TEI XQuery Exercises

James Cummings

We are now going to use the eXist Basic XQuery Interface web-form to query the XML database using XQuery. You may wish to try this very basic XQuery:

```
(: All Person Elements :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $stones in collection('/db/pc')//tei:TEI
let $people := $stones//tei:person
return $people
```

This should return a copy of each of the `<person>` elements in the Protestant Cemetery collection. Remember that you can also create your own output elements with XQueries as well. These can multiple variables. For example:

```
(: All Names and Nationalities :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
let $stones := collection('/db/pc')//tei:TEI
for $person in $stones//tei:person
let $name := $person/tei:persName
let $nationality := $person/tei:nationality
return
<result>
  {$name}
  {$nationality}
  </result>
```

This should return a `<result>` element for every person in the database containing their `<persName>` and `<nationality>` elements. Each of these hits needs to return well-formed XML, hence wrapping it in the `<result>` element.

Do as many of the exercises below as you can in the allowed time. Provide the answers to record your progress if you wish.

1. For each of the `<person>` elements where the person is a woman, return their `<persName>` and `<nationality>` elements.

   ANSWER: The first surname is ______________

2. For each British woman, list their `<persName>` and the `<country>` they were born in if known.

   ANSWER: The first British woman in the database not to have a country of birth recorded is ______________

3. Now do the same, but also listing their country of death if known.

   ANSWER: Almost all of them died in ______________

4. But this causes a problem, how can you tell the birth country and the death country apart (without giving more of the original context)? One solution is to create new `<birth>` and `<death>`elements in the output. Try that first.

   ANSWER: The first British woman in the database to be born in Portugal and die in Italy is ______________

5. That is fine if the output is going to be further processed but what if I want to output an XHTML unordered list where one such hit might be:

   ```
   <ul>
     <li> Name: SarahBarnard </li>
     <li> Birth Country: Portugal </li>
     <li> Death Country: Italy </li>
     </ul>
   ```

   In that case we need to make it clear that we want the data inside the variable. Using the `data()` function is one way to do this as it takes a sequence of items and returns a sequence of atomic values. For example in the output one could put `{data($birthCountry)}` to return the text of the `$birthCountry` variable. Try now to produce the above XHTML unordered list for each British woman.

   ANSWER: The first woman to be born in England and die in Italy is ______________

6. But if you've done something like:

   ```
   <li> Name: {data($name)}</ li>
   ```

   then you have a bit of a problem. The names are not displayed very attractively because of their whitespace. You can further expand on variables in the output by doing something like `{data($name/surname)}` Create the same output as in the last question, but have the name output as: surname, forename.

   ANSWER: The third woman's surname is ______________

7. Of course, you can also do this by creating separate variables for the `<surname>` and `<forename>` before you output the result. Try that now.

   ANSWER: The fourth woman's surname is ______________

8. But you might not want the results in the order they are in the database. If you add `order by $surname` before your `return` then your results should be alphabetically ordered by the contents of that variable. Try it now.

   ANSWER: The fifth woman's surname is ______________

9. Now, instead order them by country of birth. What does it do with those that don't have a recorded country of birth?

   ANSWER: The third woman's surname is ______________

10. You can also restrict the output by using a `where` statement like `where $deathCountry = 'Italy'` just before your ordering. This will only order

and return those records whose $deathCountry variable is equal to 'Italy'. So all blanks and other countries will disappear. Try this now, still ordering by country of birth.

ANSWER: The fourth woman's surname is ____________

11. Now do the same thing but find all those whose $deathCountry is *not* Italy. (Hint: use != for not equal.) Why are none of the blank entries returned?

    ANSWER: The first woman's country of death is ____________

12. For each British woman (not just those who died anywhere specific), output their records like:

```
        <ul >
          <li> Name: Barnard, Sarah </li>
          <li> Stone ID: S66</li>
          <li> Record Title: Stone 66</li>
          <li> Birth Country: Portugal </li>
          <li> Death Country: Italy </li>
          </ul>
```

    You may wish to use an unabbreviated XPath axis like `ancestor::` to find the `tei:TEI/@xml:id` (Stone ID). Order your output by surname.

    ANSWER: The fifth woman's surname is ____________

13. Build on the previous XQuery to add list items for year of birth and year of death. You should use the `substring-before()` function to retrieve just the year from the `value` attribute of `<birth>` and `<death>`. Order your output by birth date.

    ANSWER: The first person for which all of birth country, birth date, death country and death date are recorded is ____________

14. The problem is, if we weren't restricting this to British women (which helps sometimes with speed and memory), we would notice that sometimes multiple people are listed on a single gravestone. So we might want to produce a listing for each stone that provides information for each person on it. The output we want to produce is a nested XHTML unordered list such as:

```
        <ul>
         <li> Stone ID: S6 </li>
         <li> Record Title: Stone 6 </li>
         <li> Person:<ul>
          <li> Name: van Buren, Elizabeth </li>
          <li> Birth Country: England </li>
          <li> Birth Date: 1881 </li>
          <li> Death Country: Italy </li>
          <li> Death Date: 1961 </li>
          </ul>
         </li>
         <li> Person: <ul>
         <li> Name: van Buren, Albert William </li>
         <li> Birth Country: USA </li>
```

```
 <li> Birth Date: 1878 </li>
 <li> Death Country: Italy </li>
 <li> Death Date: 1968 </li>
  </ul>
 </li>
</ul>
```

One of the ways to do this is to nest a `for` statement. In order to find people who share stones, we should stop restricting so much by gender and nationality. Where our previous queries might have started:

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
let $stones := collection('/db/pc')//tei:TEI
```

Since we are going to be removing the restrictions of gender and nationality we might want to lower the total number of stones we are dealing with. One way to do this is:

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $stones in collection('/db/pc')//tei:TEI[@n<11]
```

This will return only the first 10 gravestones. We then want to output the gravestone's ID number and its title. As we want this for each stone only, we should do this before nesting the `for` loop to deal with each person. Overall, something like:

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $stones in collection('/db/pc')//tei:TEI[@n<11]
let $stoneID := $stones/@xml:id
let $recordTitle := $stones//tei:titleStmt/tei:title
return
<ul>
  <li>Stone ID: {data($stoneID)}</li>
  <li>Record Title: {data($recordTitle)}</li>
```

might be a good way to start this XQuery. Notice that we haven't closed the `<ul>` element yet; we will have to remember to do so at the end!

Now we want to nest the `for` loop to print out the same information as before. Remove any ordering of the results since we want the data in the order it appears in the database. We nest the `for` loop, which will produce a `<li>` element for each person, using the same curly brackets as we use to escape any XQuery inside output. So it should start something like:

```
{
          for $person in $stones//tei:person
```

Then get all the same information you got before (forename, surname, birth country, birth date, death country, death date) and output it as an `<li>` itself containing a `<ul>` as in the example of the output above. Make sure you provide the ending curly bracket and closing `<ul>` tag — your output must be well-formed XML. So your query should end something like:

```
          <li>Death Date: {$deathDate}</li>
        </ul>
      </li>
      }
    </ul>
```

ANSWER: The surname of the first family with multiple occupants on a gravestone is ______________

15. To make an actual web page out of the previous query, we can nest this entirely inside an html structure. For example, it would need to start something like:

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
<html>
  <head><title>Getting Stones</title></head>
  <body>
    {
    for $stones in collection('/db/pc')//tei:TEI[@n<11]
```

But don't forget to close the HTML `<body>` and `<html>` elements at the end.

ANSWER: To make this proper XHTML I would just have to ______________

There are many different ways you can submit an XQuery to eXist to be processed. these can come from within programs, as the action from a submitted web-form, or directly from the filesystem. In most cases users will never know that they are submitting an XQuery, they've just clicked on a link or submitted a form. The reason we use the eXist's basic XQuery interface web-form, is so that you can see the XML results of your queries.

If you don't finish all the exercises you can try them later at your leisure.