ULB Université Libre de Bruxelles

INFO-H415 Advanced Databases : Group Project

by Prabhdeep Minhas,
and Mi Zhou
26th September 2017

Professor : Esteban Zimányi

Table of contents

1. Abstract

This report reviews the database technology, XML databases, and illustrates the implementation of a chosen scenario in the database management system, BaseX. First, there is a detailed explanation of the scenario, its data organization and the constraints. Then, it is followed by a research on the best data technology and data management system to use for the scenario. On one hand, relational and non-relational databases are compared to each other. On the other hand, various tools are as well compared. For the non-relational databases, Native XML databases and enabled XML databases are set side by side. While for the management platforms, exist-DB and BaseX were the predominant choices. The reasons for choosing Native XML databases and BaseX are presented. Finally, the implementation is described and the results obtained are displayed.

2. Introduction

For a publishing house that needs to store a huge amount of information, we are asked to determine the optimal database technology and management tool for its storage. This publishing house also has to insert, modify, search and eventually delete information into/from its database. In order to choose the ideal tools, multiple comparisons are going to be made. Once the database technology and the management platform are chosen, an implementation of the scenario will take place.

3. Scenario

3.1 Scenario

Working as data analyst, at a publishing house, we have been asked to help the IT department in structuring, storing, and analyzing all the data available. These data involve countless information about articles, books, parts or chapters that are in monograph, papers that are currently in proceeding and those who are already proceeded, the collection of master and Ph.D. thesis, and web pages.

3.2 Data organization

All the data used in our scenario were obtained from an online database that has realistic XML data.[27] In this publishing house, information is separated into different records which are all independent from one another.

All the following records have some common attributes : mdate, key and eventually publtype. Mdate and key correspond respectively to the date of the last modification and the unique dblp key of the concerned record. Publtype is a permissive attribute used to specify the type of the concerned record.

- Article : It is an article from a journal or magazine. There are attributes for the author, the title, the interval of pages (begin-end), the year, the month, the volume, the journal, the number, the ee (which is a web page address) and the URL (which is the position of the record in the database).
- In proceedings : It is a paper in a conference or workshop proceedings. There are attributes for the author, the title, the interval of pages (begin-end), the year, the book title, the ee, the cross-reference and the URL.
- Proceedings : It is the proceedings volume of a conference or workshop. There are attributes for the editor, the title, the book title, the publisher, the series, the year, the ee and the URL.
- In Collection : It is a part or chapter in a monograph. There are attributes for the author, the title, the interval of pages (begin-end), the year, the book title, the ee, the cross-reference and the URL.
- WWW : It is a web page. There are attributes for the author, the title, the year and the URL which is here referring to a web page address
- Master thesis : It is a Master's thesis. There are attributes for the author, the title, the year and the school.
- Ph.D. thesis : It is a Ph.D.'s thesis. There are attributes for the author, the title, the year and the

school.

- Book : It is an authored monograph or an edited collection of articles. There are attributes for the author, the title, the year, the publisher, the ISBN, International Standard Book Number, and the URL which is a web page address.

3.2.1 Constraints

In order to have a program that is accurate and that works logically, the system has to set some constraints. Here are the ones defined for our scenario :

— Each key has to be unique.

— Every mdate's year of a record has to be greater or equal to the year in which it was written.

— The beginning page has to be smaller than the end page for an article, in proceeding papers, and in collection papers.

— All the numbers included in the records have to be positive integers. For example, the interval of pages and the year have to be positive integers.

— The ISBN, International Standard Book Number, has to be unique for each book.

— If the month is specified, it has to be an existing one.

4. Database

4.1 What is a database ?

A database is a systematic collection of information that is meant to be stored and organized in such a way that it can be easily retrieved, accessed, managed and updated. [14]

It is considered as a structured set of data, organized into rows, columns, and tables, where we can access the relevant information in various ways. As said before, the information stored in a database can be modified, we can update the data, expend it or even delete it. [15]

4.2 What is XML ?

XML or eXtensible Markup Language is a text-based generic markup computing language.

A markup language is written with an easy and simple syntax, using a generic text editor. The language leads to a document that is uncomplicated to read in its raw form, as well as defining a document which is distinguishable syntactically from the text. [1] The markup language creates the XML document through the medium of tags. These tags make it possible to structure the data of a document in a hierarchical and organized manner.

When XML was invented the creating team had a big upper hand because the content of it was very similar to HTML, specifically with the traditional markup process. They didn't have to put a lot of effort for the transition or the shifting. Moreover, contrary to HTML, the advantage that XML offers are to be able to use our own tags. XML is not replacing HTML, it just suggests new possibilities by adopting many successful features of HTML. XML was first seen as a data interchange standard, but after some time the purpose got bigger and the language has come to be used as for example serving as the core for development and deployment platforms such as Microsoft's.NET.

However, the main aim of XML is to structure the data manipulated or exchanged by the programs and to transfer information between machines. The XML language is known to be self-describing, allowing to model information systems in a natural and intuitive way. Indeed, even before being used each data field is declared. Also, it has the capacity to store any kind of data. Thus, we have an information-modeling mechanism in which we can characterize what we want to do, instead of mentioning how we have to do it. [2]

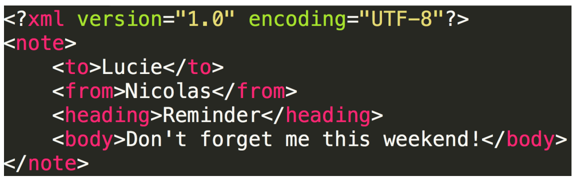
Here are some characteristics XML gives to model information [2]

- Heterogeneity : where we have the possibility to put different data fields to each ”record”. Considering the fact that in the real world, everything is not properly organized into tables, rows, and columns. XML gives us a great benefit by letting us communicate information without any kind of restrictions.
- Extensibility : where we can add a new type of data, without the need of determining it in advance, and in no particular order.
- Flexibility : where each data fields may differ in size and configuration from instance to instance. Therefore, XML puts no restrictions on data.

In order to symbolize information, XML uses four basic components : tags, attributes, data elements, and hierarchy. Each one of these items serves a special role in representing the ”dimension” of information.

Once we have the data which is the data elements, we can identify what they are, thus their tags. After that, it is easy to interpret what attributes they are. Finally, we can use the hierarchy to combine them all together, it helps in understanding how data elements are related to each other.

4.2.1 Example



```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Lucie</to>
    <from>Nicolas</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

FIGURE 4.1 – XML - Example [5]

One of the major questions which is being asked about XML is to know whether an XML document is a database or not. The answer to this question is yes, but in the strictest sense of the term, owing to the fact that it is a collection of data. Indeed, just like any kind of file, it contains some sort of data. [3]

4.3 Relational databases VS Non relational databases

4.3.1 Relational databases

Relation databases are also known as relational database management systems (RDBMS) or SQL databases, in these databases, the information is defined so that it can be reorganized and accessed in a number of different ways. They are composed of a set of tables, that contain columns and rows to store the information. The rows of a relational database are constructed with a collection of an object of the same type. Thus, a relation is defined by the table that collects this information. *Data in a table can be* *related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database.* [14]

In order to make sure that the data is constantly available and stay without any kind of error, relational tables adopt some integrity rules such as select statements, where-from clauses, joins, etc. [14] The language used for querying and maintaining the database by almost every relational database system is SQL (Structured Query Language).

Few examples of the most popular relational databases are Microsoft SQL Server, Oracle Database, MySQL. They are known as robust, simple, compatible and have a great performance in managing the data.[16]

4.3.2 Non-relational databases

Non-relational databases, or most commonly called NoSQL, which stands for "Not only SQL", give a system to store and retrieve information that is modeled in another way than the tabular relations used in relational databases. They are mostly used for real-time web application, large sets of unstructured information, and big data performances issues which are difficult to deal with rational databases. [15]

There are many different motivations behind the use of a non-relational database over a relational database : [15]

- Simplicity of design : we don't have to bother about the " impedance mismatch " between the data, as well as about the tables and rows of a relational database. This leads to control less code to write, debug or maintain.
- Using documents : the data is at one place, and not contained in various tables.
- All kinds of data : this allows to have a flexible database with unstructured and semi-structured data, due to the fact that NoSQL has not any schema.
- Speed : As we don't have tables here, we don't have to join them, thus operations are faster.
- Cost : Non-relational databases don't rely on expensive servers and storage systems. Moreover, most of these databases are openSource.

MongoDB, DocumentDB, Neo4j and XML database are the most popular NoSQL databases.[16]

XML databases

An XML database is a system that leads to specify data and store a huge amount of them in an XML format. We can do multiple operations on these, for example, it can be queried, transformed, exported to the desired format, and so on.

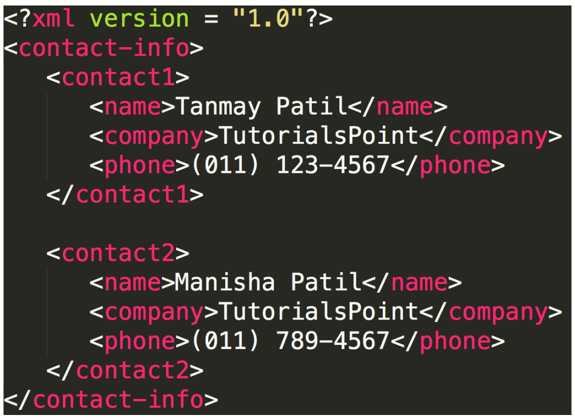
Here are some advantages that XML offers us when we consider it as a database format [4] :

- Self-describing

- Unicode
- Describe data in a tree or graph structure
- Store any well-formed XML
- Support XQuery, XSLT, XUpdate, XInclude
- For multiple interfaces

But on the other hand, it also has a disadvantage ; because of the parsing and text conversion, the access to the XML data is slowed down.

Example



```
<?xml version = "1.0"?>
<contact-info>
   <contact1>
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </contact1>

   <contact2>
      <name>Manisha Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 789-4567</phone>
   </contact2>
</contact-info>
```

FIGURE 4.2 – XML Database - Example [6]

A. Enabled XML Database The first major type of XML database is the enabled XML database.

An enabled XML database is a relational database. This database is constituted of tables in which the data is stored, consisting of rows and columns. Thus, an XML enabled database usually refers to the extension provided for the conversion of an XML document. [6]

Typically, these kinds of databases will map all the XML to a traditional database, accepting XML as input and rendering XML as output. They use a mapping layer which is given by the database itself or by a third party. The role played by this mapping layer is to manage the storage and retrieval of XML data. [7]

B. Native XML Database The second major type of XML database is the native XML database, or commonly called the NXD.

A native XML database is a database that relies on the data model provided by XML, by using query languages such as XPath or XQuery. The NXD is not based on a table format, but on the container. It has the capability of storing a huge amount of XML document and data. An XML database is usually referred as a native XML database if it provides direct XML storage. [9] In comparison to the enabled XML database, the NXD has more advantages, indeed, they are more likely ideal for storing, retrieving, querying denormalized data and maintaining the XML document. [6]

There are different reasons why we should store the data in XML documents in a native XML database. First of all, if the data is semi-structured, it means that the data has a regular structure. If mapping its structure leads to having either a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient), it would signify that the structure of the data varies a lot and so it is better to store it in an NDX in the form of an XML document.

The second argument is the retrieval speed. In point of fact, a native XML database is able to retrieve data much faster than a relational database. The strategies used by this database is to store entire documents together physically or use physical (rather than logical) pointers between the parts of the document. As it doesn't have logical joins, it is faster to obtain information than a relational database.

Another argument to use a native XML database is that it allows us to actually adopt XML-specific capabilities. [8]

4.3.3 Final choice

XML Database	**Relational Database**
Hierarchical	Represented in a model of logical relationships
Self-describing	Not self-describing
Has inherent ordering	Does not have inherent ordering
Contains collections	Contains tables

Table 4.1 – Differences between an XML database and a relational database [7]

Relational databases are in a way limited because each item can only contain one attribute. Different rows link that information through the use of several keys. But in non-relational databases, we don't have to deal with this kind of issue.

In our scenario, we have a lot of information stored in the database, so it is way easier to collect that information in one place rather than having several tables to do that job. Moreover, if we choose to use a relational database, the system can get slow in terms of response time owing to the fact that we have a very large amount of data to store. [17]

The most important argument of choosing the native XML database which is non-relational is that all the documents containing the data are in an XML format, so it is obviously better to use an XML database, over any other NoSQL database. The merge of XML and a relational database would have lost the flexibility XML gives us.

Thus, for the scenario explained before, we have decided to use a non-relational database, the native XML database.

5. Selection of a native XML database

For the many reasons disclosed in the previous chapter, we decided to use a native XML database. We must now determine which database management system is the most suitable for our scenario.

5.1 Comparison of native XML databases

Diverse native XML databases can be taken into consideration for the implementation of our scenario.

Here are the native XML databases considered from the DB-Engines Ranking of Native XML DBMS[18] :

Name	**License**	**Description**	**Primary database model**
MarkLogic Server[21]	Commercial	Enterprise NoSQL DBMS for JSON, XML, and RDF	*Multi-Model* Document store, Native XML DBMS RDF store, Search engine
Oracle Berkeley DB[22]	Open Source	Widely used in-process key-value store	*Multi-Model* Key-value store Native XML DBMS
Virtuoso[23]	Open Source	Multi-model hybrid-RDBMS Support management of data represented as relational tables and/or property graphs	*Multi-Model* Graph DBMS, Native XML DBMS, Relational DBMS, RDF store
BaseX[20]	Open Source	Light-weight Native XML DBMS with support for XQuery 3.0 and interactive GUI	Native XML DBMS
webMethods Tamino[24]	Commercial	Server integrated in Software AG's webMethods platform	Native XML DBMS
Sedna[19]	Open Source	Full range of core database services - persistent storage, ACID transactions, security, indices, hot backup.	Native XML DBMS
eXist-db[25]	Open Source	NoSQL databases built on XML technology	Native XML DBMS
searchxml[26]	Commercial	DBMS for structured and unstructured content wrapped with an application server	*Multi-Model* Native XML DBMS, Search engine

Table 5.1 – Comparison of Native XML Databases

First, all the native XML databases with a commercial license were eliminated from the selection of database management systems. Some of them offer a free version. Unfortunately, it usually comes with restricted storage and processing capabilities for personal and non-commercial use. The multi-model databases were also ruled out since our data are exclusively in XML formats. Choosing a database management system that is organized around a sole data model helps us to focus on handling a single format of documents and avoid futile confusions about how to organize, store and manipulate our data.

Thus, a precise comparison of the remaining native XML databases was conducted.

Name	**Database Type**	**Supported APIs**	**XML Querying**
BaseX[20]	Proprietary	XQJ, XML :DB, RESTful, RESTXQ, WebDAV	XQuery 3.0
eXist-db[25]	Proprietary	XQJ, XML :DB, RESTful, RESTXQ, WebDAV	XQuery 1.0
Sedna[19]	Proprietary	XQ, XML :DB	XQuery 1.0

TABLE 5.2 – Comparison of open source single data model native XML databases

The features of the remaining database management systems are quite similar. Sedna seems to be lacking in terms of APIs' support. On the other hand, eXit-DB and BaseX provide a wide and identical range of supported APIs. It is quite arduous to determine which one is more fitting for our implementation. However, BaseX allows XML Querying in the latest XQuery version. As a result, BaseX was chosen as the best choice for our scenario's implementation. There is a complete description of the chosen Native XML Database, BaseX, in the following section.

5.2 BaseX

BaseX is a native, light-weight and scalable XML Database engine with high performance. It is also known as a platform-independent and distributed XML Database management system under the Berkeley Software Distribution (BSD) License, which is a permissive free software license. It primarily started as a community project on Github[10] in 2007 to become the useful XML Database tool that we know today.

BaseX is written in Java and is NoSQL, which means that it is part of *"a class of database management systems (DBMS) that do not follow all of the rules of a relational DBMS and cannot use traditional SQL to query data"*[11]. Unlike the alternative document-oriented databases, it uses standardized query languages such as XPath and XQuery that are accordant to World Wide Web Consortium (W3C) specifications and the official Update and Full-Text extensions. XML Path Language, known as XPath, is a query language using path expressions to choose nodes or a set of nodes in XML documents. XQuery is the language for querying XML. XQuery does in XML, which stands for Extensible Markup Language, what SQL, which stands for Structured Query Language, does typically in databases. Its purpose is to find and to extract components and attributes from XML documents. BaseX supports the new features of the XQuery 3.1 and the XQuery 3.1 Functions and Operators specification.

Visualizing, querying and storing large XML documents and JSON documents and collections are the main uses of BaseX. Furthermore, this management system is an in-memory database. Therefore, the data is stored in compressed form in Random-access memory (RAM) and not in the disk storage. The performance is improved since the data access is faster. In contrast, it makes it difficult to modify the data. Memory databases like BaseX are mostly used to store and analyze data for this main reason.

5.2.1 Graphical User Interface

The Graphical User Interface allows interactive searches, explorations, and studies of the data. The XPath/XQuery expressions are constantly evaluated in real-time.

BaseX can be downloaded from its official website : http ://basex.org/.

Once BaseX is acquired on your device, the GUI instance can be launched by a double-click on the BaseX GUI start icon on your device or by running the BaseX GUI script according to the operating system.

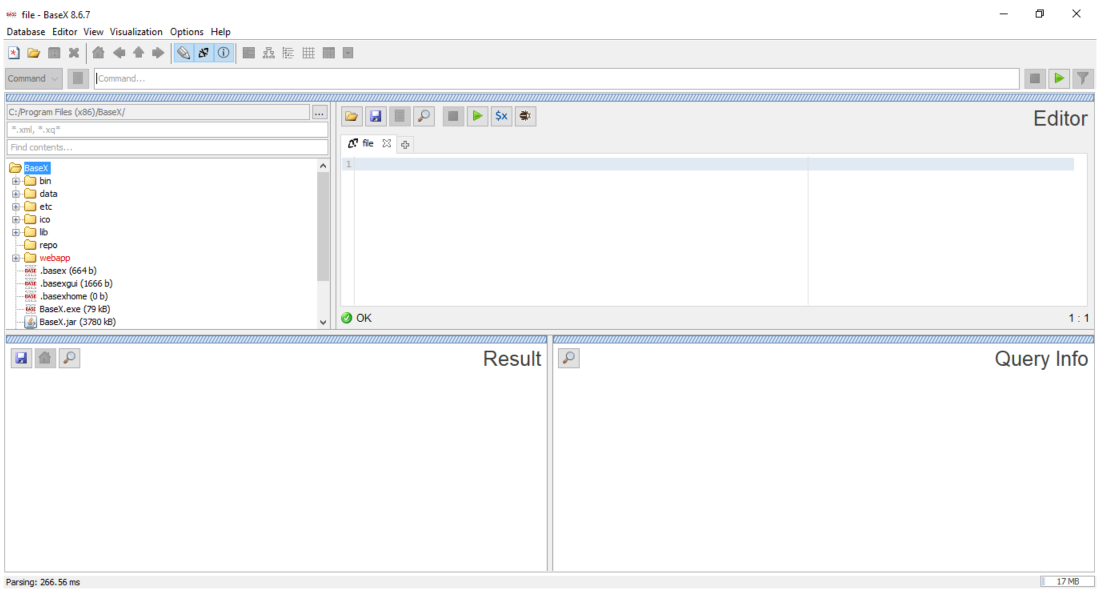


FIGURE 5.1 – BaseX - Graphical User Interface

The graphical user interface is composed of :

— A menu bar

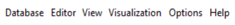


FIGURE 5.2 – BaseX - Menu Bar

— A toolbar. It used to control the windows shown, to provide the possibility to add or delete a window or a view in the interface. Each view shows a specific perspective of the manipulated data.

FIGURE 5.3 – BaseX - Tool Bar

— A small drop-down menu for command execution (Command Option), query execution (XQuery Option) and searches in the database (Find Option)

FIGURE 5.4 – BaseX - Drop-down Menu

— Windows showing different aspects of the system. (Editor, Project View, and Visualizations)

5.2.2 Starting with BaseX

To create a database, you can either use the command line CREATE DATABASE name or select Database -> New in the menu bar and browse to an XML document of your choice.

Next to the input bar, there is a drop-down menu. This menu provides three modes : Find, XQuery and Command.

In the Find mode, the user can type single symbols or texts in the input bar to find them in the ongoing database.

Here are some examples of find queries :

Query	**Description**
city	Find elements named city, and texts containing this token.
=India	Find texts matching the exact string India.
Cing	Find texts equal or similar to the token Cingdom.
id	Find attributes named id and attribute values containing this token.
@=f0_119	Find attribute values matching the exact string f0_119.
”European Chinese”	Find texts containing the phrase ”European Chinese”.
//city	Leading slash : Interpret the input as XPath expression.

TABLE 5.3 – Find queries’ examples - Graphical User Interface[12]

The user can enter XPath and XQuery expressions in the input bar in the XQuery mode.

Here are some examples of XPath and XQuery expressions :

Query	**Description**
//country	Return all country elements.
//country[name = "Switzerland"]	Return the country element of "Switzerland".

for $city in //city
where $city/population > 1000000
order by $city ascending
return $city/name
	Return the names of all cities with a
	population larger than one million
	and order the results by the
	name of the city.

Table 5.4 – XPath and XQuery expressions' examples - Graphical User Interface[12]

In the Command mode, the user can enter BaseX Commands[13]. The commands will be executed.

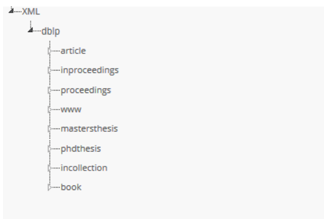
More details about BaseX GUI can be found here : http ://docs.basex.org/wiki/Graphical_User_Interface

5.2.3 Reasons to choose BaseX

Nowadays, large amounts of data on Open Data Platforms are accessible in XML format as this format eases the storing process. Therefore, using the data in their XML format in the database management system DBMS is very compelling. BaseX is the perfect candidate for handling distributed XML data. Indeed, it applies XML querying concepts such as XPath and XQuery. They are both well developed, well-known query languages. Moreover, a collection of files can be feed to Basex and it will admit visualizing, querying and storing these files as a single root document.

6. Implementation

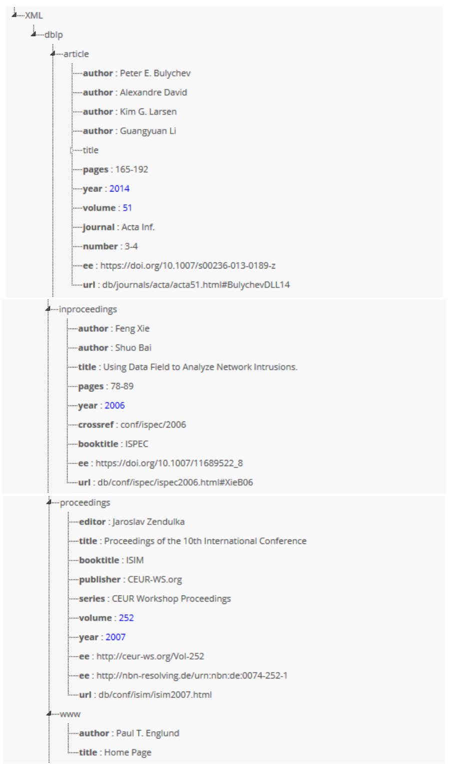
As mentioned before, the database is organized in different tables such as articles, books, inproceedings, proceedings, incollection, www, master thesis and Ph.D. thesis. Each table has its respective attributes. The data organization of our scenario can be represented by the following scheme.



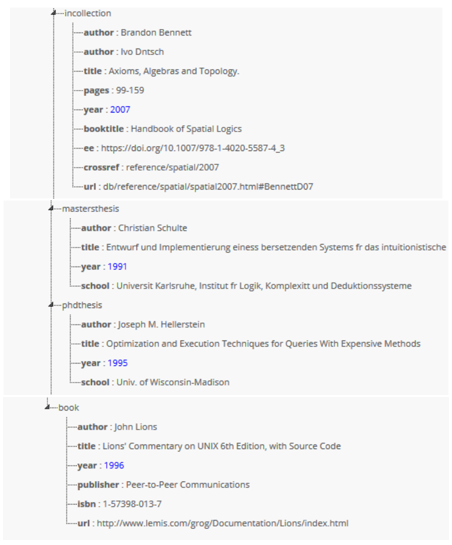
```
XML
  dblp
    article
    inproceedings
    proceedings
    www
    mastersthesis
    phdthesis
    incollection
    book
```

Figure 6.1 – Database Main Tree View

For each table, there exists a detailed tree view. Only the attributes key and ISBN are unique. All the others attributes can be duplicated.



```
XML
  dblp
    article
      author : Peter E. Bulychev
      author : Alexandre David
      author : Kim G. Larsen
      author : Guangyuan Li
      title
      pages : 165-192
      year : 2014
      volume : 51
      journal : Acta Inf.
      number : 3-4
      ee : https://doi.org/10.1007/s00236-013-0189-z
      url : db/journals/acta/acta51.html#BulychevDLL14
    inproceedings
      author : Feng Xie
      author : Shuo Bai
      title : Using Data Field to Analyze Network Intrusions.
      pages : 78-89
      year : 2006
      crossref : conf/ispec/2006
      booktitle : ISPEC
      ee : https://doi.org/10.1007/11689522_8
      url : db/conf/ispec/ispec2006.html#XieB06
    proceedings
      editor : Jaroslav Zendulka
      title : Proceedings of the 10th International Conference
      booktitle : ISIM
      publisher : CEUR-WS.org
      series : CEUR Workshop Proceedings
      volume : 252
      year : 2007
      ee : http://ceur-ws.org/Vol-252
      ee : http://nbn-resolving.de/urn:nbn:de:0074-252-1
      url : db/conf/isim/isim2007.html
    www
      author : Paul T. Englund
      title : Home Page
```



```
incollection
    author : Brandon Bennett
    author : Ivo Dntsch
    title : Axioms, Algebras and Topology.
    pages : 99-159
    year : 2007
    booktitle : Handbook of Spatial Logics
    ee : https://doi.org/10.1007/978-1-4020-5587-4_3
    crossref : reference/spatial/2007
    url : db/reference/spatial/spatial2007.html#BennettD07
mastersthesis
    author : Christian Schulte
    title : Entwurf und Implementierung einess bersetzenden Systems fr das intuitionistische
    year : 1991
    school : Universit Karlsruhe, Institut fr Logik, Komplexitt und Deduktionssysteme
phdthesis
    author : Joseph M. Hellerstein
    title : Optimization and Execution Techniques for Queries With Expensive Methods
    year : 1995
    school : Univ. of Wisconsin-Madison
book
    author : John Lions
    title : Lions' Commentary on UNIX 6th Edition, with Source Code
    year : 1996
    publisher : Peer-to-Peer Communications
    isbn : 1-57398-013-7
    url : http://www.lemis.com/grog/Documentation/Lions/index.html
```

FIGURE 6.2 – Tree View of each table

Several actions such as search, insert, update and delete are allowed on the data. The administrators of the database use these actions to keep it up-to-date. In case of an insertion and an update, the constraints specified in our scenario must be respected. Their implementation will be explained in details for an insertion and an update.

6.1 Search

To search some information in the database, the user can utilize the Find mode in BaseX to select the requested information. For further information, consult the " Starting with BaseX " section.

Another way to perform searches in the database would be by creating a file *.xq* that would contain the XQuery for the requested search. For instance, in order to find all the titles of the articles that have the value of the attribute *volume* bigger than 2, we can run the following code by importing the file *search.xq* in the BaseX editor :

```
for $article in //article
where $article/volume > 2
return $article/title
```

6.2 Insertion

An insertion in the database is needed when a new element or attribute is registered. It can be seen as adding a new table or a new attribute to an existing table in the database. An insertion can be executed in the XQuery mode in BaseX or by importing a corresponding *.xq* file. In both cases, the code executed has to respect the constraints. The use of a file *.xq* allows the implementation of functions that constantly check if the insertions respect the constraints.

The file *queries.xqy* contains the functions that verify the date, the interval of pages, the ISBN of the books and the month that are eventually inserted. For each constraint, a function was created to verify if it is respected during insertions.

- Each key has to be unique : for a key given in parameter, we have to check in each table in dblp, if it already exists or not.

```
declare function queries:checkKey($key)
as xs:boolean
  {
    not((doc("newDB.xml")/dblp/article[@key =$key])
        or(doc("newDB.xml")/dblp/inproceedings[@key =$key])
        or(doc("newDB.xml")/dblp/proceedings[@key =$key])
        or(doc("newDB.xml")/dblp/www[@key =$key])
        or(doc("newDB.xml")/dblp/masterthesis[@key =$key])
        or(doc("newDB.xml")/dblp/phdthesis[@key =$key])
        or(doc("newDB.xml")/dblp/incollection[@key =$key])
        or(doc("newDB.xml")/dblp/book[@key =$key]))
  };
```

- Every mdate's year of a record has to be greater or equal to the year in which it was written.

```
declare function queries:checkDates($mdate, $year)
as xs:boolean
  {
    let $date := substring($mdate, 1, 4)
    return($date >= xs:string($year))
  };
```

- The beginning page has to be smaller than the end page for an article, in proceeding papers, and in collection papers.

```
declare function queries:checkPages($page)
as xs:boolean
  {
    if(empty($page)) then (
      true()
    )
    else(
      xs:integer(tokenize(xs:string($page),"-")[1])
        <= xs:integer(tokenize(xs:string($page),"-")[2])
    )
  };
```

- All the numbers included in the records have to be positive integers. For example, the interval of pages and the year have to be positive integers : for this constraint, there is no need to do a separate function. Indeed, we can directly specify that the attributes year or volume, for instance, have to be integers, otherwise, BaseX shows an error and the operation can't happen.

- The ISBN, International Standard Book Number, has to be unique for each book : because the ISBN is present in only two tables proceedings and books, the program has to check the existence of ISBN in these ones.

```
declare function queries:checkISBN($isbn)
as xs:boolean
  {
    if(empty($isbn)) then (
      true()
    )
    else(
      not((doc("newDB.xml")/dblp/proceedings[isbn =$isbn])
         or (doc("newDB.xml")/dblp/books[isbn =$isbn]))
    )
  };
```

- If the month is specified, it has to be an existing one : we have decided to declare all the possible ways of writing months. Thus, we can note that **January** and **january** will both be accepted, but **JANUARY**, for instance, won't.

```
declare function queries:checkMonth($month)
as xs:boolean
  {
    if(empty($month)) then (
      true()
    )
    else (
      let $months := ('January', 'February', 'March', 'April', 'May',
      'June', 'July', 'August', 'September', 'October', 'November',
      'December','january', 'february', 'march', 'april', 'may'
      , 'june', 'july', 'august', 'september', 'october', 'november',
       'december')
    return($months = $month)
```

```
        )
      };
```

A function returns **True** if the constraint is respected and allows the insertion. Otherwise, the function returns **False** and no data is inserted.

For instance, a simple insertion of an attribute in an existing table can be performed as followed :

```
import module namespace articles="queries" at "./queries.xqy";

declare variable $key := "journals/acta/ChungES85";
declare variable $author := "Moon-Jung Chung";

if (articles:updateDB($key)) then(
  insert node <author> {$author} </author>
  into /dblp//article[@key=$key])
else()
```

The code above represents the insertion of a new author in an article using the unique key. For every insertion, the unique key has to be known. It allows us to confirm the existence of the table concerned since a new attribute can't be inserted into a table that doesn't exist.

Adding a new table can be done by the following code :

```
import module namespace masterthesis="queries" at "./queries.xqy";

declare variable $mdate := "2017-05-28";
declare variable $key := "journals/acta/ZhouS85";
declare variable $author := "Zhou";
declare variable $title := "A new title";
declare variable $year := 2015;
declare variable $school := "ULB";

if (masterthesis:addInDB($mdate, $key, empty, $year, empty, empty)) then(
  insert nodes
  <masterthesis mdate="{$mdate}" key="{$key}">
  <author>{$author}</author>
  <title>{$title}</title>
  <year>{$year}</year>
  <school>{$school}</school>
  </masterthesis>
  into /dblp)
```

```
else()
```

This code adds a new master thesis to the database. The mdate and key have to be known, and for the remaining attributes we are free to add them or not, and we can multiply them as well. At first, the attributes that are subject to the constraints are verified by the corresponding function with the function *addInDB* and the functions that it calls.

```
declare function queries:addInDB($mdate, $key, $page, $year as xs:integer, $month,
$isbn, $volume as xs:integer)
as xs:boolean
  {
   (queries:checkDates($mdate, $year)
     and (queries:checkPages($page)
     and (queries:checkKey($key)))
     and (queries:checkMonth($month))
     and (queries:checkISBN($isbn)))
  };
```

This function will check for every attributes if the constraints are respected. In the example just above, we can observe that for certain parameters **empty** is sent to the function. This will just ignore those parameters when the corresponding functions are called. And for the parameters $year and $volume, it is specified that they have to be an integer, if they aren't the query is not executed. If the *addInDB* returns **True**, then the new table is inserted in the database.

6.3 Update

An update is really similar to an insertion. It differs in a few aspects. To update a table, only the key and the value of the attribute that we want to update have to be known. Before the update, we solely need to verify that the key exists with the function *updateDB*.

```
declare function queries:updateDB($key)
as xs:boolean
  {
   (queries:existKey($key))
  };

declare function queries:existKey($key)
as xs:boolean
  {
        ((doc("newDB.xml")/dblp/article[@key =$key])
        or(doc("newDB.xml")/dblp/inproceedings[@key =$key])
        or(doc("newDB.xml")/dblp/proceedings[@key =$key])
        or(doc("newDB.xml")/dblp/www[@key =$key])
        or(doc("newDB.xml")/dblp/masterthesis[@key =$key])
        or(doc("newDB.xml")/dblp/phdthesis[@key =$key])
        or(doc("newDB.xml")/dblp/incollection[@key =$key])
        or(doc("newDB.xml")/dblp/book[@key =$key]))
  };
```

For instance, in order to update the value of the attribute *volume* of an article, the following code can be executed :

```
import module namespace articles="queries" at "./queries.xqy";

declare variable $key := "journals/acta/Hehner78";
declare variable $volume := 40;

if(articles:updateDB($key)) then (
  for $article in /dblp//article
  where $article[@key=$key]
  let $update := $article/volume
  return (replace value of node $update with $volume)
 )
else ()
```

6.4 Deletion

A table can be deleted from the database by using its unique key. At first, the key has to be verified with the function *existKey*. If it is an existing key, then the table of the concerned data is deleted. For example, the following code deletes the table of the concerned website by using its unique key :

```
import module namespace www="queries" at "./queries.xqy";

declare variable $key := "homepages/14/2161";

if(www:existKey($key)) then (
  for $www in /dblp//www
  where $www[@key=$key]
  return (delete node //www/title)
 )
else ()
```

7. Results

As we have said before, the data used in our project was obtained from an online database [27]. This realistic database has over 50 million lines, with a huge amount of information about books, articles, etc. So, it took quite some time to open this database in BaseX. Once it was done, we could clearly see how it was organized, via the tree structure or even the map structure. It was obviously difficult to work with such a huge database, as the queries or the commands would take a lot of time.

Therefore, we decided to create a smaller database compared to the original one. With this smaller version, it would be easier to observe if the queries are correctly made, or that the constraints are respected. The XML file containing this smaller database is called **smallDblp.xml**.

At first, we tried to execute the obvious queries in BaseX, such as a search or an insert in the database via the BaseX command line. This was quite straightforward to do, as we just had to implement what the documentation of BaseX told us. This part was explained in the section **5.2.2 Starting with BaseX**.

After that, we had to implement the different constraints of our scenario. To do so, we thought that triggers were the only way to implement the contraints. Unfortunately, triggers are not yet completely implemented in BaseX, the code is still under construction as we can see in GitHub. [28]

Thus, we decided to create our own functions that will do the same job. Before any update or add in the database, we have to call a function that will check if all the constraints are respected. All of these XQuery functions are located in the file called **queries.xqy**. For instance, before adding any kind of table we have to call the function **addInDB()**, that will make sure the constraints are respected. If there are other constraints that have to be added to the system, we will just create a new function, and insert it in the **addInDB()** function. With this system of functions, we can say that it is not so different from triggers, it does practically the same thing but in a slightly different way.

Because our data were stored in a XML format, the BaseX platform really helped us in the storing process, it was the ideal candidate to choose for handling distributed XML data. We were also able to practice XQuery, which is a well-developed query language. Moreover, BaseX allowed us visualizing such a massive database. This platform was really efficient for searching and adding information but it was a bit slow for updating the various tables present in it (for the original database). It wasn't a big problem for our scenario because the main operations in the publishing house are adding or searching information.

Nonetheless, if we want to have an overview of this platform, it might not be the most optimal solution for storing a very large database that modifies itself a lot. Finally, we can say that working with BaseX was a nice experience, as it was uncomplicated to work with and it was visually a great tool to play with. Moreover, it helped us realizing that if a platform does not accept the implementation of triggers, there are other ways to complete the same task.

8. Conclusion

A publishing house needing to store a huge amount of information, as well as modify, insert, delete or search data was presented. After several comparisons, we selected the XML database with the management tool BaseX as the ideal choice for the application of the scenario. The implementation was straightforward to do. The only difficulty encountered was the use of triggers for the constraints. Sadly, they were not available in BaseX. But fortunately, we overcame this problem by creating functions in XQuery acting as triggers.

Bibliography

[1] Structure Data XML https ://openclassrooms.com/courses/structurez-vos-donnees-avec-xml

[2] XML data management http ://etutorials.org/XML/xml+data+management/

[3] XML databases http ://www.rpbourret.com/xml/XMLAndDatabases.htm

[4] Native XML databases http ://www.bcs.org/upload/pdf/cwallace-120608.pdf

[5] XML examples https ://www.w3schools.com/xml/xml_display.asp

[6] XML databases tutorial https ://www.tutorialspoint.com/xml/xml_databases.htm

[7] XML databases slides https ://www.slideshare.net/hawlery1989/xml-databases

[8] Storing XML databases https ://nativexmldatabase.com/2010/09/28/5-reasons-for-storing-xml-in-a-database/

[9] XML databases definitions https ://www.techopedia.com/definition/30558/xml-database

[10] BaseX Github https ://github.com/BaseXdb

[11] NoSQL https ://www.techopedia.com/definition/27689/nosql-database

[12] Graphical User Interface http ://docs.basex.org/wiki/Graphical_User_Interface

[13] BaseX Commands http ://docs.basex.org/wiki/Commands

[14] Database oracle https ://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[15] Database definition http ://searchsqlserver.techtarget.com/definition/database

[16] RD vs Non-RD http ://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/

[17] Database volume https ://www.guru99.com/introduction-to-database-sql.html#7

[18] DB-Engines Ranking of Native XML DBMS https ://db-engines.com/en/ranking/native+xml+dbms

[19] Sedna http ://www.sedna.org/

[20] BaseX http ://basex.org/

[21] MarkLogic http ://www.marklogic.com/

[22] Oracle Berkeley DB http ://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html

[23] Virtuoso https ://virtuoso.openlinksw.com/

[24] webMethods Tamino http ://techcommunity.softwareag.com/ecosystem/communities/public/webmethods/products/tamino/

[25] eXist-db http ://exist-db.org/exist/apps/homepage/index.html/

[26] Searchxml http ://www.searchxml.net/

[27] DBLP http ://dblp.uni-trier.de/

[28] Triggers in BaseX https ://github.com/BaseXdb/basex/