Report Project 3 : XQuery

Rémy Detobel & Nathan Liccardo

May 23, 2018

1 Introduction

This document aims to detail each choices and hypothesis we made during our implementation. As a reminder, we were assigned to implement three different XQuery programs. Each of those query use a part of the DBLP database (DBLP-excerpt). The structure of this report will be divided into three parts (one per program).

2 First XQuery program

For the first program, we were assigned (for each author) to return the number of co-authors and the number of joint publications with each of them. To realise this, we start by iterate on each author as below :

- ```
  for $author in //author
  return <author>
      ....
  </author>
  ```

Which will create an `<author> ... </author>` bloc for each author. Inside each of these blocs, we will find following informations (in sequence).

2.1 Author name and Co-authors informations

The author name and the number of co-authors are respectively obtained using `data` and `count` functions (defined by XQuery). Below instructions are formal definitions of `name` and `coauthors` tags :

- `<name>{data($author)}</name>`
- `<coauthors number="{count(distinct-values(//*[author=$author]/author))-1}"> ... </coauthors>`

2.1.1 Co-authors informations

For each co-author, we must obtain his name and the number of joint publications. Both informations will be contained inside `<coauthor> ... </coauthor>` tags (itself contained in the `coauthors` bloc). Each of the `coauthor` blocs are created by iterating on co-authors as below :

- ```
  for $coauthor in distinct-values(//*[author=$author]/author[not(.=$author)])
  return <coauthor>
  ....
  </coauthor>
  ```

Co-author name As for the author name, co-author name is reached using the `data` function on the `coauthor` variable. This is realised as follow :

- `<name>{data($coauthor)}</name>`

Joint publications For each co-author, we must retrieve the number of joint publications. Once again, this is achieved using the `count` function available in XQuery. Here is the final instruction :

- `<nb_joint_pubs>{count(//*[author=$author]/author[.=$coauthor])}</nb_joint_pubs>`

3 Second XQuery program

For the second request, we were assigned to give (for each proceedings) its title and titles of articles appearing in it. Each `proceeding` blocs are created by the use of an iteration :

- ```
  for $proceeding in //proceedings
      return <proceedings>
      ....
      </proceedings>
  ```

For each of these blocs, we must retrieve the title (`proc_title`) and a list of articles (`title`). This is achieved using the value of the `key` attribute (inside `proceedings` tags) and the value contained inside `crossref` tags. Regarding to the title, we obtain the below instruction :

- `<proc_title>{data($proceeding/title)}</proc_title>`

Finally, we can create the list of titles using the `key` value :

- ```
  for $inproceeding in //inproceedings[crossref=data($proceeding/@key)]
  return <title>{data($inproceeding/title)}</title>
  ```

4 Third XQuery program

For the last program, we were assigned to compute (for each pair of authors x and y, $x \neq y$) the distance between x and y. First, we will explain how functions has been used to implement complex operations. After that, we will describes how we have built the final `<distance>` bloc.

4.1 First function : `getListAuthorDistance`

The most important function that we have implemented is `getListAuthorDistance`. It allows to return a list of co-authors that are at a distance comprises in $[1, d]$ from the current author. The function starts by checking if the distance given in parameter is greater then zero. If it is the case, then the function will be called iteratively to get co-authors at a distance d, $d-1$, ... 2, 1. In the other case, the function return the author without any other values. Formally, this loop has been written as below :

- ```
  for $newCoAuteur in distinct-values($allContext[author=$intermediateAuteur]/author)
  where not($intermediateAuteur = $newCoAuteur)
    return ($newCoAuteur,
        custom:getListAuthorDistance($allContext, ($distance - 1), $newCoAuteur))
  ```

4.2 Other functions

To be able to get every distances for every authors, we had to implement two other functions. First, we have implemented `authorInASpecificDistance`. It allows to get every authors that are at a specific distance from the current author. Formally, it returns every authors that can be reached using a distance d (no more, no less). The function returns XML data information and has been written as below :

- ```
  let $previousCheck := distinct-values(
                  custom:getListAuthorDistance($allContext, ($distance - 1), $author))
  let $currentCheck := distinct-values(
                  custom:getListAuthorDistance($allContext, $distance, $author))
  for $coAuteur in $currentCheck[not(.=$previousCheck)]
  ```

Finally, `loopDistance` has been implemented to check if the result set of `authorInASpecificDistance` is empty or not. If this set is empty, it means that every distances has been reached and that we can stop the iteration.

4.3 Final instruction

The last operation was to create the `<distances> ... </distances>` bloc. This bloc will contains a set of `<distance>` tags that are created using the `authorInASpecificDistance` function. After that, these tags are added to the final set using the `loopDistance` function. Finally, let's remark that for each author of the DBLP database, we will have a call to the recursive function. The final bloc is obtained as below :

- ```
  <distances>
  {
      for $author in distinct-values(//author)
          return custom:loopDistance($author, //*, 1)
  }
  </distances>
  ```
  ```