3. Unit: XQuery & Mondial

Information about the XML course, recommended tools as well as the Mondial Database, is found under
`http://www.stud.informatik.uni-goettingen.de/xml-lecture`

The following exercises use the *Mondial* database and should be solved using the XQuery web interface of the *eXist* XML database system[a] and/or using the saxon XQuery engine.

[a] Remember to set the collection context in the web interface to "/db/xmlcourse".

Exercise 3.1 Give name and population of the country with the highest population.

```
for $ctr in /mondial/country
where($ctr/population = max(/mondial/country/population))
return
<result>
{$ctr/name}
{$ctr/population}
</result>

(: the where-clause can also be moved into the XPath part,
although it is harder to understand then :)

for $ctr in /mondial/country[population = max(/mondial/country/population)]
return
<result>
{$ctr/name}
{$ctr/population}
</result>

(: or, because it is only one country, also a 'let' can be used: :)

let $ctr := /mondial/country[population = max(/mondial/country/population)]
return
<result>
{$ctr/name}
{$ctr/population}
</result>

(: Ergebnis: China 1210004956 :)
```

Exercise 3.2 For each organization, return its name and the sum of the population of its members (in descending order, ignore different member types).

```
for $org in /mondial//organization
let $sum := sum($org/members/id(@country)/population)
order by $sum descending
```

```
return
<result>
  <org>{$org/name}</org>
  <pop>{$sum}</pop>
</result>
(: a typical for-let-combination :)
(: 168 hits (including organizations where no member are stored, otherwise 152) :)
(: first result: International Olympic Committee, pop = 5741497820 :)
```

Exercise 3.3 Imagine the moment of sunrise in Dakar on 21th of September. Which is the city where the sun rises next?

```
let $cities :=
  for $c in /mondial//city
  where (number($c/longitude) < number(/mondial//city[name = 'Dakar']/longitude))
  return $c
for $city in $cities
where $city/longitude = max($cities/longitude)
return $city

(: another nice example for preparing using a 'let' :)
(: Ergebnis: Hafnarfjoerdur,IS,Iceland,12000,-22,64 :)
```

Exercise 3.4 Which lakes, seas and rivers does Russia share with *exactly one* other country?

```
for $water in /mondial//(lake|river|sea)
where $water/located/id(@country)/name="Russia"
  and count($water/located/id(@country)) = 2
order by $water/name
return
  element {$water/name()} {$water/name/text()}
(: result: 9 items: Argun,Dnepr,Irtysch,Ischim, ... :)
(: ...Ozero Chanka,Sea of Azov,Sea of Japan,Tobol,Ural :)
(: note the explicit result element constructor :)
```

Exercise 3.5 The database contains redundant information about the population of countries (in country and in province). Compute all inconsistent countries, i.e., those whose population differs more than 10% from the sum of the population of their provinces.

```
for $con in /mondial/continent
let $area := sum (
  for $c in /mondial/country[id(encompassed/@continent)=$con]
  return (($c/@area) * ($c/encompassed[@continent=$con/@id]/@percentage) div 100))
where $area > $con/area/text() * 1.1
   or $area < $con/area/text() * 0.9
return <continent>
            {$con/name}
```

```
            {$con/area}
            {$area}
          </continent>
(: yes: Australia :)
(: note that the 'where' is evaluated after the 'let' (and uses the
   let's variable)! :)
```

Exercise 3.6 Compute all pairs of european countries that are adjacent to the same set of seas.

```
(: mondial.xml hat leider keine vollstaendige Meeresdaten (z.B. fuer Albanien) :)

let $europcountries := /mondial/country[encompassed/@continent="europe"]
for $c1 in $europcountries
let $seas1 := /mondial/sea[located/@country = $c1/@car_code]/name
for $c2 in $europcountries
let $seas2 := /mondial/sea[located/@country = $c2/@car_code]/name
where $c1/name/text() < $c2/name/text()
  and exists($seas1)
  and deep-equal($seas1,$seas2)
return  <result>{$c1/name} {$c2/name} {$seas1}</result>

(: it is also possible to compare the sets item-by-item instead of
   using deep-equal (which deep-compares the complete XML sequences
   bound to the variables)
   Note the implicit set-based comparisons in the 'every' parts
   with $seas1 and $seas2 :)

let $europcountries := /mondial/country[encompassed/@continent="europe"]
for $c1 in $europcountries
let $seas1 := /mondial/sea[located/@country = $c1/@car_code]/name
for $c2 in $europcountries
let $seas2 := /mondial/sea[located/@country = $c2/@car_code]/name
where $c1/name/text() < $c2/name/text()
  and (every $s1 in $seas1 satisfies $s1 = $seas2)
  and (every $s2 in $seas2 satisfies $s2 = $seas1)
return  <result>{$c1/name} {$c2/name} {$seas1}
        </result>

(: result: 598 pairs or 3 non landlocked pairs :)
```

Exercise 3.7 How many countries and how much area are encompassed by the islands in the Caribbean Sea?

```
let $countries := /mondial/sea[name="Caribbean Sea"]/located/id(@country)
return
<result>
  {$countries/name}
  <area> {sum($countries/@area)} </area>
</result>
```

```
(: result: 7 countries, 4375760 qkm (again, incomplete data mondial.xml) :)
```

Exercise 3.8 For all countries, give the sum of the population of all its neighbors.

```
for $c in /mondial/country
let $sum := sum($c/border/id(@country)/population)
return
<result>
  {$c/name}
  <neighbor_pop>{$sum}</neighbor_pop>
</result>
(: Ergebnis: es sollten alle 260 Laender sein :)
(: Albanien 23257187  :)
(: Andorra 97498564  :)
(: Austria  175884037  :)
(: note that for islands (which do not have neighbors), a '0' is explicitly
   returned which is different from join-based SQL solutions where an
   outer join must explicitly be forced :)
```

Exercise 3.9 For each country with at least 3 cities, compute the sum of the inhabitants of the three biggest cities.

```
for $country in /mondial//country[count(.//city) > 2]
 let $cities_pops :=
  (for $c in $country//city[population]
   let $pnum := number($c/population[1])
   order by $pnum descending
   return $c/population[1]
  )
return
<result>
  {$country/name}
  <three-cities>
    {sum($cities_pops[position()<=3])}
  </three-cities>
</result>

(: - note that the intermediate result $cities_pops is an ordered
     sequence of nodes
   - take only cities that have a population entry :)
(: Result: 82 items, Albania, 314000 :)

(: In XML it is also possible to return the names of the largest three
   cities, and the sum of their population: :)
(: xs:int used since fn:number does not work :)

declare namespace xhtml = "bla";
for $country in /mondial//country[count(.//city) > 2]
```

```
let $cities :=
  (for $c in $country//city[population]
   order by xs:int($c/population[1]) descending
   return $c
  )
return
<result>
  {$country/name}
  <three-cities>
    {$cities[position()=1]/name}
    {$cities[position()=2]/name}
    {$cities[position()=3]/name}
    <sum>{sum($cities[position()<=3]/population)}</sum>
  </three-cities>
</result>
```

Exercise 3.10 Compute all cities that have more inhabitants that the average of all cities in that country.

```
for $country in /mondial/country[.//city/population]
let $cities := $country//city[population]
let $pops := $cities/population[1]
let $avg_pop := sum($pops) div count($cities)
let $bigcities := $country//city[number(./population[1]) >= number($avg_pop)]
return
<result>
  <country>{$country/name/text()}</country>
  <cities>{$bigcities/name}</cities>
  <average>{$avg_pop}</average>
</result>

(: result: 565 items :)
```
