INFO-H415 - Advanced Database :
Group Project

by Richard Bauwens,
and Marine Devers
17th December 2018

Professor : Esteban Zimanyi

Table of contents

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 an explanation of our scenario, followed by the data structure and the constraints used. Then, there is a few explanations about what is a database and what is XML followed by the choices made. There are also some information about BaseX and why we use it. Finally, there is the process of our implementation with some examples to illustrate our database.

1. Introduction

For this project, we were asked to study a database technology and illustrate it with an application developed in a database management system. We decided to study the XML databases using BaseX. Once our subject found, we were asked to find a scenario using, in our case, XML and BaseX. Then, we will have to make some research about our topic and explain our choices. Finally, a database will have to be implemented.

2. Scenario

2.1 Scenario

We are two students in a small university in the countryside, and our university recently got a small government bonus to modernize their IT infrastructure. Their main idea was to develop a mobile application, along with an API, to help students with their schedule. The app and API already taking a lot of the bonus, we are tasked with the preliminary work before the professional developer team comes in.

Our university being really old-fashioned, and small, they still use *Excel* sheet to manage their students, classes and personnel. Our task is to help them going from there to an easy to manage database, allowing the development of an API (and ultimately a mobile app).

The old secretary also really likes her desktop *Excel* GUI, and would prefer if the new infrastructure would let her have a simple GUI instead of some complicated desktop app.

2.2 Data structure

All the data used in our project were obtained from the old *Excel* files that the secretary used before the update. The whole database of the university is separated into three main categories :

- **Student** which contains the name of its studies, its year of study, its name, its birthdate and its id.
- **Personnel** which contains its job also called status (teacher, cleaning, restaurant...), its name, its birthdate, its bank account and its id.
- **Class** which contains its name, the teacher's name who teaches it, its mnemonic and the room where the class is given with the name of the room, the time and the week the class is given.

2.3 Constraints

Our program must be accurate and work logically. This is why the system must have to set some constraints. Here are the ones we defined for our scenario :

- The personnel/student id's are unique.
- Two different classes can not be given at the same time in the same local.

- Two different classes can not be given at the same time by the same teacher, and the teacher must be registered in the database.
- The status of every personnel should be in the list of accepted status.
- Everything has to be named in the university (students, personnel and classes).
- Every student must be registered in one of the accepted study courses.
- Mnemonic are being used as id's for classes, therefore they should be unique and never empty.

3. Database

3.1 What is a database ?

In the most general sense, a database is an organized collection of data. It is more specifically an electronic system that allows data to be easily accessed, manipulated and updated. It is usually used by an organization as a method of storing, managing and retrieving information.[1]

"Data is organized into rows, columns and tables, and is indexed to make it easier to find relevant information".[2]

3.2 What is XML ?

XML stands for *eXtensible Markup Language* and is used to describe data. The XML standard helps us, in a flexible way, to create information formats and to electronically share structured data via internet.[3]

XML data is considered as self-describing (or self-defining) and informationally complete. In other words, it means that the structure of the data is embedded with the data. Hence, there is no need to pre-build the structure to store the data that just arrived. It is dynamically understood within the XML, meaning that applications can use this feature to automatically build themselves with little or no programming required

XML defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML documents consist entirely of characters from the *Unicode* repertoire. However, there exists a small number of specifically excluded control characters that are not defined by *Unicode*.[4] XML allows us to model information in a natural and intuitive way.

Here are some powerful capabilities that XML brings to information modeling [5] :

- **Heterogeneity** : each "record" can contain different data fields. The real world is not neatly organized into tables, columns and rows. There exists a great advantage in being able to express information, as it exists, without restrictions.
- **Extensibility** : new types of data can be added at will and they do not need to be determined in advance. This allows us to embrace, rather than avoid, change.
- **Flexibility** : data fields can vary in size and configuration from instance to instance. XML imposes

no restrictions on data. Each data element can ba as long or as short as necessary.

XML uses four basic components to express information. Each of these components serves a unique purpose meaning that each represents a different ”dimension” of information. **Data elements** represent data which is meaningless unless we know what the data definitions are. XML adds context to data, giving it meaning by adding **tags** that describe what data elements are. The **attributes** give us more information about data elements or about how to interpret them. Finally, the **hierarchy** helps us to determine how to string everything together and to understand how they are related to each other.

3.2.1 Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Chandler</to>
  <from>Monica</from>
  <heading>Reminder</heading>
  <body>Don't forget to clean the house!</body>
</note>
```

FIGURE 3.1 – Example - note.xml [6]

where <to>, <from>, <heading> and <body> have text content, and <note> has element contents but no attributes.

3.3 Relational databases vs Non Relational databases

3.3.1 Relational databases

A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.[7] These databases are also known as relational database management systems (RDBMS) or SQL databases.

The relational model organizes data into one or more tables which are sometimes called *relations*. Each table contains one or more columns representing the data categories, also called *attributes*, and one or more rows representing a unique instance of date, or *key*, for the categories defined by the columns.

The main advantages are that they give to the user the possibility to easily categorize and store data that can later be queried and filtered to extract specific information. These databases are also easy to extend and they are not reliant on physical organization. Moreover, the data is stored just once, which eliminates data deduplication. Relational databases are flexible (easy for users to carry out), collaborative which means that multiple users can access the same database, trusted and secure, meaning that data in tables within RDBMSes can be limited to allow access by only particular users.

The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2. [8]

3.3.2 Non Relational databases

A non-relational database is any database that does not incorporate the table/key model that RDBMSes promote.[9] In other words, they give a system to store and retrieve information that is modeled in another way than the tabular relations. They are mostly used in big data, large sets of unstructured information, and real-time web applications.

There exist many different motivations behind the use of a non-relational database instead of a relational one :

- **Simplicity of design** : we do not have to deal with the *impedance mismatch* between the data, as well as about the tables and rows of a relational database. Doing so, we have less code to write, debug, and maintain.
- **Better horizontal scaling** : using documents is much easier because all the data is at one place, instead of being contained in various tables.
- **Finer control over availability** : we can add or remove servers without causing an application downtime. Indeed, non-relational databases store multiple copies of data across the cluster to ensure high availability and disaster recovery.
- **Flexible** : non-relational databases can easily and quickly accommodate to any new type of data without being disrupted by content structure changes.That is because document database are schemaless.
- **Speed** : many operations are faster in non-relational databases than in relational databases due to not having to join tables.
- **Cost** : non-relational databases use clusters of cheaper commodity servers than the ones that RDBMSes tend to rely on.

Usually, we call a non-relational database *NoSQL*, also called *Not Only SQL* to emphasize that they may also support SQL-like query languages. The most popular non-relational databases are MongoDB, DocumentDB, Neo4j and XML database.

3.3.3 Comparison

Features	Relational	Non-Relational
Data Type	Hard to store rich data type	Able to store any type of data
Cost	Expensive to build and to maintain	Around 10% cost to relational database
Representation	Data in tables and rows	Data as collections of JSON documents
Query	SQL	Object querying
Multi-originated	JOIN operation	Multi-dimensional data type
Schema	Require to define tables and columns before storing	No schema

TABLE 3.1 – Comparison between relational and non-relational databases [10]

3.4 XML databases

An XML database is a database that stores data in XML format. Its data can be queried, transformed, exported and returned to a calling system. Usually, this kind of databases is not a relational database which stores data according to their relation to other pieces of data. [11]

An XML database offers some advantages such as :

- It is **self-describing**, meaning that it contains not only the data but also the meta data.
- It describes data in a **tree** or **graph structure**.
- It uses **Unicode**.
- It supports **XQuery, XSLT, XUpdate, XInclude**.
- It can be used for **multiple interfaces**.

On the other side, it has also a big disadvantage ; the access to the XML data is slowed down because of the parsing and the text conversion.

3.4.1 Example

```
<?xml version = "1.0"?>
<contact-info>
   <contact1>
      <name>Bill Gates</name>
      <company>Windows</company>
      <phone>(011) 123-4567</phone>
   </contact1>

   <contact2>
      <name>Steve Jobs</name>
      <company>Apple</company>
      <phone>(011) 789-4567</phone>
   </contact2>
</contact-info>
```

FIGURE 3.2 – Example - XML database [12]

Here is a table of contacts that holds records of contacts (Bill Gates and Steve Jobs), which in turn consists of three entities : name, company, and phone.

3.4.2 Enabled XML Database

The first major type of XML databases is called the enabled XML database. It is a relational database constituted of tables consisting of rows and columns, where data is stored. Thus, an XML enabled database usually refers to the extension provided for the conversion of XML document. [12]

This kind of database will usually map all XML to a traditional database that will accept XML as input and give XML as output. It uses 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. [13]

3.4.3 Native XML Database

The second major type of XML databases is called the native XML database, or commonly called the NXD. It is a database that is based on the container rather than table format. Each container can contain a large amount of XML documents or XML data.[12]

In comparison to the enabled XML database, the native XML database has more advantages to offer. Indeed, they are much better capable of storing, maintaining and quering large amounts of XML documents than relational databases. Moreover, no complicated designs have to be built before setting up the database. [14]

There are several reasons to prefer using the native XML database rather than enabled 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 mostly signify that the structure of the data varies a lot. Then, it is surely better to store it in an native XML database in the form of an XML document.

The second reason is the retrieval speed. Indeed, a native XML database is able to retrieve data much faster than a relational database. The explanation for this stands in the strategies used by the native XML databases which store entire documents physically together or use physical (rather than logical) pointers between the parts of the document. Because it does not have logical *joins*, it is faster to obtain information than a relational database.

Another reason to use a native XML database instead of an enabled XML database is that it allows us to actually adopt XML-specific capabilities. [15]

3.5 Why an XML database ?

In the case of our scenario, the choice for the new database is heavily guided towards XML because it allows us to easily convert old *Excel* files (since *Excel* offers an option to save a spreadsheet as XML).

But it is not all, XML is (almost) human readable and easy to write, meaning that the secretary will have no difficulty understand the database, even raw. Indeed, there is no need for a long SQL query to insert or modify anything. Moreover, XML is supported by (almost) any web technology in a native way. In other words, almost every web technology accepts the XML language without requiring external parsers or libraries. Hence, it is easy to parse and to use. It is also easy to display on a web page or on a web application. In our case, it also implies that the API will almost certainly be compatible with our database.

XML uses a *text file* kind of format. Even though this format forces a slower parsing and thus a slower access or query which represents a disadvantage, the XML format by itself allows a tree representation

of the data. It is then easy to organize data in a hierarchic kind of way. XML can also contain a lot of different kinds of data without having any sparse format, even if there are a lot of missing data, either because a lot of the arguments are non-mandatory or because the database is not complete. Since the old Excel files most certainly contain errors, this will ensure that the database will still be usable.

4. Choice of a native XML database

We decided to use a native XML database because of all the reasons diclosed in the previous chapter. Then, we determined that *BaseX* was the most suitable database management system for our scenario. This chapter will explain all the reasons why.

4.1 What is BaseX ?

BaseX is a native and light-weight XML database management system and XQuery processor. It is specialized in storing, querying and visualizing large XML documents. [16] BaseX is distributed under a permissive free software license. The included GUI enables users to interactively search, explore and analyze data, but also to evaluate XPath/XQuery expressions in realtime.

BaseX is written in Java and is NoSQL. More precisely, NoSQL or Not Only SQL is a class of database management systems that do not follow all the rules of a relational DBMS and that cannot use traditional SQL to query data. [17]

The main uses fo BaseX are visualizing, querying and storing large XML documents and JSON documents. Moreover, the management system is an in-memory database meaning that the data is stored in compressed from in Random-access memory (RAM) and not in the disk storage. Hence, it improves the performance since the data access is faster. However, it makes it difficult to modify the data. That is why memory databases like BaseX are mostly used to store and analyze data.

BaseX can be downloaded from its official website : http://basex.org/ [18].

4.2 Why BaseX ?

First of all, we decided to choose BaseX because it is a free platform and thus it will not add to the fees, letting what is left of the bonus for other updates on the infrastructure. The code is totally open source that we can find on GitHub [19].

BaseX is also a single-model database meaning that the database management system is organized around a sole data model, which is the XML format. It helps us to focus on handling a single format of documents and to avoid some confusions about how to organize, store, and manipulate our data.

Then, it offers a GUI (Graphical User Interface) that allows interactive searches, explorations, and studies of the data. It has a simple enough *Desktop GUI* for the secretary, before the university actually comes up with a full desktop app. Moreover, it supports XML Querying in the latest XQuery version.

Finally, it is easily usable meaning that it is easy to navigate in the database. BaseX is the perfect candidate for handling distributed XML data. Furthermore, it is cross platform, meaning that it can be used on all the different operating systems such as Windows, Linux or MacOS.

4.3 Small BaseX Tutorial

BaseX is not very difficult to use and is very well documented [20].

There exist two different ways to create a database. We can either use the command line *"CREATE DATABASE <name>"*or select *Database* → *New* in the menu bar and browse to a chosen XML document.

There is a drop-down menu right next to the input bar. 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 4.1 – Find queries' examples [21]

In the *XQuery mode*, the user can enter *XPath* and *XQuery* expressions in the input bar. 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 > 1 000 000 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 4.2 – XPath and XQuery expressions' examples [21]

In the *Command mode*, the user can enter BaseX Commands and they will be executed.

More details about BaseX GUI can be found on this website : http://docs.basex.org/wiki/Graphical_User_Interface [21].

5. Implementation

The first step for us is to convert the *Excel* spreadsheets into XML files. This step is easily completed with a couple of mouse clicks, giving us three files : *class.xml*, *student.xml* and *personnel.xml*. Each of those files containing several instances of the following data structures :

```
<personnel>
  <status></status>
  <name></name>
  <birthdate></birthdate>
  <bank_account></bank_account>
  <id></id>
</personnel>
```

FIGURE 5.1 – Personnel structure

```
<class>
  <name></name>
  <teacher></teacher>
  <mnemonic></mnemonic>
  <room>
      <name></name>
      <time></time>
      <week></week>
  </room>
</class>
```

FIGURE 5.2 – Class structure

```
<student>
  <study year=""></study>
  <name></name>
  <birthdate></birthdate>
  <id></id>
</student>
```

FIGURE 5.3 – Student structure

The next step is to create a new database and add the files to it. It is done easily enough in the BaseX desktop GUI by clicking on *New*. It is a good idea to merge the different files under a single data structure <*uni*>.

The final step will be for us to determine the integrity constraints related to this database so that the secretary can not make any mistakes when adding a new entry to the database. Those constraints (cf. *2.3 Constraints* can be implemented as follow (in the case of an insertion or update on the data) :

Before each new insertion or update we should check if the relevant functions return *true* so that we know that the data is correct and does not corrupt the integrity of the database. Here are our functions, for each of the constraints :

1.
```
declare function constraint:checkID($id)
  as xs:boolean
    {
      not(doc("project_final/final.xml")//id=$id
      or empty($id)
      or $id="")
    };
```

2.
```
declare function constraint:checkClassLocal($classname, $name, $week, $time)
as xs:boolean
    {
        not(doc("project_final/final.xml")//class[name!=$classname]
        /room[name=$name and week=$week and time=$time])
    };
```

3.
```
declare function constraint:checkClassTeacher($classname, $teacher,
$week, $time)
as xs:boolean
  {
    not(doc("project_final/final.xml")//class[name!=$classname
      and teacher=$teacher
      and room/week=$week
      and room/time=$time])
    and exists(doc("project_final/final.xml")//personnel[status="Teacher"
      and name=$teacher])
  };
```

4.
```
declare function constraint:checkStatus($status)
as xs:boolean
    {
        let $all_status := ("Teacher", "Administration",
          "Cleaning", "Restaurant")
        return($status = $all_status)
    };
```

5.
```
declare function constraint:checkStudy($study)
as xs:boolean
  {
    let $all_studies := ("Computer Science", "Cybersecurity",
      "Biology", "Philosophy", "Law")
    return($study = $all_studies)
  };
```

6.
```
declare function constraint:checkName($name)
as xs:boolean
  {
    not(empty($name)
    or $name!="")
  };
```

```
7. declare function constraint:checkMnemonic($mnemo)
   as xs:boolean
     {
       not(doc("project_final/final.xml")//mnemonic=$mnemo
       or empty($mnemo)
       or $mnemo="")
     };
```

To sum up those functions we can regroup them under bigger ones that will be called when adding an element in the database :

```
declare function constraint:addPersonnel($id, $status, $name,
$birthdate, $bank_account)
as xs:boolean
    {
        constraint:checkName($name)
        and constraint:checkID($id)
        and constraint:checkStatus($status)
    };

declare function constraint:addStudent($id, $study, $name, $birthdate)
as xs:boolean
  {
    constraint:checkName($name)
    and constraint:checkID($id)
    and constraint:checkStudy($study)
  };

declare function constraint:addClass($name, $teacher, $mnemonic,
$room_name, $time, $week as xs:int)
as xs:boolean
  {
    constraint:checkName($name)
    and constraint:checkMnemonic($mnemonic)
    and constraint:checkClassLocal($name, $room_name, $week, $time)
    and constraint:checkClassTeacher($name, $teacher, $week, $time)
  };
```

With that in mind, we can write a short exemple of adding a new element in the database :

```
declare variable $id := "nbaudoux";
declare variable $status := "Teacher";
```

```
declare variable $name := "Nicolas Baudoux";
declare variable $birthdate := "06 06 1966";
declare variable $bank_account := "BE61310145731457";

if (const:addPersonnel($id, $status, $name, $birthdate, $bank_account)) then (
  insert node
  <personnel>
    <id>{$id}</id>
    <status>{$status}</status>
    <name>{$name}</name>
    <birthdate>{$birthdate}</birthdate>
    <bank_account>{$bank_account}</bank_account>
  </personnel>
  into /uni)
  else ()
```

And a short exemple of an update, where we will call the relevant function directly. We also need to check if the old value exists in the database :

```
declare variable $mnemonic := "INFO F 405";
declare variable $old_mnemonic := "INFO F 406";

if (//mnemonic=$old_mnemonic
and const:checkMnemonic($mnemonic)) then (
  let $old_node := //class[mnemonic=$old_mnemonic]
  return (replace value of node $old_node/mnemonic with $mnemonic)
) else ()
```

And an exemple of a deletion in the database. Exactly as for the update, we need to verify that the value exists before deleting it :

```
declare variable $id := "ribauwen";

if (//student[id=$id]/study) then (
  delete node //student[id=$id]/study
) else ()
```

And finally, to search the database, the secretary can use the *FIND* input bar on the GUI.

FIGURE 5.4 – Find Bar

with keywords directly, or with queries like : *//student[name="Richard Bauwens"]* to find evert student

named *Richard Bauwens*, as explicated in the small tutorial earlier.

For more complicated queries it could be easier for her to write them down once, in the *BaseX* editor and save it as a *.xq* file for future usage.

6. Results

After creating our database and implementing all those functions, we have a database structured as followed :

```
<personnel>
  <status>Teacher</status>
  <name>Paul Dupond</name>
  <birthdate>17-12-1961</birthdate>
  <bank_account>BE61310126985517</bank_account>
  <id>pdupond</id>
</personnel>
```

FIGURE 6.1 – Example of the personnel structure

```
 <student>
   <study year="MA1">Cybersecurity</study>
   <name>Richard Bauwens</name>
   <birthdate>08-10-1996</birthdate>
   <id>ribauwen</id>
 </student>
```

FIGURE 6.2 – Example of the student structure

```
<class>
  <name>Introduction to Cryptography</name>
  <teacher>Olivier Markowitch</teacher>
  <mnemonic>INFO-F-405</mnemonic>
  <room>
      <name>Forum C</name>
      <time>We:8-10</time>
      <week>1</week>
      <week>2</week>
  </room>
  <room>
      <name>Forum D</name>
      <time>We:8-10</time>
      <week>3</week>
  </room>
</class>
```

FIGURE 6.3 – Example of the class structure

```xml
<uni>
  <student>
    <study year="MA1">Cybersecurity</study>
    <name>Richard Bauwens</name>
    <birthdate>08-10-1996</birthdate>
    <id>ribauwen</id>
  </student>
  <student>
    <study>Computer Science</study>
    <name>Marine Devers</name>
    <birthdate>06-09-1996</birthdate>
    <id>mdevers</id>
  </student>
  <class>
    <name>Introduction to Cryptography</name>
    <teacher>Olivier Markowitch</teacher>
    <mnemonic>INFO-F-405</mnemonic>
    <room>
        <name>Forum C</name>
        <time>We:8-10</time>
        <week>1</week>
        <week>2</week>
    </room>
    <room>
        <name>Forum D</name>
        <time>We:8-10</time>
        <week>3</week>
    </room>
  </class>
  <personnel>
    <status>Teacher</status>
    <name>Paul Dupond</name>
    <birthdate>17-12-1961</birthdate>
    <bank_account>BE61310126985517</bank_account>
    <id>pdupond</id>
  </personnel>
</uni>
```

FIGURE 6.4 – Overview of a filled database

7. Conclusion

To conclude, a small university needing to change its way of storing a huge amount of information was presented. Because the old files were from *Excel* and because the secretary still wanted an easy desktop GUI, an XML database with the management tool BaseX was the ideal choice. The two only difficulties were to find a practical data structure for the university and to learn the XQuery language used by BaseX. But once we overcame both of them, the implementation was pretty smooth.

Bibliography

[1] Technopedia, "Definition of database." https://www.techopedia.com/definition/1185/database-db. Accessed on 2018-10-20.

[2] Techtarget, "Definition of database." https://searchsqlserver.techtarget.com/definition/database. Accessed on 2018-10-20.

[3] Techtarget, "Definition of xml." https://searchmicroservices.techtarget.com/definition/XML-Extensible-Markup-Language. Accessed on 2018-10-20.

[4] Wikipedia, "Xml." https://en.wikipedia.org/wiki/XML. Accessed on 2018-11-24.

[5] Etutorials, "Xml." http://etutorials.org/XML/xml+data+management. Accessed on 2018-11-24.

[6] W3schools, "Xml example." https://www.w3schools.com/xml/xml_server.asp. Accessed on 2018-11-24.

[7] TechTarget, "Definition of relational database." https://searchdatamanagement.techtarget.com/definition/relational-database. Accessed on 2018-11-24.

[8] J. S. Blog, "Examples of relational databases." https://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/. Accessed on 2018-11-24.

[9] Technopedia, "Definition of non-relational databases." https://www.techopedia.com/definition/25218/non-relational-database. Accessed on 2018-11-26.

[10] Hackermoon, "Comparison between relational and non-relational databases." https://hackernoon.com/relational-versus-non-relational-database-d5d1c439fb86. Accessed on 2018-11-26.

[11] Technopedia, "Definition of xml database." https://www.techopedia.com/definition/30558/xml-database. Accessed on 2018-11-26.

[12] TutorialsPoint, "Example of xml database." https://www.tutorialspoint.com/xml/xml_databases.htm. Accessed on 2018-11-26.

[13] SlideShare, "Types of xml databases." https://fr.slideshare.net/hawlery1989/xml-databases. Accessed on 2018-11-30.

[14] XML4Pharma, "Native xml database." http://www.xml4pharma.com/XMLDB/. Accessed on 2018-11-30.

[15] XML and Databases, "Native xml database." http://cecas.clemson.edu/~juan/CPSC862/XML-Databases.htm#datainnative. Accessed on 2018-11-30.

[16] Wikipedia, "Basex." https://en.wikipedia.org/wiki/BaseX. Accessed on 2018-10-22.

[17] Techopedia, "Nosql." https://www.techopedia.com/definition/27689/nosql-database. Accessed on 2018-10-22.

[18] BaseX, "Download basex." http://basex.org. Accessed on 2018-12-05.

[19] GitHub, "Github basex." https://github.com/BaseXdb. Accessed on 2018-12-05.

[20] BaseX, "Documentation on basex." http://docs.basex.org/wiki/Main_Page. Accessed on 2018-12-05.

[21] BaseX, "Documentation on basex graphical user interface." http://docs.basex.org/wiki/Graphical_User_Interface. Accessed on 2018-12-05.