* This namespace inside of the XSL document recognize XSL elements and attributes.

The XML source and XSL stylesheet are two separate documents, stored in separate files. Differences between XML and XSL document are obvious and even XML document can contain processing instructions and DTD, XSL document can be very complex and contains all elements and attributes described in the appendix "B Element Syntax Summary" (W3C XSLT 2011), as well as XPath expressions and XSL-FO formatting elements.

The HTML document that is generated by XSLT is also a Tree structure and can be represented graphically as DOM. This is illustrated in the Figure 4:

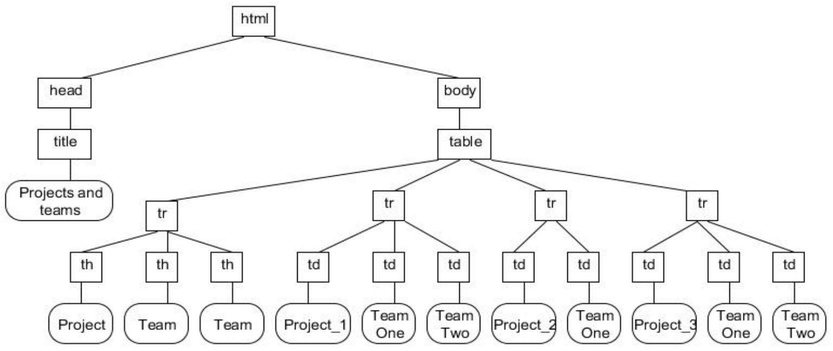


Figure 4. HTML DOM tree structure

Differences between source XML document and generated HTML document are easy to spot if Figure 4 is compared to Figure 3. The HTML can be also very complex and besides HTML elements and attributes can contains Cascading Style Sheets (CSS), script languages code, such as Java Script, Visual Basic script or even Java code embedded inside of the Java Server Pages (JSP). All these depend of the HTML elements that are specified in the HTML 4.01 Specification by W3C. The newest Extended Hyper Text Markup Language (XHTML) specification is based on the HTML 4.01 specification but is more strict regarding well-formedness and case sensitivity requirements, and require that each tag must end with end-tag and HTML elements must use lower cases for elements and attributes, as well as attributes shall be always quoted. The full specification is available at W3C Web page at URL http://www.w3.org/TR/xhtml1/#diffs.

Comparing XSL and HTML tree structures to binary search tree described in Figure 1 can lead to quick and wrong conclusion that XSL tree and HTML trees are more complex. Each tree structure has purpose and is optimized to certain kind of operations. A binary search tree is optimized for implementation of quick search and sort algorithms, while XML tree is optimized for hierarchical data presentations and storing of hierarchical data and HTML tree is optimized for data presentations and formatting.

Besides presenting different tree structures and of course XML tree structures and their differences, purpose of this presentation is also to show that tree structures are everywhere, in Word processors, and in flat files, in society and nature, in science and industry. There is almost always possible dissemble and create a hierarchical structure that has a root and parent and child nodes, never mind how you would call them, documents, pages, headers and footers, titles and table of contents and paragraphs and formatting instructions.

Famous scientist Donald E. Knuth in his lecture presented a different kind of trees created from a river base in State Georgia, a computer graphic created by different kind of binary search trees or a tree created by molecules. (Knuth 2006)

XQuery Working Group

Even the XML XPath is under XSLT Working Group responsibility, XML XPath is also developed and shared with XQuery Working Group. These two W3C Working Groups have made specification of XPath 1.0 and XQuery 2.0 Data Model (XDM). XDM is different data model than DOM, and besides different node types defines different data types.

For example while DOM model defines twelve different node types, XDM defines seven node types. XDM introduced data types such as xs:untyped, xs:anyAtomicType, xs:untypedAtomic that are not existing in DOM model. There are of course other differences as well as similarities. Both models are representing hierarchical tree structure and well-formed XML document. More details and full specifications are available at W3C Standards Web page at http://www.w3.org/standards/.

These two models are not only available XML models and there are also JDOM, DOM4J, XOM, Apache Axiom (Axiom) XML models, and all of these claims that are easy to learn and use, and provide better performances or optimized memory use and compatibility with DOM APIs.

XQuery is using XDM model to query and execute operations on XML data. XQuery is a language for querying XML data as well as it is SQL a language for querying data stored in a relational database.

XQuery history starts in 1998, when W3C show interest for query languages XQL and XML-QL. Jonathan Robie,, former XML Research Specialist at Software AG, currently the XML Program Manager at data connectivity software provider DataDirect Technologies Inc., together with Joe Lapp, the Principal Architect of webMethods Inc, developed the most of XQL query. They become members of W3C Working Group for XML-based quering languages and this working group created 90 use cases and tried this with seven query languages. Not any language was perfect and they take the best part of each and created the XQuery language. (http://gcn.com/Articles/2005/03/16/SQL-vs-XML-in-a-database-world.aspx)

"*XQuery is derived from an XML query language called Quilt [Quilt], which in turn borrowed features from several other languages, including XPath 1.0 [XPath 1.0], XQL [XQL], XML-QL [XML-QL], SQL [SQL], and OQL [ODMG].*" (W3C XQuery 2010)

First of all, XQuery is interesting because World Wide Web Consortium accepted and published XQuery 1.0 second edition recommendations (W3C XQuery 2010), and major database vendors, that already added XML support in their databases, are adding XQuery support too. The section "XML AND DATABASE" contains more details about this subject.

Even XSLT and XQuery purpose looks very similar and both operate on XML data, XQuery page at W3C at http://www.w3.org/XML/Query/ claims that XQuery is a language "*for combining documents, databases, Web pages and almost anything else*" and is "*replacing middleware languages and Web Applications languages*" as well as "*replacing complex Java or C++ programs with a few lines of code*".

Combining unstructured, semi-structured and structured data is very important for Online Analytical Processing (OLAP), Data warehousing and generally speaking Business Intelligence (BI) that analyses past, current and future of business operations and business performances. Data analysis, decision support systems and forecasting discovered very early that unstructured data such as text files, documents created by word processors,, e-mail, Content Management, Web pages, multimedia files, WIKI pages etc, are sources of valuable information. While processing of semi-structured data from XML documents and structured data stored inside of relational databases were very well supported, when are analyzed separately, combining these kinds of data and including unstructured data was a challenge.

IBM offered solution to this problem by introducing Federated Database Technology that were able to merge data from different data stores that can be a databases from different database vendors and enable analysis or processing of merged data by using existing applications and a common interface.

"*Today, federation capabilities enable unified access to any digital information, in any format -- structured and unstructured, in any information store.*" (http://www.ibm.com/developerworks/data/library/techarticle/0203haas/0203haas.html)

Other database vendors implemented Data Federation in their products and Oracle, for example, offers Data Federation capabilities through Oracle Data Service Integrator product, formerly known as AquaLogic Data Service Platform from BEA.

Importance of data combining from structured, semi-structured and unstructured data sources can be illustrated by information from The Data Warehousing Institute (TDWI) published in the "TDWI Best Practice Reports" under title "BI Search and Text Analytics: New Additions to the BI Technology Stack" written by Philip Russom, senior manager of research and services, where research study predicts that unstructured data and natural language information in text, such as voice recognition, WIKIs, RSS feeds, instant messaging and document management system will experience greatest increase, from approximately 60 to 80 percentage, while semi structured and structured data experience moderate increase. (http://download.101com.com/pub/tdwi/Files/TDWI_RRQ207_lo.pdf)

"IBM DB2 with pureXML" video presentation called "Seamless Integration of XML and relational data" claims that pureXML speeds development process by reducing project set-up time from 40 hours to 4 hours, by reducing development from 8 hours to 30 minutes, and reducing database changes from 40 hours to 5 minutes. It also significantly improves application performances for example in financial application by processing 5 times more customers' orders. (ftp://public.dhe.ibm.com/software/data/sw-library/db2/demos/purexml/index.html)

All this improvements are achieved by changing database architecture, but XQuery is also an important part of the pureXML implementation.

However, these solutions are not standardized and are vendor specific and limits vendor choice flexibility as well as portability under another software infrastructure. XQuery, as I can see it, is until now a successful attempt to make standardized solution.

Following are few examples of XQuery that queries XML document described in Figure 3 and results of execution:

Example 1:

```
(: XQuery select all teams :)
<project>{
  for $x in doc("data/XMLPMv1.xml")/projectmanagement/project
  let $pname := $x/team
  return {$pname}
}</project>
```

Result of execution:

```
<?xml version="1.0" encoding="UTF-8"?>
<project><team>Team One</team>
<team>Team Two</team>
<team>Team One</team>
<team>Team One</team>
<team>Team Two</team></project>
```

Example 2:

```
(: XQuery select projects different than "Project_1" :)
  for $x in doc("data/XMLPMv1.xml")/projectmanagement/project
  let $pname := $x/project
  where $x/@id!="Project_1"
    return {$x}
```

Result of execution:

```
<?xml version="1.0" encoding="UTF-8"?>
<project id="Project_2">
        <id>2</id>
  <team>Team One</team>
 </project><project id="Project_3">
        <id>3</id>
  <team>Team One</team>
  <team>Team Two</team>
 </project>
```

Example 3:

```
(: XQuery Ex3.: replace tag and attribute names :)
<allteams> {
  for $x in doc("data/XMLPMv1.xml")/projectmanagement/project
    let $pname := $x
    let $tname := $x/team
  return {  <teamname name="{data($pname/@id)}">
    {$tname} </teamname>}
} </allteams>
```

Result of execution:

```
<?xml version="1.0" encoding="UTF-8"?>
<allteams><teamname name="Project_1"><team>Team One</team><team>Team
Two</team></teamname><teamname name="Project_2"><team>Team
One</team></teamname><teamname name="Project_3"><team>Team
One</team><team>Team Two</team></teamname></allteams>
```

All these examples are developed by using of Eclipse Indigo, open source Java IDE (http://www.eclipse.org/), and QXDT, open source XQuery IDE for Eclipse (http://www.xqdt.org/) and Zorba the XQuery Processor (http://www.zorba-xquery.com). The QXDT provides an interactive environment for developing and testing of XQueries and support other XQuery processors, such as MarkLogic, Saxon or MXQuery.

The XQuery language reference and last XQuery 3.0 W3C Working Draft from 14 June 2011 are available at W3C http://www.w3.org/TR/xquery-30/#id-try-catch.

While XML document is optimized for storing and exchanging of data, relational database is optimized for storing and querying data. XQuery tries to reconcile differences between these two approaches and make a language that is able to compete to SQL by performances and flexibility.

”*Use XQuery to take data from multiple databases, from XML files, from remote Web documents, even from CGI scripts, and to produce XML results that you can process with XSLT. Use XQuery on the back-end of a Web server, or to generate Enterprise-wide executive reports.*” (http://www.w3.org/XML/Query/s)

XML and database

All major relational database vendors, besides XML support, added also XQuery support in their database. For example, Oracle, IBM and Microsoft added XQuery support in the:

1. Oracle XML DB,
2. Microsoft SQL Server 2005 and 2008,
3. IBM DB2 with pureXML.

There are four different approaches to storing an XML document in the relational database:

1. Storing XML in a large objects such as TEXT or Large Objects (LOBs) columns, known also as XML stuffing,
2. Decomposition of an XML document and mapping to relational tables and columns, known also as shredding,
3. Specialized Data Base Management Systems (DBMS) for storing and managing XML documents,
4. Hybrid solution, where relational database is extended by native XML data type for storing and manipulating of XML documents.

The first solution, stuffing an XML document, looks very much as storing an XML document in file system and flat files. Advantage of storing an XML document in the database LOBs is mostly related to the availability of database data management system, such as data security, access control, backup, recovery, indexing, etc. While this approach is working well in case when entire document is required for processing, access to the particular nodes or portion of an XML document would introduce performance overhead, because an XML document needs to be read and parsed again and again for each new request to access particular document node.

The second solution, shredding, requires that an XML document is parsed and each part is mapped to one or more database tables and columns. In this case processing overhead is moved to a storing an XML document, but searching and retrieving particular nodes and portions of an XML document can be very fast. Another issue in this case is increased complexity of SQL statements because of huge number of tables that can be involved in the SQL transaction. For example, storing an Financial products Markup Language (FpML), the industry standard for complex financial products, in relation database could require 485 tables. Storing of 1500 electronic form in relational database tables could require more than 30.000 tables. (IBM Redbooks 2006)

Working with such huge number of tables, as well as management and maintenance can introduce extreme high complexity.

The third solution, specialized XML DBMS, offers specialized DBMS for XML documents. One example is Oracle XML DB. "*Oracle XML DB is a feature of the Oracle Database. It provides a high-performance, native XML storage and retrieval technology. It fully absorbs the W3C XML data model into the Oracle Database, and provides new standard access methods for navigating and querying XML. With Oracle XML DB, you get all the advantages of relational database technology plus the advantages of XML. In Oracle Database 11g Release 2, new capabilities, such as XMLIndex structured component and partitioning, as well as numerous enhancements of Oracle XML DB are introduced to dramatically improve its performance, scalability, XML schema management, and XML information lifecycle management.*"(http://www.oracle.com/technetwork/database/features/xmldb/index.html)

This approach until now did not attract significant customer interests and introduce a new unproven DBMS, where scalability, reliability and availability, as well as availability of competent and skilled developers can be an issue. (IBM Redbooks 2006)

The fourth solution, Hybrid solution, offers combination of relational database and XML native data type that is used for storing XML document. "*The objective of such systems is to preserve the benefits associated with commercial relational DBMS offerings – including high levels of scalability, reliability, availability, concurrency, and customer support – while making it easy to manage and integrate existing corporate data with data modeled in hierarchical XML structures.*" (IBM Redbooks 2006)

What are challenges in case of XML stored in the database? The first one is performance consideration. If XML is stored as character data, then each access to any XML data requires first parsing of XML document and then searching or querying. This should be repeated for each new query.

To speed up this process XML document can be divided, and portions of a document stored in a separate tables. This would speed up parsing and querying, but move complexity to queries that need to search a whole document, as well as will create document well-formedness and validation issues.

Another important issue is XML document validation. If XML document shall be validated each time when it is stored or changed in database that would require that DTD or XML Schema is stored in database too. Storing XML Schema can be a challenge, because XML Schema can be very complex and include or depend of another XML Schema or XML Schemas. For storing XML Schema complex types can be necessary to define a table with more than 1000 column. Oracle, for example limits number of columns per table to 1000 column, and in this case it would require more tables or implement partitioning to store more than 1000 columns.

If XML Schema is partitioned and stored in more tables, then validation can be an issue, because only a part of XML document shall be validated and only a corresponding part of XML Schema shall be used. Such requirement can create unforeseen complexity and require additional time to design a proper solution. This can also affect performances and prevent optimization of both, XML document and XML Schema.

If an XML document is frequently changed, changes shall replace corresponding part of XML document. If an XML document is stored as whole, then whole document shall be retrieved, parsed, modified and stored. If only a portion of XML document is stored then it can speed up XML document processing.

If XML document is stored inside of relational database tables, then it can require hundreds of tables and relations, indexes and all these is not for free. It affects storage resources and performances because indexes need to be maintained and in case of frequent XML document changes and huge number of indexes can affect performance. Indexing each tag and path in a XML document can be time consuming process, but it improves query performances.

Each of database vendors offers own solution to these common issues. For example Oracle XML DB version 11g (Oracle Database 11g 2010) offers XMLType, an abstract data type and three different XML document storage and index options (Oracle Database 11g 2010):

1. Structured storage – XMLType is stored relationally and optimized for query performances and in place updates,
2. Binary XML storage – XMLType is stored in a post-parsed binary format and optimized for flexibility,
3. Unstructured storage – XMLType is stored in Character Large Object (CLOB) instance and shall be used very seldom in case when full XML document is frequently retrieved.

These storage models can be mixed and one part of an XML document can use one model and another part can use other model. Oracle also offers different indexes on XMLType (Oracle Database 11g 2010):

1. B-Tree index on Structured storage,
2. XMLIndex with structured componenton all storage options,
3. XMLIndex with unstructured component on Binary XML and Unstructured storage,
4. Secondary B-Tree index on the secondary tables for structured and unstructured component.

Oracle defines a set of Use Cases and recommendations which storage and indexing model to use to optimize performances or flexibility as well as to satisfy processing requirements.

"Each of these storage models and index combination has its own advantages and disadvantages in different dimensions, such as performance and flexibility. No single storage or index is right for every use case." (Oracle Database 11g 2010)

Microsoft SQL Server offers XML data type where an XML document or fragments of an XML document can be stored. (Microsoft SQL Server 2008 R2 2011) Optionally, an XML Schema collection can be associated with XML data column. Microsoft also describes limitations of XML data type such as (Microsoft SQL Server 2008 R2 2011):

1. a size of XML data type cannot exceed 2GB,
2. it cannot be compared and sorted,
3. it cannot be used as a subtype of sql_variant instance,
4. does not support casting and converting to text either ntext,
5. it cannot be used as a parameter to any scalar, built-in functions other then ISNULL, COALESCE and DATALENGTH,
6. it cannot be used as a key column in an index.

XML data type and five built-in XML data type methods provide mechanism for handling and processing of XML data by using Transact-SQL Select statements, as well as querying by using XQuery standard language.

Microsoft offers a XML Data Modification Language (XML DML) as an extension to XQuery standard that is missing data manipulation language. This extension "*provides a fully functional query and data-modification language that you can use against the xml data type*" (Microsoft SQL Server 2008 R2 2011) and added to XQuery keywords (Microsoft SQL Server 2008 R2 2011):

1. *insert* – inserts one or more nodes,
2. *delete* – deletes one or more nodes,
3. *replace* value of- updates value of node.

The XML data type can be handled by using ActiveX Data Objects (ADO) for .Net (ADO.NET) and Microsoft .NET Framework. If it is necessary to work with XML data type contents, then contents shall be first assigned to a XMLReader type. (Microsoft SQL Server 2008 R2 2011)

IBM pureXML solution implements a hybrid solution and combines XML database with a relational database. According to IBM papers, this solution "*features hybrid data management technology that incorporates proven relational capabilities with first-class support for storing, searching, sharing, validating, and managing XML data. The result is a reliable, scalable platform that provides high performance for accessing and integrating "traditional" corporate data as well as XML data.*" (IBM Redbooks 2006)

Figure 5, inspired by IBM Figure 1-2 (IBM Redbooks 2006)¨illustrates IBM pureXML solution:

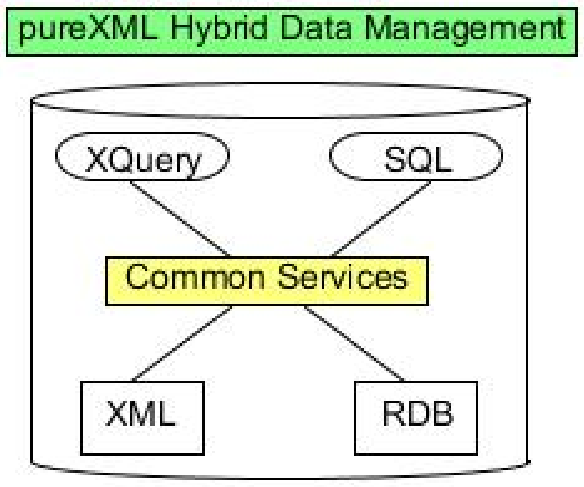


Figure 5. IBM pureXML Solution Overview

IBM has implemented new storage for efficient management of hierarchical structures, as well as SQL/XML enhancements and query optimizations technique, and enables using of XQuery or SQL or combination of XQuery and SQL for XML and SQL data handling and processing. Integrated data management provides transactions and security support, as well as high level of scalability and reliability.

IBM claims that pureXML:

1. Improve productivity and lower development costs,
2. Increase application performances,
3. Simplify operating environment.

There is no doubt that IBM is strong and innovative partner and each IBM solution keep track in software as well as in hardware industry.

However, making decision which database use for future project that will almost for sure use XML data is not an easy decision and depends of a number of different factors as well as of previous experience and environment in which are running existing applications.

Before making any decision I would, if it is possible, recommend serious research and testing of each solution. Solution that are built-in inside of PL/SQL and Transact-SQL or are using any kind of extension that are not widely accepted standard can significantly limit or even prevent application portability. As it is today, solution based on XQuery 3.0, looks as an emerging technology and a standard solution for next decade and is already implemented in all databases of major relational database vendors.

Conclusion

Many are using XML document and tools for parsing, browsing, as well as traversing, deleting, inserting and modifying a XML structure without even thinking what kind of structure XML document really is.

Differences between an XML tree structure and binary tree structure are most visible in the tree structures. While binary search tree is obsessed by sorting and tree node balancing, an XML document is obsessed by well-formedness and validation.

Similarities are most obvious in case of tree structure searching and querying operations. Both are looking for the fastest way to return data based on the search or query requirements and criteria.

A Binary search tree and an XML tree are not mutually exclusive and often are combined to achieve common goals, such as optimizing tree structure for searching or storing a data, or for both, searching and storing. An example is implementation of database indexes, where B-tree is combined by XML, and keeps indexes to XML.

XML has been primary designed for interoperability, as a platform independent subset of SGML. Changes in the architecture of software applications, and Service Oriented Architecture (SOA) and Web Services, heavily increased use of XML documents as a primary data model for exchanging data between different applications across platform and network boundaries.

Globalization and outsourcing introduced new set of requirements for global communications across locations, networks and country borders. Internet infrastructure provides a global network. Using Internet and a widely accepted and adapted standards, such as XML, for interoperability and data exchange, was quite natural choice, because creating and widely accepting a standard can be time consuming and daunting task in the world where each company compete for advantages and market shares

Another data model for storing and retrieving and querying data, relational data model, has been already widely accepted and implemented, but hierarchical XML data model does not fits naturally to a relational model. Requirement for information exchange growth rapidly in last decade, and as well amount of exchanged data growths rapidly too. Suddenly it was not enough just exchange data, but also necessary to search and query and compare data that were stored in an XML document. Sequential processing of XML document stored in the file system was not able to satisfy those requirements and it was quite naturally to experiment with storing XML document in a database, and of course in a relational database, that is still dominant and proven, and most used database model.

However, fundamental differences in the design of these two models created issues that are described in the section "XML AND DATABASE", and can be summarized as storing and querying challenges, as well as validation of a stored and modified XML document.

While validity of data stored in the relational database is verified by table design and designed indexes and keys, and data types and data length, validity of an XML document verified by XML Schema can be quite complex and daunting task. XML Schema is an external document that is itself an XML document and storing and retrieving XML Schema assumes all challenges that are associated by storing and retrieving XML document. Maintenance of links between XML Schema and corresponding XML document or portion of corresponding XML document can be a challenge.

Maintenance of XML Schema document can be very complex task, and external dependencies that are part of XML Schema design, can introduce unforeseen issues and issues that can create a lot of troubles and consume a lot of processing time.

Some are considering the Internet infrastructure as a huge distributed XML database, and XQuery as a language for queering this huge dataset. This is also an important difference between Binary Search Trees and XML tree data. While Binary Search Trees searching and modifying is done by algorithms, XML tree data searching and modifying is accomplished by other tools, languages and software packages. The same is valid for these two tree structures transformations. A Binary Search Tree is also transformed by implementing of an algorithm, (for example balanced tree algorithms), while an XML transformations are accomplished by the XSLT or XQuery processors.

I had noticed, when I executed XQueries examples by Zorba XQuery Processor, that the same XQueries executed by MXQuery XQuery Processor, issues errors and exceptions. This would means that even XQuery is standardized, XQuery processor implementation can have some specific non-compatible solutions. This can be compared to the SQL and SQL implementation by different vendors. For example, Oracle delivers PL/SQL and Microsoft Transact-SQL, extensions to SQL. Using one of these tools is best way to get in a vendor lock situation. This means that using vendor specific extensions would limit portability of source code or make it very expensive, or even impossible.

By presenting different solutions and approaches to implementing and manipulating tree structures, this document describes that today is very important to think out of box and out of limitations that are existing in any kind of technology. In real life project, there is never only one solution proper for all kind of even very much related challenges. Especially today, when requirements for distributed applications, distributed processing and distributed data management are part of even most simple application, one technology or one kind of tool is never sufficient. Rather it is a combination of different tools and technologies that enables effective implementation.

References

Apache Software Foundation. (2011). *The Apache Xerces Project*. Retrieved from http://xerces.apache.org/index.html

Department of Genome Sciences and Department of Biology. (2011). *PHYLIP*. University of Washington. Retrieved from http://evolution.genetics.washington.edu/phylip.html

Galperin, I., & Rivest, R. L. (1988). *Scapegoat trees.* Cambridge, MA: Laboratory for Computer Science, Massachusetts Institute of Technology. Retrieved from http://www.akira.ruc.dk/~keld/teaching/algoritmedesign_f07/Artikler/03/Galperin93.pdf

GRAPHWIZ.ORG (2011). *Graph Visualization Software*. AT&T Labs Research. Retrieved from http://graphviz.org/

IBM Redbooks. (2006). *DB2 9: pureXML Overview and Fast Start.* Retrieved from http://www.redbooks.ibm.com/abstracts/sg247298.htm

Knuth, D. E. (2006). *Trees, rivers and RNA*. Stanford University. Retrieved from http://scpd.stanford.edu/knuth/index.jsp

Microsoft SQL Server 2008 R2. (2011). *Implementing XML in SQL Server*. Microsoft MSDN. Retrieved from http://msdn.microsoft.com/en-us/library/ms189887.aspx

Oracle Database 11g. (2010). *Oracle XML DB: Choosing the best XML type storage option for your use case*. Oracle Retrieved from http://www.oracle.com/technetwork/database/features/xmldb/index.html

Oxford Advanced Learner's Dictionary. (2011). *Hierarchy.* Retrieved from http://www.oxfordadvancedlearnersdictionary.com/dictionary/hierarchy

Pavlopoulus, G. A., Soldatos, T. G., Barbosa-Silva, A., & Schneider, R. (2010). A reference guide for tree analysis and visualization. *Bio Data Mining, 3*(1). Retrieved from http://www.biodatamining.org/content/pdf/1756-0381-3-1.pdf

SAXProject. (2011). *Genesis*. SAX Project. Retrieved from http://www.saxproject.org/sax1-history.html

Sedgewick, R. (2008). *Left-leaning red-black trees.* Princeton, NJ: Department of Computer Science, Princeton University. Retrieved from http://www.cs.princeton.edu/~rs/talks/LLRB/LLRB.pdf

Sleator, D. D., & Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the Association for Computing Machinery, 32*(3), 652-686. Retrieved from http://www.cs.princeton.edu/courses/archive/fall07/cos521/handouts/self-adjusting.pdf

W3C DOM (2005). *Document Object Model (DOM)*. W3C. Retrieved from http://www.w3.org/DOM/

W3C XML (2003). *Extensible Markup Language*. W3C. Retreived from http://www.w3.org/XML/

W3C XML 1.0 (2008). *Extensible Markup Language (XML) 1.0* (5th ed.). W3C. Retrieved from http://www.w3.org/TR/2008/REC-xml-20081126/#sec-starttags

W3C XML 1.1 (2006). *Extensible Markup Language (XML) 1.1* (2nd ed.). W3C. Retreived from http://www.w3.org/TR/2000/REC-xml-20001006

W3C XSLT (2011). *The Extensible Stylesheet Language Family (XSL)*. W3C. Retrieved from http://www.w3.org/Style/XSL/

W3C Transformation (2011). *Transformation*. W3C. Retrieved from http://www.w3.org/standards/xml/transformation

W3C XQuery (2010). *XQuery 1.0: An XML Query Language* (2nd ed.). W3C. Retrieved from http://www.w3.org/TR/xquery/

Biographies

Aleksandar Bulajic is working for IBM Denmark as a consultant and full time employee. He is working more than twenty five years as a computer expert and consultant for some of the top companies in the world.

Aleksandar Bulajic is PhD Candidate at Faculty of Information Technology, Metropolitan University.

He graduated from the University of Liverpool, with a Master's in Science degree (Cum Laude) in Information Technology, and from the Economic University with a Bachelor's (BA) degree.

His papers and articles were presented and published at several international IT conferences in USA, Australia, Canada and Serbia, and presented at Liverpool University and published in professional journals and on IBM Intranet. He wrote and published several novels and a stage play, and is working on screenplay manuscripts. Besides his professional work and writings, his current interests and writings are mostly related to film and theatre and interdisciplinary multimedia experiments. He is a member of Liverpool University Alumni Community and Informing Science Institute. E-mail: **LANB@45.dk** **aleksandar.bulajic.1145@fit.edu.rs**

Nenad Filipovic is Professor of Biomechanics and Informatics at the Mechanical Engineering Faculty of the University of Kragujevac, Serbia, Faculty of Information Technology, Metropolitan University, Belgrade, Serbia and Research Associated at Harvard School of Public Health at University of Harvard (USA). His research interests are in the area of software development for fluid mechanics, coupled problems; fluid-structure interaction, heat transfer; biofluid mechanics; biomechanics, multi-scale modeling, discrete modeling, molecular dynamics, computational chemistry and bioprocess modeling. He has published more than 80 papers in peer review journals and 5 international books in area of computer simulation for bioengineering. He is Director of Center for Bioengineering at University of Kragujevac and leads joint research projects with Harvard University and University of Texas in area of bio-nano-medicine computer simulation. He also leads a number of national projects in area of bioengineering. He is a Managing Editor for Journal of Serbian Society for Computational Mechanics and member of European Society of Biomechanics (ESB) and European Society for Artificial Organs (ESAO).