Suggested solutions XQ1 - XQ16 (lesson 2)

EXERCISES IN XQUERY

The following exercises shall be solved with XQuery based on the movie database (the file movies.xml). The database is described in chapter 10 and is available in iLearn. The exercises can be interpreted in different ways, so, try different solutions with different result structures.

EXERCISE XQ1

Retrieve all the directors!

```
element Directors {
for $dn in distinct-values(//Director/@Name)
return element Director {$dn}
}
```

If we want more information, but still no duplicates:

```
element Directors {
for $dn in distinct-values(//Director/@Name)
let $d := (//Director[@Name = $dn])[1]
return $d
}
```

We could compact this with an XPath expression using the right axis:

```
element Directors {
//Director[not(@Name = following::Director/@Name)]
}
```

EXERCISE XQ2

Retrieve all the actors from USA that are part of the movie Driven!

```
element Result {
//Actor[@Country='USA'][../@Title='Driven']
}
```

or

```
element Result {
//Movie[@Title='Driven']/Actor[@Country='USA']
}
```

EXERCISE XQ3

Show the number of actors per movie!

```
element Result {
for $m in //Movie
return element Movie {$m/@Title, count($m/Actor)}
}
```

Or with the number as an attribute:

```
element Result {
for $m in //Movie
return element Movie {$m/@Title, attribute NumberOfActors {count($m/Actor)}}
}
```

EXERCISE XQ4

Find the movies that have a director that is an actor in the movie!

```
element Result {
for $m in //Movie[Actor/@Name = Director/@Name]
return element Movie {$m/@Title}
}
```

Or a solution with a where clause:

```
element Result {
for $m in //Movie
where some $a in $m/Actor satisfies $a/@Name = $m/Director/@Name
return element Movie {$m/@Title}
}
```

Or with the director in focus:

```
element Result {
for $m in //Director[@Name = ../Actor/@Name]/..
return element Movie {$m/@Title}
}
```

EXERCISE XQ5

Show the movies per production company!

```
element Result {
for $p in distinct-values(//ProductionCompany)
let $movies := for $m in //Movie[ProductionCompany=$p]/@Title/string()
                return element Movie {$m}
return element Company {attribute Name {$p}, $movies}
}
```

EXERCISE XQ6

Show the movies per actor!

```
element Result {
for $an in distinct-values(//Actor/@Name)
return element Actor {attribute Name {$an},
                      for $mt in //Movie[Actor/@Name=$an]/@Title
                      return element Movie {$mt}}
}
```

Or with a let clause:

```
element Result {
for $an in distinct-values(//Actor/@Name)
let $movies := for $mt in //Movie[Actor/@Name=$an]/@Title
                return element Movie {$mt}
return element Actor {attribute Name {$an}, $movies}
}
```

With XQuery 3:

```
element Result {
for $a in //Actor
group by $an := $a/@Name
let $movies := for $m in $a/.. return element Movie {$m/@Title}
return element Actor {attribute Name {$an}, $movies}
}
```

Or if we create the Movie elements before grouping:

```
element Result {
for $a in //Actor
let $m := element Movie {$a/../@Title}
group by $an := $a/@Name
return element Actor {attribute Name {$an}, $m}
}
```

Or with more information about the movies and actors:

```
element Result {
for $a in //Actor
let $m := element Movie {$a/../@*, attribute Director {$a/../Director/@Name}}
group by $an := $a/@Name
return element Actor {$a[1]/@*, $m}
}
```

EXERCISE XQ7

Show the age of each actor per movie!

```
element Result {
for $m in //Movie
let $actors := for $a in $m/Actor
                let $age := number($m/@Year) - number($a/@YearOfBirth)
                return element Actor {$a/@Name, attribute Age {$age}}
return element Movie {$m/@Title, $actors}
}
```

Or perhaps we wanted the result per actor and not per movie:

```
element Result {
for $a in //Actor[not(@Name = following::Actor/@Name)]
let $yob := number($a/@YearOfBirth)
let $movies := for $m in //Actor[@Name = $a/@Name]/..
                 return element Movie {$m/@Title, number($m/@Year) - $yob}
return element Actor {$a/@Name, $movies}
}
```

EXERCISE XQ8

Show the titles of movies without Sharon Stone!

```
element Result {
for $m in //Movie
where every $an in $m/Actor/@Name satisfies $an != 'Sharon Stone'
return element Movie {$m/@Title}
}
```

Or

```
element Result {
for $mt in //Movie[not(Actor/@Name = "Sharon Stone")]/@Title
return element Movie {$mt}
}
```

EXERCISE XQ9

Show the movies that only have actors from USA!

```
element Result {
for $mt in //Movie[not(Actor/@Country != "USA")]/@Title
return element Movie {$mt}
}
```

Or

```
element Result {
for $m in //Movie
where every $c in $m/Actor/@Country satisfies $c = "USA"
return element Movie {$m/@Title}
}
```

EXERCISE XQ10

Find all the actors that Eric Darnell has directed!

```
element Result {
for $p in distinct-values(//Actor[../Director/@Name="Eric Darnell"]/@Name)
return element Person {$p}
}
```

Or without having the condition in an XPath predicate:

```
element Result {
for $n in distinct-values(for $m in //Movie
                          where $m/Director/@Name = "Eric Darnell"
                          return $m/Actor/@Name)
return element Person {$n}
}
```

EXERCISE XQ11

Retrieve all the people and create an attribute that has the value "director", "actor" or "both"!

```
element Result {
for $pn in distinct-values(//@Name)
let $t :=  if (exists(//Actor[@Name = $pn]) and exists(//Director[@Name = $pn]))
           then "both"
           else if (exists(//Actor[@Name = $pn]))
                then "actor"
                else "director"
return element Person {attribute Name {$pn}, attribute Type {$t}}
}
```

We could use the type as the element name instead:

```
element Result {
for $pn in distinct-values(//@Name)
let $t :=  if (exists(//Actor[@Name=$pn]) and exists(//Director[@Name=$pn]))
           then "both"
           else if (exists(//Actor[@Name=$pn]))
                then "actor"
                else "director"
return element {$t} {attribute Name {$pn}}
}
```

With XQuery 3:

```
element Result {
for $pn in //@Name
let $en := $pn/../name()
group by $pn
let $t := switch (string-join(distinct-values($en), ""))
          case "Actor" return "actor"
          case "Director" return "director"
          default return "both"
return element Person {attribute Name {$pn}, attribute Type {$t}}
}
```

EXERCISE XQ12

What are "Driven", "Pitof", "Universal" and "Canada"? Show if they are attribute values of text nodes and for what element!

```
element Result
{
 for $v in ("Driven", "Canada", "Universal", "Pitof")
 for $type in distinct-values(//(* | @*)[. = $v]/name())
 return element Data {attribute value {$v}, attribute type {$type}}
}
```

Or a more complex solution in order to find the name of the element for attribute values:

```
element Result
{
 for $v in ("Driven", "Canada", "Universal", "Pitof")
 for $en in distinct-values(//*[. = $v]/name())
 return element TextNodeData {attribute value {$v}, attribute element {$en}} ,
```

```
  for $v in ("Driven", "Canada", "Universal", "Pitof")
  for $en in distinct-values(//*[@* = $v]/name())
  for $an in distinct-values(//*[name() = $en]/@*[. = $v]/name())
  return element AttributeNodeData {attribute value {$v}, attribute attribute {$an},
                                    attribute element {$en}}
}
```

Exercises XQ13, XQ14 and XQ15 are supposed to make changes to existing nodes. When copy-modify expressions or transform-with expressions are used, the changes are applied to a copy and not to the original. Thus, the changes are not saved. In order for the changes to be saved, the function put() should be used. Alternatively, updating expressions can be used. They return nothing, but the changes are applied to the original (in BaseX, they are applied to the database, not the file used to create the database).

EXERCISE XQ13

Fix the year of the movie "The Kid And I" to 2005!

```
copy $mcopy := /Movies
modify replace value of node $mcopy//Movie[@Title = "The Kid And I"]/@Year with 2005
return $mcopy
```

The previous solution works with the root element. This one works with the document node:

```
copy $dcopy := (/)
modify replace value of node $dcopy//Movie[@Title = "The Kid And I"]/@Year with 2005
return $dcopy
```

Using XQuery Update Facility 3, the simpler, more compact, syntax of a transform-with expressions is available:

```
//Movie[@Title = "The Kid And I"] transform with {replace value of node @Year with 2005}
```

This has the same effect as the copy-modify expression, namely modifying and returning a copy. In order to return the entire document, it needs to be the node on the left side of the transform-with expression:

```
(/) transform with {replace value of node .//Movie[@Title = "The Kid And I"]/@Year with
2099}
```

Using an updating expression like the following, makes the change in the original and does not return anything (BaseX returns the entire document after the updating expressions is done). Updating expressions require XQuery Update Facility 3.

```
replace value of node //Movie[@Title = "The Kid And I"]/@Year with 2005
```

EXERCISE XQ14

Add Joe Mantegna (1947, USA) as an actor to the movie "The Kid And I"!

```
copy $dcopy := (/)
modify insert node element Actor {attribute Name {"Joe Mantegna"},
                                  attribute YearOfBirth {1947},
                                  attribute Country {"USA"}}
               before $dcopy//Movie[@Title = "The Kid And I"]/Director
return $dcopy
```

Or without computed constructors:

```
copy $dcopy := (/)
modify insert node <Actor Name="Joe Mantegna" YearOfBirth="1947" Country="USA" />
               before $dcopy//Movie[@Title = "The Kid And I"]/Director
return $dcopy
```

EXERCISE XQ15

Fix Sylvester Stallone's name! Sylvester instead of Silvester.

```
copy $dcopy := (/)
modify (for $s in $dcopy//*[@Name = "Silvester Stallone"]
        return replace value of node $s/@Name with "Sylvester Stallone")
return $dcopy
```

EXERCISE XQ16

Restructure the data with minimal redundancy!
Would the previous exercises be easier or harder to solve against the new structure?

We can have each person once, and group them by country to avoid repeating the countries. The group the movies per productions company to avoid repeating the production company names and from each movie refer to the people that are the director and the actors:

```
element MovieData {
   for $country in distinct-values(//@Country)
   let $persons := for $pn in distinct-values(//*[@Country = $country]/@Name)
                   let $yob := (//*[@Name=$pn]/@YearOfBirth)[1]
                   return element Person {attribute Name {$pn}, $yob}
   return element Country {attribute Name {$country}, $persons},
   for $pc in distinct-values(//ProductionCompany)
   let $movies := for $m in //Movie[ProductionCompany = $pc]
                  let $director := attribute Director {$m/Director/@Name}
                  let $actors := for $a in $m/Actor return element Actor {$a/@Name}
                  return element Movie {$m/@*, $director, $actors}
   return element ProductionCompany {attribute Name {$pc}, $movies}
}
```

Solution with ids created with the count clause (XQuery 3):

```
element MovieData {
let $countries :=
   for $p in//*[@Name]
   group by $pn := $p/@Name
   count $pid
   let $person := element Person {attribute id {$pid}, $p[1]/@Name, $p[1]/@YearOfBirth}
   group by $country := $p[1]/@Country
   return element Country {attribute Name {$country}, $person}
let $companies :=
   for $pc in distinct-values(//ProductionCompany)
   let $movies := for $m in //Movie[ProductionCompany = $pc]
                  let $director := attribute Director {$countries//Person[@Name =
$m/Director/@Name]/@id}
                  let $actors := for $a in $m/Actor
                                 return element Actor {$countries//Person[@Name =
$a/@Name]/@id}
                  return element Movie {$m/@*, $director, $actors}
   return element ProductionCompany {attribute Name {$pc}, $movies}
return ($countries, $companies)
}
```

Solutions to XQ1-XQ15 based on the new structure (the first one without the IDs)

Some solutions have become simpler because of the eliminated redundancy, while others have become more complex since all the data is no longer available in the same element (for example the country and year of birth of a movie's director are not available inside the Movie element and one of them is inside Person while the other one is inside Country). Some solutions are identical.

EXERCISE XQ16:XQ1
Retrieve all the directors!

```
element Result {//Person[@Name = //@Director]}
```

This does not include the country though. To add the country, we need to get it from the parent element:

```
element Result {
for $p in //Person[@Name = //@Director]
return element Director {$p/@*, attribute Country {$p/../@Name}}
}
```

EXERCISE XQ16:XQ2

Retrieve all the actors from USA that are part of the movie Driven!

```
element Result {
  //Country[@Name="USA"]/Person[@Name = //Movie[@Title="Hide and
Seek"]/Actor/@Name]
}
```

Maybe easier to read without nested predicates:

```
element Result {
  let $actornames := //Movie[@Title="Hide and Seek"]/Actor/@Name
  return //Person[@Name = $actornames and parent::Country/@Name="USA"]
}
```

EXERCISE XQ16:XQ3

Show the number of actors per movie!

```
element Result {
for $m in //Movie
return element Movie {$m/@Title, attribute NumberOfActors {count($m/Actor)}}
}
```

EXERCISE XQ16:XQ4

Find the movies that have a director that is an actor in the movie!

```
element Result {
for $f in //Movie[Actor/@Name = @Director]
return element Movie {$f/@Title}
}
```

EXERCISE XQ16:XQ5

Show the movies per production company!

```
element Result {
//ProductionCompany
}
```

This includes the actors and the director though. If we only want the title:

```
element Result {
for $pc in //ProductionCompany
let $movies := for $m in $pc/Movie return element Movie {$m/@Title}
return element Company {$pc/@Name, $movies}
}
```

EXERCISE XQ16:XQ6

Show the movies per actor!

```
element Result {
for $a in //Person[@Name = //Actor/@Name]
let $movies := for $mt in //Movie[Actor/@Name=$a/@Name]/@Title
                return element Movie {$mt}
return element Actor {$a/@Name, $movies}
}
```

EXERCISE XQ16:XQ7

Show the age of each actor per movie!

```
element Result {
for $m in //Movie
let $actors := for $a in //Person[@Name = $m/Actor/@Name]
               let $age := number($m/@Year) - number($a/@YearOfBirth)
               return element Actor {$a/@Name, attribute Age {$age}}
return element Movie {$m/@Title, $actors}
}
```

EXERCISE XQ16:XQ8

Show the titles of movies without Sharon Stone!

```
element Result {
for $m in //Movie
where every $an in $m/Actor/@Name satisfies $an != 'Sharon Stone'
return element Movie {$m/@Title}
}
```

Or

```
element Result {
for $mt in //Movie[not(Actor/@Name = "Sharon Stone")]/@Title
return element Movie {$mt}
}
```

EXERCISE XQ16:XQ9

Show the movies that only have actors from USA!

```
element Result {
for $m in //Movie
let $countries := distinct-values(//Person[@Name = $m/Actor/@Name]/../@Name)
where not($countries[2]) and $countries = "USA"
return element Movie {$m/@Title}
}
```

EXERCISE XQ16:XQ10

Find all the actors that Eric Darnell has directed!

```
element Result {
for $p in distinct-values(//Actor[../@Director="Eric Darnell"]/@Name)
return element Person {$p}
}
```

Or if we want to keep the Person element:

```
element Result {
//Person[@Name = //Actor[../@Director="Eric Darnell"]/@Name]
}
```

And if we want to include the country:

```
element Result {
for $p in //Person[@Name = //Actor[../@Director="Eric Darnell"]/@Name]
return element Actor {$p/@*, attribute Country {$p/parent::Country/@Name}}
}
```

EXERCISE XQ16:XQ11

Retrieve all the people and create an attribute that has the value "director", "actor" or "both"!

```
element Result {
for $p in //Person/@Name
let $type :=  if (exists(//Actor[@Name = $p]) and exists(//Movie[@Director=$p]))
                        then "both"
                        else if (exists(//Actor[@Name = $p]))
                                                then "actor"
                                                else "director"
return element Person {$p, attribute Type {$type}}
}
```

EXERCISE XQ16:XQ12

What are "Driven", "Pitof", "Universal" and "Canada"? Show if they are attribute values of text nodes and for what element!

```
element Result
{
  for $v in ("Driven", "Canada", "Universal", "Pitof")
  for $type in distinct-values(//(* | @*)[. = $v]/name())
  return element Data {attribute value {$v}, attribute type {$type}}
}
```

Or a more complex solution in order to find the name of the element for attribute values:

```
element Result
{
  for $v in ("Driven", "Canada", "Universal", "Pitof")
  for $en in distinct-values(//*[. = $v]/name())
  return element TextNodeData {attribute value {$v}, attribute element {$en}} ,

  for $v in ("Driven", "Canada", "Universal", "Pitof")
  for $en in distinct-values(//*[@* = $v]/name())
  for $an in distinct-values(//*[name() = $en]/@*[. = $v]/name())
  return element AttributeNodeData {attribute value {$v}, attribute attribute {$an},
                                    attribute element {$en}}
}
```

Our new structure does not have any text nodes, so the first part will not return anything. Also, since we have used the names as references, Pitof is now both the name of a Director attribute and a Name attribute.

EXERCISE XQ16:XQ13

Fix the year of the movie "The Kid And I" to 2005!

Unchanged. The new structure does not affect the solution to this exercise.

EXERCISE XQ16:XQ14

Add Joe Mantegna (1947, USA) as an actor to the movie "The Kid And I"!

Here we need to add Joe Mantegna as a person (if he is not already there) and as an actor to right movie. And in order to add him as a person, we may need to create a country (if USA is not already there). We can do this with the following:

```
copy $dcopy := (/)
modify (
  let $joe := element Person {attribute Name {"Joe Mantegna"}, attribute YearOfBirth {1947}}
  return if (//Country[@Name = "USA"])
         then if (//Person[@Name = "Joe Mantegna"])
              then ()
              else insert node $joe as last into $dcopy//Country[@Name = "USA"]
         else insert node element Country {attribute Name {"USA"}, $joe}
                   as first into $dcopy/MovieData
         ,
         insert node element Actor {attribute Name {"Joe Mantegna"}}
                as last into $dcopy//Movie[@Title = "The Kid And I"]
    )
return $dcopy
```

EXERCISE XQ16:XQ15

Fix Sylvester Stallone's name! Sylvester instead of Silvester.

Here we need to fix the name both for the Person element and for all references (from Movie/Actor/@Name and Movie/@Director):

```
copy $dcopy := (/)
modify (let $oldname := "Silvester Stallone"
        for $an in ($dcopy//Person/@Name[. = $oldname],
                    $dcopy//Movie/@Director[. = $oldname],
                    $dcopy//Movie/Actor/@Name[. = $oldname])
        return replace value of node $an with "Sylvester Stallone")
return $dcopy
```