Benjamin THEVENEAUX
DE – S22

Library – data pipeline assignment

Introduction

In this report, I describe the 7 main nodes (the root one and the 6 direct child nodes).
Each node contains at most one attribute, which is used either as a key (e.g. idBook)
or as a reference to another key.
If they are references, the “Ref” suffix is added in the name (e.g. idBookRef).
In the schemas, these attributes are in italic and bolded (except for LIBRARY: in italic).
The type of each node is also in italic (the types of my namespace are prefixed with **lib:**).

Then I comment and summarize the choices and the improvement points I discovered.

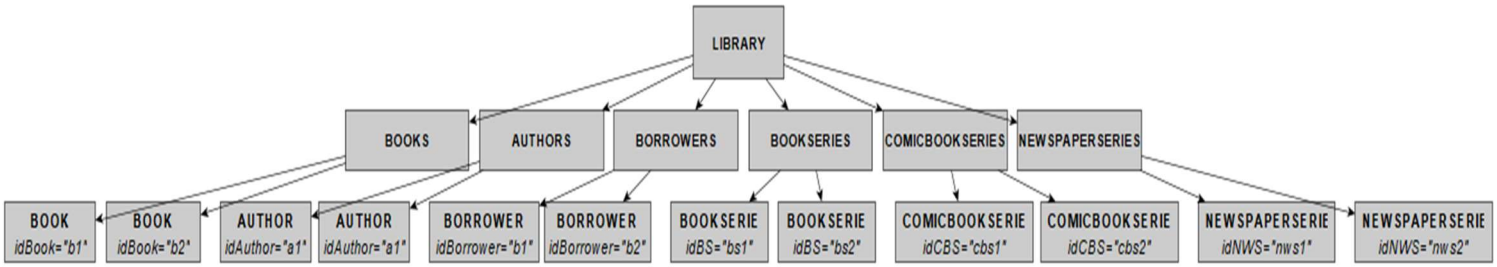
Then comes the list of the transformations I worked on.

At the end of this report, I give the tools I worked with.

The files attached to the zip file are
- *library.xsd* (the schema) and *library.xml*
- The xsl transformation files: *library_transfo[1to5].xsl*
- The output files: *output[1to3].html*, *output4.xml*, *output5.json*
- The json schema applied to output5.json: *library_comics_output5.schema.json*

1. The structure

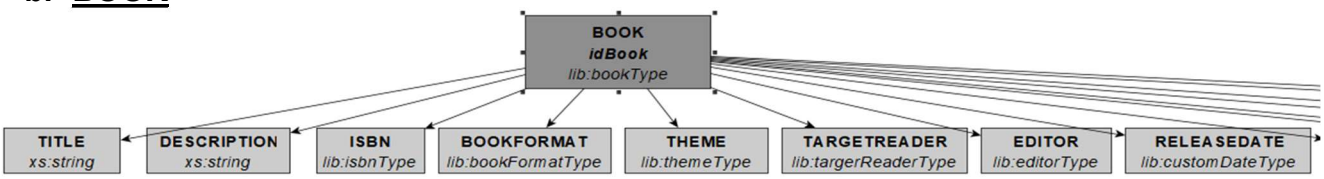
a. LIBRARY (type: libraryType)

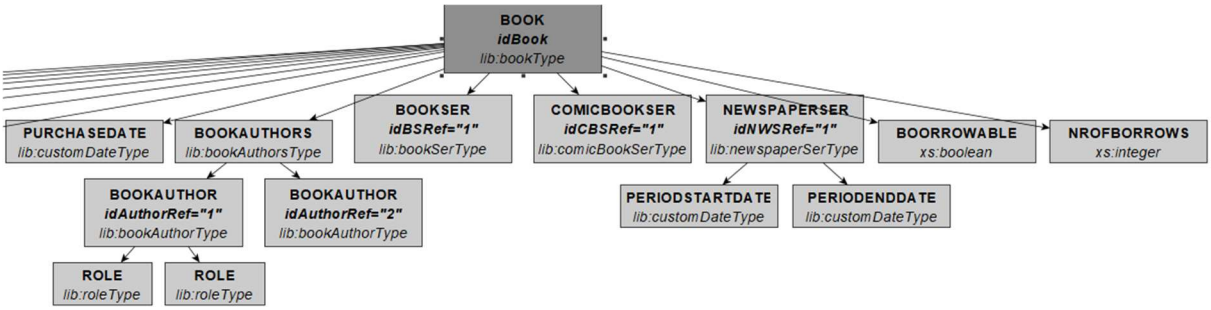


For a sake of readiness (mainly for the ability to expand and reduce with the usual text editors and web browsers, by clicking on the left of the node), I chose, under the root node LIBRARY, to have 6 nodes as lists of elements. They are described in the next sections.
E.g. BOOKS contains BOOK elements.

It also contains the keys and key references used, as an example:
A key on BOOK/@idBook with a referring key on BORROWING/@idBookRef refers to it.

b. BOOK



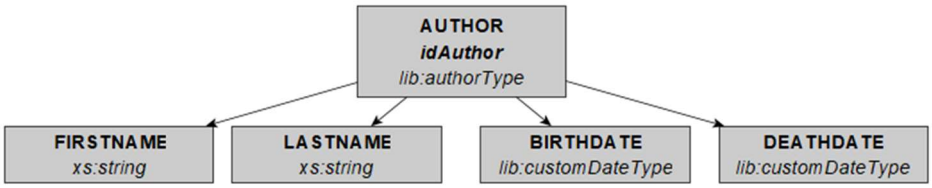


This node is the biggest one of the schema. It contains all information about an article available in the library, directly or indirectly (using references to other nodes, with the keys and key references).

Each direct “child” node of BOOK appears once, except for

- min: 0: DESCRIPTION, ISBN (no ISBN for a newspaper), BOOKAUTHORS (min: 0)
- BOOKSER / COMICBOOKSER/ NEWSPAPERSER (min: 0). These three nodes are under a “xs:choice” tag. Indeed, a book can have one format only, hence at most at most one “serie” and one “serie” type linked to it.

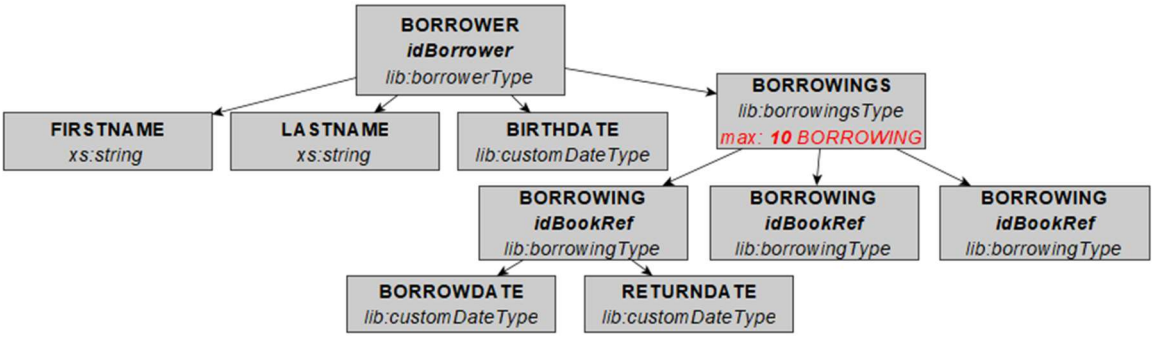
c. AUTHOR



It contains usual info about an author.

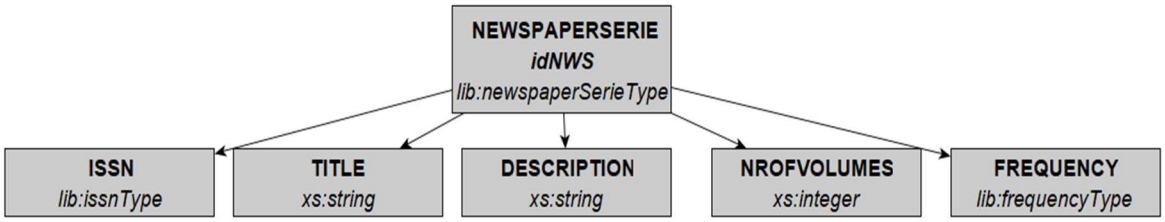
NB: there is no link to the BOOK. The link is made with BOOK/BOOKAUTHORS/BOOKAUTHOR/@idBookRef = BOOK/@idBook .
Indeed, the roles of an author can vary between the books (in my xml file, it is obvious in the “Comic” book type. See also the 2nd transformation).

d. BORROWER



The BORROWINGS node has its content limited, it is the way I limit the number of books (BORROWING) borrowed by each user.
The link with the BOOK is made using @idBookRef = BOOK/@idBook.

e. SERIE elements



(I didnt paste BOOKSERIE and COMICBOOKSERIE, see explanation below)

These elements are used under BOOK, which refers to them using, e.g. for a "Comic", COMICBOOKSER/@idCBSRef referring to COMICBOOKSERIE/@idCBS.

The NEWSPAPERSERIE contains 2 more elements than BOOKSERIE and COMICBOOKSERIE:

- ISSN: the unique identifier of a newspaper (equivalent of ISBN for a book)
- The FREQUENCY (frequencyType) of the newspaper ("Daily", "Fortnightly", etc).

f. Custom types created

I created complexType for the main elements (BOOK, AUTHOR etc).
I also created simpleType for restricting data

- Dates: customDateType in YYYY-MM-DD format based on string, this format is "standard" and the sorting works well with this one.
- ISBN / ISSN: regular expressions.
- BOOKFORMAT, TARGETREADER, THEME, EDITOR : "enumeration" values.

2. Comments about the choices made on the schema

I chose to make one BOOK element for the three formats I identified ("Book", "Comic" and Newspaper). Indeed, I wanted to avoid repeating information (title, description, etc).

At the end of this project, I acknowledge one main disadvantage to this structural choice.
The "Book" and the "Comic" have a lot in common (ISBN for example), unlike a newspaper, who has

- No ISBN, but an "ISSN", at the "serie" level
- Frequency and period start/end date (in the schema as child nodes of the NEWSPAPERSER element).

I suppose I was too "driven" with the three book formats I identified. I could have created (it is too late now!)

- a book / bookserie pair for the "Book" and "Comic" formats, with ISBN
- a newspaper / newspaperserie pair for the "Newspaper" format, with ISSN, frequency…

Another advantage of that : on the BORROWER side I would have two "borrowing" types, with two maximum number of occurrences, so that the user could have different limits for book/comic and newspaper (I experienced that system in two different libraries).

About the "borrow" information, I didn't create a dedicated (and independent) element under library. I found it more convenient to add it under the "BORROWER" node. Otherwise I would have created two key/keyref pairs:

- one between the borrow info and the book
- one between the borrow info and the borrower.

As another enhancement point: I didn't add any "reserved" node: when a book is already borrowed, usually there can be a queue of users which "reserve" the book. As soon as the article comes back to the library, the user on the top of the queue can get it.

3. XSL transformations

a. 1st transformation: list of books borrowed per borrower.

It lists, for each borrower, the articles borrowed (title, borrowdate and return date). The books are sorted by return dates.
The borrowers are sorted by lastname.
The number of books borrower and the number of books the user can still borrow are also displayed.

The first template is for BORROWER. Inside this template, the BORROWING is called, the BOOK is called (for TITLE retrieval), using a filter (idBook = idBookRef of the BORROWING).

b. 2nd transformation: List of the books per author, with the role(s).

Not available for the newspapers (there is no author linked to the newspapers in my schema).

It interesting to know which author worked in which book, with the roles. Especially for the comic books. An author can be scriptwriter on one, illustrator on another one, both of them on another one…

The authors are sorted by lastname, then the books are sorted by release date.
If the book / comic is part of a serie, hence the "serie" title is also added.

The output is adapted to the fact a book is a "one-shot" or not, and to the fact an author has several roles (displayed as a list) or not (at the same line as the book title).

The templates called are, in this order:
AUTHORS (for sorting) -> AUTHOR -> BOOK (BOOKAUTHOR/@idAuthor = idAuthoRef of the BOOKAUTHOR) -> BOOKSERIE / COMICBOOKSERIE (using BOOK/BOOKSER/@idBSRef = BOOKSERIE:@idBS).

c. 3rd transformation: articles sorted by popularity

Under the BOOK node, I added NROFBORROWS: a random number between 0 an 100, for the number of times the books has been borrowed.
We can then get, for each "Theme" (Adventure, History, etc…) the most popular books. It can help the librarian, for example to highlight the less popular one, or to discard them from borrowing if nobody borrows them.

The templates called are THEME (with a filter to avoid duplicates), then BOOK (sorted by NROFBORROWS descending).

d. 4rd transformation (XML file): Books by authors (Writer / Scriptwriter / Novelist).

I chose to discard the Illustrator and Coulourist, since they are not independent form the others (an illustrator always work with a Scriptwriter / Writer).
I wanted to explore another way of sorting the information. Indeed, in the library I'm used to borrow my books, the librarians recently changed the way they sort the comic books. Previously the articles were sorted by author… I liked that way of sorting, since if I like the work of some author, I want to check the other books of the same author.
Now the comics are sorted by serie. Working on that assignment made me aware of the purpose of this rearrangement: actually, for some long-term serie, the scriptwriter(s) can change along time. E.g. for Largo Winch, Blake and Mortimer, Astérix…

e. 5th transformation (json): basic "Comic" information sorted by serie

For each **comicbook** serie, we get the information about the serie title, description, number of books, title of each book (sorted by release date) then author of each book.

It can be useful, as an JSON API for a website like www.bedetheque.com which lists the comics.

NB: I also created the JSON schema related to the output file (*library_output5.schema.json*).

4. Tools used

Sources of inspiration and data:
I used

- https://www.bedetheque.com/ : this is a referential for comic books. It contains various information about ISBN, authors, etc… I got the idea of having a "serie" element even with "one-shot" comic books from this website

- https://www.mediathequegeorgeswolinski.fr/ : my city's library. Here I got the idea about the structure of the newspaper. With their "search" tool, we choose a period, unlike for the books.
- Wikipedia for the information about authors, ISBN, titles.
- https://portal.issn.org/ for the ISSN information (newspaper identifier).

Development tools.

- Microsoft Visual Studio: for the creation of the XSD and XML files
- https://www.liquid-technologies.com/online-xsd-to-xml-converter: while creating the XSD file, I used this website to generate dummy XML files (it creates a short XML files with dummy values from the XSD file):

XML validation against XSD:

- Notepad++ "XML tools" plugin
- Freeformater website: https://www.freeformatter.com/xml-validator-xsd.html

XML file creation (from real data).

I stored the data as several CSV files, then converted it to XML blocks using https://www.convertcsv.com/csv-to-xml.htm

XSL file creation:

- Notepad++ "XML tools" plugin
- https://www.freeformatter.com/xsl-transformer.html (I used it once when I had issues with the charset/bad parsing about characters "é" and "è". At last, I decided to "play" manually with the encoding from Notepad++

Graph editor: I used yEd (https://www.yworks.com/products/yed)

JSON Schema validator: https://www.jsonschemavalidator.net/