Coursework C3: XQuery

For this piece of coursework you are asked to express natural language queries over documents containing movies in XQuery. On the course page you find the DTD for the `movies.xml` documents and one sample document. Of course, your queries have to work correctly over all possible documents satisfying the DTD, not only over the one sample. Note that DTD and sample document are the same as the ones for the coursework on XSLT.
You can develop and test the queries using the Kernow front end to Saxon. Note that to run Kernow und Linux, you need the official Java JDK from the Java website at Oracle.
After having run 100 queries, Kernow will ask you for a code. If you send me an email (`werner dot nutt at unibz dot it`), I will let you know it. Of course, you should not wait until the system requests the code from you, but send me an email when you start to work with Kernow.
Hint: Start your code always with the initial line

```
declare option saxon:output "indent=yes";
```

This tells the XQuery processor Saxon to indent the resulting XML code.

Movie Queries

1. Return an element "`movie_titles`" that contains a list of alphabetically sorted titles of all movies in the document.

2. Modify the code so that each movie title has an attribute "`number`", indicating the position of the title on the list.

3. Return an element "`movies`" with a list of `movie` elements, containing in turn attributes title and year, ordered by year in descending order. Write two versions, one with static and one with dynamic attribute and element constructors.

4. Restructure the movies document so that

   - movies appear according to their year, with the most recent years first, and, within the same year, according to their title
   - countries appear in alphabetical order
   - for each director, the first name is given before the last name
   - actors of a movie appear according to their last name and, for actors with the same last name, according to their first name.

   Everything else should remain as before.

5. Restructure the document so that movies are ordered according to their genre, that is:

   - Below the node "`movies`", there should be several genre elements one for each of the genres occurring in the document, with an attribute "`name`" indicating the name of the genre.
   - Below each genre element, there should be the movies of that genre. Each movie should be structured as for the previous problem, except that there should not be a genre element in the movie.

6. Output movies (title and year) and within each movie those actors whose role occurs in the summary of the movie.

7. Return an element "`actors`" that contains for each actor occurring in the input document one element "`actor`" with the attributes "`first_name`", "`last_name`", and "`yob`" (= year of birth) and an element "`film`" with attributes "`title`" and "`year`" and child element "`role`", where the films are ordered according to `year`.

   Note that for each actor in the input document, there should be exactly one actor in the output. Make sure your code works also if you have two actors with the same last name, say "Fred Smith" and "John Smith", that is, "Fred Smith" should appear before "John Smith".

8. Rewrite the code produced under 7. by introducing the following two functions:

```
declare function local:create-actor(
                                $ln as xs:string,
                                $fn as xs:string,
                                $ms as element()*)
  as element(),
```

which takes as input two strings, the last name and the first name of the actor, and a list of movies, returning the actor element for the actor with that name;

```
declare function local:create-film-with-role(
                                $m as element(),
                                $ln as xs:string,
                                $fn as xs:string)
  as element(),
```

which takes a movie and two strings, the last name and the first name of the actor, returning a film element and with the role played by the actor.

9. Modify the code so that every actor has an attribute “id” and the values of id are integers running from 1 to the total number of actors.

   Hint: Introduce a function

   ```
   declare function local:sorted-actor-names(
                                   $as as element()*)
     as element()*
   ```

   that takes a sequence of <actor> elements as input and outputs a sorted list of elements of the form

   ```
   <actor_name last_name="..." first_name="..."/>,
   ```

   with each actor_name occurring exactly once.

   Use this sequence to produce a numbered sequence with the functions you have crated before.

10. Write XQuery code that “disentangles” movies, directors and actors in the folowing way:

    - The result document has a top element cinema with two child elements, artists and movies. Below artists, there are artist elements and below movies there are movie elements.
    - An artist has an id attribute, with a unique id, and contains elements last_name, first_name, and birth_year.
    - A movie element has the same kind of children as before, except for two changes
      - a director element has an attribute director_id, with the id of the artist who directed the film, and no child elements,

  - an actor element has an attribute `actor_id`, with the id of the artist who acted in the film, and a child element `role` as before.

The XQuery code should take as input a document satisfying the movies DTD and output a document satisfying the description above.

Deliverable

Your deliverable will consist of

- a `.txt` file that contains for each query your formalization in XQuery.

The file `movie-queries.txt` on the website contains a numbered list of all queries. Write your answers into that file, each answer below the corresponding question.
Please, submit your work by email to `werner dot nutt AT unibz dot it` no later than

Mon, 20 January 2013, 23:30 hours.