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 XQuery.

Exercise 3.1 (Mondial - Maximum Population) 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>

(: or as XPath :)
//country[population = max(//country/population)]/(name|population)

(: Result: China 1210004956 :)
```

Exercise 3.2 (Mondial - order organizations by inhabitants)

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 //organization
let $sum := sum($org/members/id(@country)/population[1])
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 (Mondial - Sunrise in Dakar)

Consider the moment of sunrise in Dakar on 21st 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 (Mondial - Sharing Waters with Russia)

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 :)

(: Short in XPath :)
/mondial/(sea|river|lake)[located/@country="R"
       and count (located) = 2]/name/text()
```

Exercise 3.5 (Mondial - European Countries and Seas)

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

```
let $europcountries := /mondial/country[encompassed/id(@continent)/name="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/id(@continent)/name="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>

(: faster solution: compute seas only once :)

let $tmp :=
  for $c in /mondial/country[encompassed/@continent="europe"]
  return
   <country>
     { $c/name }
     <seas>
       { /mondial/sea[id(@country) is $c]/name }
     </seas>
   </country>
for $c1 in $tmp, (: runs over the <country> elements in $tmp :)
    $c2 in $tmp
where $c1/name/text() < $c2/name/text()
  and $seas1/name and $c2/seas/name  (: only the nonempty ones are of interest :)
  and deep-equal($c1/seas,$c2/seas)
return  <result>{$c1/name} {$c2/name} {$c1/seas}</result>

(: Med.Sea: MC/IT/SRB/AL/MAL/CY/GR
   Baltic: PL/SF/LT/LV
   North: B/NL
   North+Baltic: D/S/DK :)
```

Exercise 3.6 (Mondial - The Caribbean)

How many countries are adjacent to (or ecompassed by) the the Caribbean Sea? How much area do they cover altogether?

```
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 3.7 ("Every" and "Some" - a Comparison)
Consider again Exercise 3.30. Solve each of the below queries by using the "every ... satisfies" or "some ... satisfies" construct. Give also an XPath 1.0 solution if possible. Discuss the alternative variants.

- Give the names of all organizations that have no european member countries.
- Give the names of all organizations that have at least one european member country.
- Give the names of all organizations that have *only* european member countries.
- Give the names of all organizations where *all* european countries *which are members of at least 2 organizations* are members.

```
(: IMPORTANT: for ‘‘every’’, do not consider organizations
   where no members are listed :)

(: no europeans: 45 results : [saxon: alle 3 Anfragen  uebereinstimmend]
  ACP  OPANAL  ABEDA  ACC  AFESD  AL  AMU  AMF  APEC  ASEAN  Mekong Group  ANZUS
  Caricom  UDEAC  BCIE  CACM  CP  CAEU  Entente  EADB  ESCWA  CEEAC  CEPGL  ECOWAS
  G-2  G-3  G-11  G-15  G-19  G-24  GCC  IGADD  LAES  OAU  OAPEC  OECS  OPEC  RG
  SAARC  SPF  Sparteca  SACU  SADC  Mercosur  WADB :)

/mondial/organization[members]
   [not (members/id(@country)/encompassed/id(@continent)/name="Europe")]/name

/mondial/organization
   [members and
    (every $c in members/id(@country)/encompassed/id(@continent)
      satisfies $c/name!="Europe")]/name

for $org in /mondial/organization[members]
let $con := $org/members/id(@country)/encompassed/id(@continent)
where every $c in $con/name/text() satisfies $c != "Europe"
return <answer>
         {$org/name}
         {$con}
       </answer>
```

```
(: some europeans: 108 results [saxon: alle 3 Anfragen uebereinstimmend] :)

/mondial/organization
   [members/id(@country)/encompassed/id(@continent)/name="Europe"]/name

/mondial/organization
   [some $c in members/id(@country)/encompassed/id(@continent)
     satisfies $c/name="Europe"]/name

for $org in /mondial/organization
let $con := $org/members/id(@country)/encompassed/id(@continent)
where some $c in $con/name/text() satisfies $c = "Europe"
return <answer>
         {$org/name}
         {$con}
       </answer>
```

```
(: only europeans: 8 hits [saxon: alle 3 Anfragen uebereinstimmend]
    Benelux Economic Union
    Central European Initiative
    European Free Trade Association
    European Investment Bank
    European Union
    Group of 9
    Nordic Council
    Nordic Investment Bank

  Note: different results can be due to ‘‘only countries that are
   completely in Europe’’ vs. countries that are at least partly in
   Europe’’ :)

/mondial/organization
 [members and
  not (members/id(@country)/encompassed/id(@continent)/name != "Europe")]/name

/mondial/organization
 [members and
  (every $c in members/id(@country)/encompassed/id(@continent)
   satisfies $c/name="Europe")]/name

for $org in /mondial/organization
let $con := $org/members/id(@country)/encompassed/id(@continent)
where $org/members and
  (every $c in $con/name/text() satisfies $c = "Europe")
return <answer>
         {$org/name}
         {$con}
       </answer>
```

```
(: all europeans: 3 hits [saxon: beide Anfragen uebereinstimmend]
    International Telecommunication Union
    United Nations
    World Intellectual Property Organization   :)

let $europeancountries :=
  /mondial/country[
    count(id(@memberships)) > 1 and
    encompassed/id(@continent)/name="Europe"]
for $org in /mondial/organization
where every $c in $europeancountries satisfies
                  $c = $org/members/id(@country)
return $org/name

for $org in /mondial/organization
where not
  ( /mondial/country[
    count(id(@memberships)) > 1 and
    encompassed/id(@continent)/name="Europe"
    and not (.= $org/members/id(@country))])
return $org/name

(: here: NO WAY IN XPATH SINCE JOIN IS NEEDED INSIDE NOT/NOT) :)
(: THE FOLLOWING ILLUSTRATES THE PROBLEM :)
/mondial/organization
 [not
  ( /mondial/country[
    count(id(@memberships)) > 1 and
    encompassed/id(@continent)/name="Europe"
    and not (COUNTRY = ORG/members/id(@country))])]/name
```

Discussion:

- "`some ... satisfies`" is redundant since the XPath set comparison has implicit existential semantics
- "`every ... satisfies`" is nice syntactic sugar, but can also be replaced by "`not some (not ...)`" or even "not (not ...)". The latter is also the usual way to solve such things in SQL.
- the 4th query, there is no way to transform it into XPath because a join is needed in the inner subquery.

Exercise 3.8 (Mondial - Population of Neighbors)

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: 260 countries
    Albania 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 (Mondial - Biggest Cities) 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 :)

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 (Mondial - Cities population above average)

Give all cities that have more inhabitants than the average of all cities in that country.

```
(: result: 565 items :)

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>

for $c in //country[count(city/population/text())=count(city)]
(: some countries have cities with two population numbers :)
let $avg := avg($c//city/population[1]/text())
return
<country>
  {$c/name}
  <avg>{$avg}</avg>
  {
    for $city in $c//city
    where $city/population/text() > $avg
    return
    <city>
          {$city/name}
          {$city/population}
    </city>}
</country>
```

Exercise 3.11 (User-defined Function: Functional Programming – Faculty) Write a recursive function that computes the faculty of a natural number.

```
(:call saxonXQ faculty.xq x=5 :)
declare variable $x external;
declare function local:faculty($n as xs:integer) as xs:integer
{  if ($n=1) then 1
   else $n* local:faculty($n - 1)
};
local:faculty($x)
```