Master of Applied Computer Science
2015-2016

XML DATABASES

Advanced Databases

Student : Angeliki – Ioanna Katsiampouri
Professor : Esteban Zyimani

Contents

1.Introduction

1.1 Definition of XML

Extensible Markup Language (XML) [2] [3] is a meta-markup language that was developed to handle the shortcomings of HTML. It is both human and machine readable. It is a textual format data, supported by Unicode, so as can be widely used technology-independent. The most common use of XML is for documents, though is also widely used for data structure description, such as the web services.

Example:

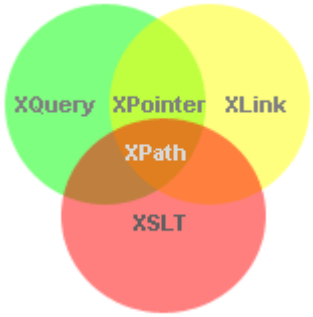
```
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
       <body>Don't forget me this weekend!</body>
</note>
```

1.2 Definition of XSLT

Extensible Stylesheet Language Transformation (XSLT) , is a stylesheet language applied to XML documents to transform them into html or another XML (or other types).

1.3 Definition of Xpath

Xpath is a syntax for defining parts of XML documents, and it uses path expressions to navigate within the document. It contains a library of standard functions that help on navigation and manipulation of the elements. It is a tool used a lot in XSLT to parse the XML document in order to transform it.



1.4 Definition of Xquery

Xquery is for XML what sql is for database tables. It is designed to query XML fragments out of XML documents. It is built on Xpath expressions, and is supported by all major databases.

```
eg. for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

1.5 Definition of XQL

XQL (XML Query Language) [16], is the language that allows users to intelligently query XML data sources. The basic construction of XQL correspond directly to the basic construction of XML. XQL is closely related to Xpath. A very simple example follows with the XML document and the XQL query:

```
<invoice>
  <customer>
        Wile E. Coyote, Death Valley, CA
  </customer>
</invoice>
```

Query: //customer

Result:

```
<xql:result>
  <customer> Wile E. Coyote, Death Valley, CA </customer>
</xql:result>
```

2.XML Database

2.1 Definition of XML Database

According to Wikipedia, XML Database is defined as a data persistence software system that allows data to be specified, and sometimes stored, in XML format. Those data can be manipulated in the same way as with relational database, which means they can be added, deleted, updated, queried, transformed, exported and returned to the calling system. They are document oriented, since they store whole XML documents.

2.2 Different Types

There are two different types of XML databases:

- XML enabled databases and
- XML native databases

2.2.1 XML Enabled Databases

XML Enabled Databases (which most commonly is the relational and the object-oriented databases), are those that are extended to hold XML data. They can transfer data between XML documents and their own data structures. The data must be modelled in both structures, XML based and relational based, and that is not always handy.

Some products that support XML are: Access 2007, Cache, DB2, extremeDB, FileMaker, FoxPro, MySql, Oracle, PostgreSQL,SQL server and many other databases.

In order to have a better understanding of the concept, we will go in more details to one of those technologies combined with XML. For our instance, we take MySQL,a relational database [5]. All code is written in Perl.

Let's assume that we have a table named animal with two columns, name and category.

```
+----------+--------------+
| name     | category     |
+--------+-----------+
| snake    | reptile      |
| frog     | amphibian    |
| tuna     | fish         |
| racoon   | mammal       |
+--------+-----------+
```

One way to generate XML documents out of relational data, is to print manually all the XML document, row by row, and have as values the retrieved values from the table.

```
my $sth = $dbh->prepare ("SELECT name, category FROM animal");

$sth->execute ();

print "<dataset>\n";

while (my ($name, $category) = $sth->fetchrow_array ()){

    print " <row>\n";
     print "  <name>$name</name>\n";
     print "  <category>$category</category>\n";
     print " </row>\n";
}
  $dbh->disconnect ();
  print "</dataset>\n";
```

This piece of code, generates the XML beneath:

```
<dataset>
 <row>
  <name>snake</name>
  <category>reptile</category>
 </row>
 <row>
  <name>frog</name>
  <category>amphibian</category>
 </row>
 <row>
  <name>tuna</name>
  <category>fish</category>
 </row>
 <row>
  <name>racoon</name>
  <category>mammal</category>
 </row>
</dataset>
```

This approach, however, could be used only for simple XML documents. To make it more dynamic, and scalable for complex and big XML documents, we can use utility classes.

Perl contains some utilities that support the flexible XML generation. For instance, *XML::Generator::DBI* helps to convert the contents of the animal table to XML. The SAX handler is obtained from the *XML::Hander::YAWriter* (yet another writer) module.

```
use strict;use DBI;use XML::Generator::DBI;use XML::Handler::YAWriter;

 my $dbh = DBI->connect ("DBI:mysql:test",
                  "testuser", "testpass",
                  { RaiseError => 1, PrintError => 0});
 my $out = XML::Handler::YAWriter->new (AsFile => "-");
 my $gen = XML::Generator::DBI->new (
                       Handler => $out,
                       dbh => $dbh
                     );
 $gen->execute ("SELECT name, category FROM animal");
 $dbh->disconnect ();
```

The output of the code is more or less the same with the XML from the former approach.

So far , we have seen how the relation table data can be transformed to XML data. For the opposite, there are also utility classes. Perl provides a parser class that parses the XML, retrieves the values of specific tags and stores it on database.

Assuming that we have as input the XML document generated above, we can have the code beneath to retrieve data from the document.

```
use strict;  use DBI;use XML::Parser;

  my %row = ("name" => undef, "category" => undef);

  my $dbh = DBI->connect ("DBI:mysql:test",
                  "testuser", "testpass",
                  { RaiseError => 1, PrintError => 0});
  my $parser = new XML::Parser (
                  Handlers => {
                     Start => \&handle_start,
                     End   => \&handle_end,
                     Char  => \&handle_text
                  }
               );
  $parser->parsefile ("animal.xml");
  $dbh->disconnect ();
```

Moreover, Xpath can be used to parse XML documents and get the values we want to store on the database.

An XPath object contains an in-memory representation of the XML file, then gets a pointer to a list of the elements we need within the document. For each of these, it extracts the text and inserts them into the table.

The code below shows the retrieval of values in the XML code we provided before, using Xpath.

```
use strict;use DBI;use XML::XPath;use XML::XPath::XMLParser;

my $dbh = DBI->connect ("DBI:mysql:test",
                        "testuser", "testpass",
                        { RaiseError => 1, PrintError => 0});
my $xp = XML::XPath->new (filename => "animal.xml");
my $nodelist = $xp->find ("//row");
foreach my $row ($nodelist->get_nodelist ())
{
    $dbh->do (
        "INSERT INTO animal (name, category) VALUES (?,?)",
            undef,
            $row->find ("name")->string_value (),
            $row->find ("category")->string_value ()
        );
}
$dbh->disconnect ();
```

Most of database technologies that support XML, have these kinds of approaches for reading and generating XML documents.

2.2.1.1 XML Enabled Databases Architecture

Let's take for example a relational database as underlying storage, to see the architecture on high-level[17]. The Xml document schema must be translated to relational schemas, before it accesses the corresponding files. XQL must be also translated to SQL before the relational tables are accessed. XQL is implemented as add-on function to the RDBMS. The relational engine parses then the XQL and translates it to SQL. Then SQL executes the query in RDBMS.

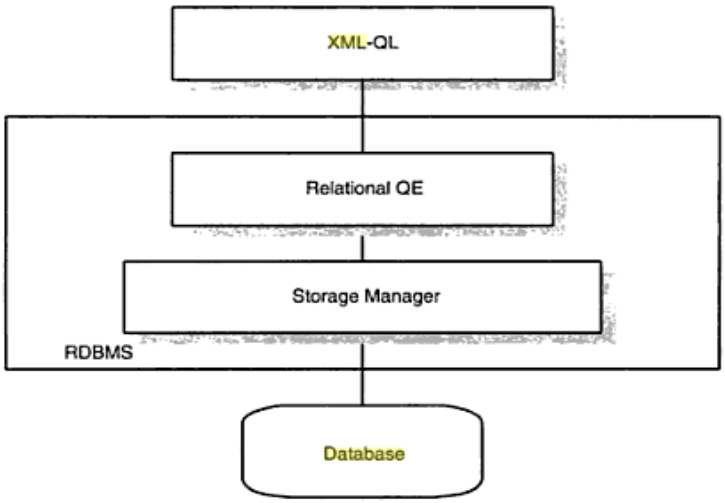


Figure 1 : XML Enabled Database Architecture

2.2.2 XML Native Databases

2.2.2.1 Definition of XML Native Databases

XML:DB initiative [6] offers its own definition for XML native databases.

They define a logical model for the XML documents, on which they are based to store and retrieve the documents. This model, must contain at least elements, attributes, document order and PCDATA. Xpath and XML Infoset are examples of this type of models. Furthermore, they have an XML document unit as the (logical) storage point, exactly like the relational databases have a row in a table as the logical unit to store the data. Those databases require no specific underlying physical storage model, but they can be built on top of other databases(eg relational).

2.2.2.2 Features of XML Native Databases

Since the XNB are not really independent models, they cannot replace the existing databases, but they can offer some different and interesting features that can help developers on their projects, according to their requirements. The following list provides the general features of XNB, but that does not mean that all the databases have the same or not more features [6] [7].

XML storage - The data are stored as a unit on the XML document. The levels of complexity can be arbitrary. With the appropriate mapping, the data are stored on the underlying storage model as well. That is to ensure that the data will be maintained. To retrieve the data from the underlying model, should be done also with NXB tools, such as XSLT or DOM. Otherwise, the data that will be retrieved , will not be very representative of an XML document.

Collections - Documents can be stored as a set on the database. That is similar to the concept of the table on the relational databases. A collection does not need to associated with a schema, so the documents can be stored regardless the schema. The databases that support this functionality , are called *schema-independent*.

Queries - A good tool that is expanded so as to allow queries on documents or collections, is the Xpath. Though, it was not built as query language, and so it still misses some basic functionalities, such as grouping, sorting, or join of two documents. XSLT can be used to fill those gaps on functionalities, but yet the complexity remains. Because of those lacks, another tool is now under development, in order to support more functionalities for NXB and make it more competitive to other types of databases. This tool is Xquery, and several prototypes have been already released to work on databases. NXDs support also indexing, in order to improve the performance of quering.

Updates and Deletes - NXBs can update or delete the whole documents, mainly through DOM tree to languages that specify how to handle pieces of a document. XML:DB initiative has developed the Xupdate language , that is used from the most technologies on the NXB domain.

Transaction, Locking and Concurrency - All NXB support transaction and rollback. Locking , however remains mostly on the level of a whole document and not on parts of it, so multi-users concurrency is lowered down. Node level is also possible, but the implementation is not easy, as it would require validating and enforced schemas. So , the bigger the XML document is, the less multi-user capacity we have.

Those are some general features of the Native XML Databases. Different technologies might use all or some of those features, but they might also provide some more. To make the concept of NXB more comprehensible, there will be an extended information / explanation-illustrated with examples on a technology of this domain. This example, will be the technology named *eXist,* that is open source and can be used by everyone.

3. eXist

3.1 Overview

eXist is an open source database management system, built on XML technology. It stores the XML data according to the logical model and it has an index-based Xquery processing. eXist is mainly written in Java. It supports many web technologies, and that is why it is a very good tool to work with when developing web applications.:

- Xpath, Xquery, XSLT web tools
- SOAP, REST, ATOM protocol, WebDav, XMLRPC interfaces
- XMLDB, Xupdate, Xquery update extensions for database specific

3.2 Features

Some of the provided features by eXist are already mentioned in the general features of the NXB. Exist offers more :

- *Authorization mechanism* : Unix-like access permissions for users/groups at collection- and document-level.
- *Security* : it uses the Extensible Access Control Markup Language for Xquery access control.
- Deployment : eXist can be deployed as a standalone database server, as an embedded Java library or as part of a web application.
- *Backup /restore* : this functionality is provided either through Ant scripts or through Java admin client. Backup contains resources in XML form to be readable. It allows a full restore of database, including the user permissions.

3.3 Architecture

Let's take a closer look at the architecture of eXist, to have a better understanding of how the XML documents are processed and stored [10].

The high-level architecture of eXist technology is shown at the image that follows:

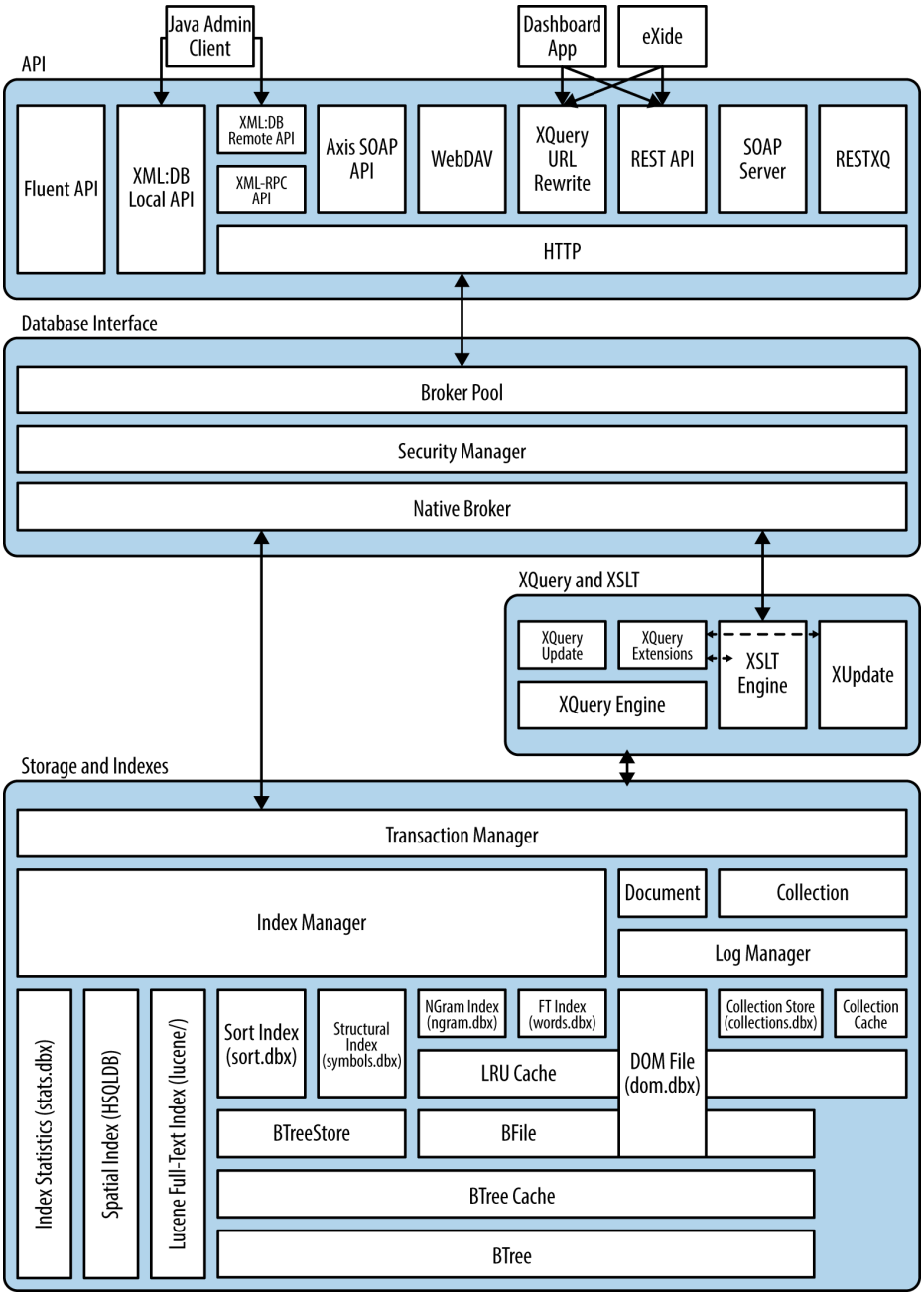


Figure 2 : eXist Architecture

Each API is connected with eXist through a broker pool, that by default contains 20 brokers, but this is configurable by the user. Each broker represents a request from API to database, so there can be maximum requests the amount of brokers that are configured on database. If there are more requests, they have to go to a queue and wait until the thread(broker) can execute them. A request might be an update operation (CRUD), or a query operation (Xpath, Xquery, XSLT). In the second case, there must be

Exist provides several APIs for different situations. In case that eXist DBMS is embedded to the application, then two APIs are offered for communication : the XML:DB Local API and the Fluent API . When there is a server-client architecture, then APIs like WebDAV API , XML:DB Remote API or XML-RPC are used. The first one allows users to manipulate documents from database, like they were stored in their local machine.

The second one, allows users to connect to DBMS through Java admin client, but then first the third API needs to have been enabled.

Independently the type of architecture that exists between the application and eXist, the DBMS internally has some specific functionality.

When the XML document reaches eXist, then it is parsed and all information is extracted and stored by indexing its features. Every part that is extracted, is then stored on disk as binary file. Indexing helps on efficient and quick response when the XML document will be asked by the user. Collections of documents are handled in eXist like folders in filesystems. Each collection is identified by a URI. It stores all the metadata of itself and of all the documents that are contained in the collection.

Exist manipulates XML files with dynamic labeling numbering (DLN). DLN helps eXist to identify a node and its connection with other nodes, by giving a unique label to each node. This is done in hierarchy, starting from the root node to the children. For each node, DLN includes the ID of all the previous elements. That makes it easy to connect the nodes hierarchically, when the XML document needs to be reconstructed and retrieved.

Paging and caching is one more feature of eXist technology. Size of page in eXist is configurable. DOM and Collection files are splitted into two parts, the data and the index section. Data section contains the node,document or collection metadata, while the index section ensures the quick lookup of them. Cache is also used for keeping the most used documents on memory, so they are not needed to be requested from disk.eXist uses LRU) Least Recently Used) caches, so pages are retrieved from caches, based on the time they were last accessed.

3.4 eXist Demo

Official site of eXist provides some demos of small applications built with eXist-db [11]. There are several examples, based on different concepts, such as unit testing, Xforms and content extraction.

In the following session, we are going to analyse one simple application example that can be found on internet , together with its source code. The example that will be used is part of the *Web examples* section, and is named *Shakespear*. The concept of this small application is to query as user anything that could be connected to famous writer Shaksepear. Then , this query is passed to the database that contains XML files with a lot of Shakespear's creations and their contents. Every match that is found is returned to the screen.

The image beneath, is the html representation of the application. In the field *Query* we can type anything, and we can filter it through the *Mode* field, which have three filters: *near, anything,* and *phrase*.

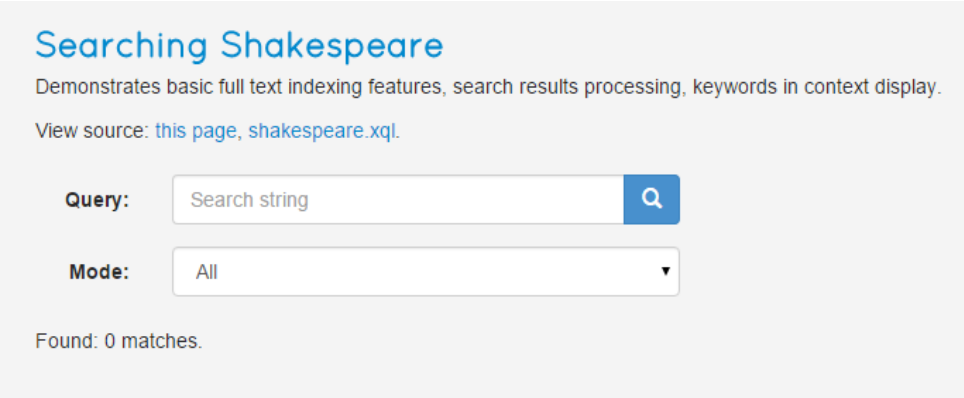


Figure 3 : Shakespear html output

To view the html code, we can click on the source code link, and we will be redirected to the exIDE[12]. In the IDE, we can view the source code of all the example applications, as well as the XML documents that are stored for each of those applications. The html, ajax and xql page that are used for querying, as well as the XML documents stored on the db for this application are stored in those paths:

Figure 4 : Paths to database files

The xql takes as input the parameters passed from html page and sends the request to db that parses all XML documents attached to this application , and returns the result.

```
<div id="results" class="shakespeare:query">
  <p>Found: <span id="hit-count" class="shakespeare:hit-count"/> matches.</p>
  <div class="shakespeare:show-hits?howmany=20"/>
</div>
```

This piece of html code calls functions of the xql page, and it displayes then the output of these functions.

The main function, takes input parameters from the query , executes the query by searching on the db, and then it returns a map. The templating module recognizes this map and will merge it into the current model, then continue processing any children of $node.

```
declare
  %templates:wrap
function shakes:query($node as node()*, $model as map(*), $query as xs:string?, $mode as
xs:string?) {
  session:create(),
  let $hits := shakes:do-query($query, $mode)
  let $store := session:set-attribute($shakes:SESSION, $hits)
  return
    map:entry("hits", $hits)
};
```

The do-query function does the call the db, and eXist searches on the collection of XML documents for hits matching on that query.

```
declare function shakes:do-query($queryStr as xs:string?, $mode as xs:string?) {
  let $query := shakes:create-query($queryStr, $mode)
  for $hit in collection($config:app-root)//SCENE[ft:query(., $query)]
  order by ft:score($hit) descending
  return $hit
};
```

In high level, the design of the simple application is as follows:

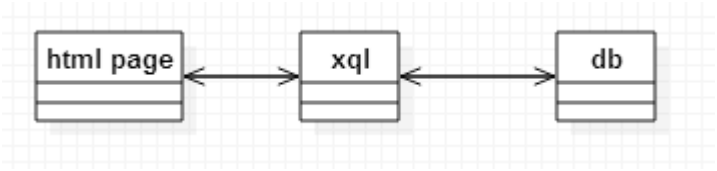


Figure 5 : High-level construction of Shakespear

4. XML Database VS other databases

4.1 XML VS Relational

XML database is a good solution, but not for all the applications. Every DBMS has a different purpose and can function better under specific circumstances. In the section below, there will be some tips provided on when XML database should be chosen as the solution for the application we build or not [13] [14] [15].

XML documents contain the data and the information on how this data is structured in a way that both machines and people can read. An XML document can be digitally transferred to other parties and all the information is carried with it, and that makes it self describing. Because of this self-describing data feature, XML db has become a key standard in Service Oriented Architectures (SOA) and Web Services. XML is also hierarchical. A document expresses the relationship between data items in the form of the hierarchy. Furthermore, sequence and order on elements in doecuments is important.

A relational database, on the other hand, is a storage technology that uses tables and rows in the tables. Data are related with each other with keys. Data is readable by executing SQL in a DBMS tool, but the extraction of the data requires that we have understanding of the database structure, and the foreign key relationships. That knowledge requires significant training and experience. Order and sequence play no role in relational databases.

In general, there are a couple of situations that relation db is better solution than XML, even if the data is already in XML form. Some of these situations are described in this section:

- When process of the data is done on relational form.When data are going to be stored on a relational database as underlying database, then is better to do not include the XML database at all in the application.
- When performance plays an important role. Parsing and interpreting in XML database remains a time-expensive feature. Applications that cannot afford to have even a small loss in performance, should better implement a relational database instead of an XML.
- When the values that should be provided do not really need an hierarchy in the database. If the values that the children nodes provide, have no actual relationship with the parent nodes, then there is no reason to implement an XML db.
- When the data should be represented as table either way when they will be retrieved from database.

On the other hand, XML db can be a better solution than relational db:

- When the schema of the data changes often, then modifying the relational database is time cost and relatively difficult.

- When the nature of data contains hierarchy. Some data are naturally tree structured.Then having a database that keeps the same form and relationship between elements, is more elegant solution.
- When some attributes contain multivalued data. For instance, if we consider that an employee can have multiple phone numbers in his possesion, then creating a table only to store those values is not a good idea.
- When structure of small quantities of data can lead to very big relational schemas. For example, an application that has different levels of data and for every level a different table is needed. Many levels can be represented better in a single XML document.

  Despite their differences, since the XML database can be implemented on top of a relation database (XML Enabled Database), is always a good solution for users. They can select the most effective data model for storing the data.

5. Overview

5.1 Overview of XML Database

XML Database have gained a significant part in database fields over the past decades. The tools mostly used for the implementation of an XML database to an application is Xpath, Xquery, XQL and XSLT. There are two types of XML database: the Native XML Databases and the Enabled XML Databases. That means that either we can use an XML database as a standalone application, or we can integrate the XML database on top of other types (eg relationals). That helps users significantly to implement more efficient applications, since combination of different types of databases can be used on a single application.

Like every other database, XML has also some advantages and disadvantages. We need to have a good understanding of what an XML database can offer to the application we are building, in order to choose it as a solution. Furthermore, we need to know the nature of data we want to store, and what are the demandings of the application , such as performance.

In any case, XML database is worth taking it into consideration, because it will keep evolving on the next years, more tools are coming out, that are more efficient and easy to use, and of course XML technologies play important role nowadays on many applications.

6. References

[1] Wikipedia – XML Database
https://en.wikipedia.org/wiki/XML_database

[2] Wikipedia – XML
https://en.wikipedia.org/wiki/XML

[3] D.Obasanjo : An exploration of XML in database Management Systems
http://www.25hoursaday.com/storingandqueryingxml.html

[4] R.Bourret : XML Database Products
http://www.rpbourret.com/xml/ProdsXMLEnabled.htm

[5] P.Dubois : Using XML with MySQL
http://www.kitebird.com/articles/mysql-xml.html

[6] <XML:DB>
http://xmldb-org.sourceforge.net/

[7] G. Powell : Beginning XML Databases

[8] Xquery
http://www.w3schools.com/xsl/xquery_intro.asp

[9] eXist
http://cdi.uvm.edu/exist/facts.xml

[10] eXist Architecture
https://www.safaribooksonline.com/library/view/exist/9781449337094/ch04.html

[11] eXist Demo
http://exist-db.org/exist/apps/demo/index.html

[12] eXIDE
http://exist-db.org/exist/apps/eXide/index.html

[13] Rich Rollman: XML Tips & Tricks
https://technet.microsoft.com/en-us/library/aa224799(v=sql.80).aspx

[14] F.Font :When to use XML instead of relation database
http://www.room4me.com/index.php?option=com_content&view=article&id=8:xmlvsdb&catid=2:technology&Itemid=5

[15] Comparing XML and relational storage: A best practices guide
http://xml.coverpages.org/IBM-XML-GC34-2497.pdf

[16] J.Robbie :XQL: XML Query Language
http://www.ibiblio.org/xql/xql-proposal.html

[17] A.Chaudhri, A. Rashid, R. Zicari : XML Data Management: Native XML and XML Enabled Database Systems
https://books.google.gr/books?id=7LNhdOeQulQC&pg=PA548&lpg=PA548&dq=xml+enabled+database+architecture&source=bl&ots=jZ2RkXaKSw&sig=HgGULxNOcISiVlb9wMLMvKH10j4&hl=en&sa=X&ved=0ahUKEwiX6Lyu59jJAhVH7g4KHT80B78Q6AEILzAD#v=onepage&q=xml%20enabled%20database%20architecture&f=false