Assignment 3: XQuery

BAKKALI TAHERI Abdeslam , BOUGLAM Sara

April 2021

1 Introduction

XQuery is a query and functional programming language that performs the transformation of structured as well as unstructured data, generally in XML format, In the following sections XQuery programs will be implemented that satisfy some indicated queries, detailed description of the implementations as well excerpts from the written program will be provided along.

2 Query 1

For the implementation of the first query, initially the part of the XML section that is the centre of interest is the section containing the current author, hence an excerpt has been extracted from the given `XML` document:

```
    declare function local:sample($author, $dblp){
    for $author_ in $dblp//author where  $author_=$author
        return   $author_//parent::*
};
```

Later, the coauthors have been extracted, as follow :

```
declare function local:extractAuthor($sample, $author) {
    for $authors in $sample
        for $author1 in  $authors//author where  $author1 != $author
         return $author1
};
```

the condition `$author1 != $author` is to ensure that an author is not coauthor with himself. Finally on order to extract the number of proceedings with each coauthor, the following has function has been implemented :

```
    declare function local:jointPub($samples , $author) {
        for $sample in $samples
            for $author1 in $sample//author where $author1=$author
            return $author1/parent::*
};
This function is applied to each author's coauthor .
```

Finally everything is put together:

```
for $author in distinct-values( $dblp//author)
         let $sample := local:sample($author, $dblp)
         let $coauthors :=distinct-values(local:extractAuthor($sample, $author))
         let $c:= count($coauthors)
         return
         <author>
         <name>{$author}</name>
         <coauthors number="{$c}">{
            for $coauthor in $coauthors
             let $joinPubNumber :=count( local:jointPub($sample, $coauthor))
                return <coauthor>
                <name> {$coauthor}</name> <nb_joint_pubs>
                {$joinPubNumber}
                </nb_joint_pubs>
                </coauthor>
         }</coauthors>
       </author
```

In the excerpt above,initially a loop through all authors have been performed, to later on, count the the number of coauthors and loop through them to finally extract the number of the joint publications.

3 Query 2

To extract the articles for each proceeding, the following has been implemented:

```
let $dblp := fn:doc("dblp-excerpt.xml")
for $proc in $dblp//proceedings
return <proceedings>
{
<proc_title>{$proc/title}</proc_title>,
(: Articles list :)
let $articles := ( $dblp//inproceedings[crossref=$proc/@key])
for $article in $articles
return
  <title>{$article/title}</title>
}
</proceedings>
```

Note that articles for each proceedings is accessed through the `crossref`

4 Query 3

For the last query, the first step is to generate a graph showing the relation between the author and the coauthors for each element of dblp. This is done

by the `setUp()` function, the obtained graph has the following structure :

```
    <authors>
  <author>
    <name>Mazeyar E. Makoui</name>
    <coauthors/>
    <coauthors/>
  </author>
  <author>
    <name>Gunter Saake</name>
    <coauthors>
      <author>Kai-Uwe Sattler</author>
      <author>Andreas Heuer</author>
    </coauthors>
    <coauthors/>
  </author>
  ......
  <authors>
```

The graph has been produced as follow, where `coauthor` and `sample` function are the same as `Query 1`:

```
declare function local:setUp(){
let $dblp:=fn:doc("dblp-excerpt.xml")//dblp
 return<authors> {
   for $author in distinct-values( $dblp//author)
      return <author>
       <name> {$author}</name>
       {
          for $sample in  local:sample($author,$dblp)
          return <coauthors>
             {  for $coauthor in local:coauthors($sample,$author)
                return $coauthor
              }
              </coauthors>
            }
       </author>
}
</authors>};
```

The distance-one() function makes it possible to convert the structure previously created into a first part of the solution but for those whose distance is equal to 1.

```
    declare function local:distance_one($graph){

  let $authors:= $graph/author
  for $author in $authors
```

```
    for $coauthor in $author/coauthors/author
      where $coauthor != ''
      return <distance> <author1> {data($author/name)} </author1> <author2>
      {data($coauthor)} </author2> <dist>1</dist> </distance>
};
```

Then, distance2() is a recursive function that generates the solution using two methods:

```
declare function local:distance2($dblp, $graph_dist, $cont){
  if ($cont = 0)
    then $graph_dist
  else
    let $next_dist := local:distance($dblp, $graph_dist)
    let $test2 := trace(count($next_dist),"count -> next_dist: ")
    let $test2 := trace(local:height($next_dist),"height:")
    let $current_cont := local:height($next_dist)
    let $sol := ($graph_dist,$next_dist)
(: return <cont>{$current_cont}{$sol}</cont> :)
    return local:distance2($dblp, $sol, $current_cont)
};
```

- distance(): this function will browse the authors A, B, and C. and if C is the coauthor of A and B, then this will create a new entry having a distance equal to dist (A, C) + dist (C , B)

```
declare function local:distance($dblp, $graph_dist){
let $test := trace("distance","txt")
for $author_mid in distinct-values( $dblp//author)
    for $dist_author2 in $graph_dist
      for $dist_author1 in $graph_dist
      where $author_mid = data($dist_author1/author2) and $author_mid =
      data($dist_author2/author1) and data($dist_author1/author1) !=
      data($dist_author2/author2) and
      (local:couple_already_exist($dist_author1/author1, $dist_author2/author2,
      $graph_dist) eq false())
        let $var1 := data($dist_author1/dist)
       let $var2 := data($dist_author2/dist)
       let $sum := $var1+$var2
       return <distance> {$dist_author1/author1} {$dist_author2/author2}
       <dist> {$sum} </dist> </distance>

};
```

- height(): Checks that the result of the distance() function is not empty.

```
declare function local:height($graph_dist){
```

```
  let $result:=
    if (fn:empty($graph_dist/*))
    then 0
    else 1
  return $result
};
```

Finally distance2() concatenates the previous solutions and the one generated by distance, when distance() provides an empty structure, then the algorithm stops.

Note: The proposed program for the query may take up to 20 min to finish execution