3. Unit: XQuery

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 mostly 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[last()]))
return
<result>
{$ctr/name}
{$ctr/population[last()]}
</result>

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

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

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

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

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

(: Result: China 1360720000 :)
```

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[last()])
order by $sum descending
return
<result>
  <org>{$org/name}</org>
  <pop>{$sum}</pop>
</result>
(: a typical for-let-combination :)
(: 168 results :)
(: first result: United Nations, pop = 7.046.917.773 :)
```

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 $water/id(@country)/name="Russia"
  and count($water/id(@country)) = 2
order by $water/name
return
  element {$water/name()} {$water/name/text()}
(: result: 16 items :)
(: note the explicit result element constructor :)
(: note: river/located/@country exists only for countries that have provinces! :)

(: Short in XPath :)
/mondial/(sea|river|lake)[located/@country="R"
       and count (id(@country)) = 2]/name/text()
(: located/@country=R saves the dereferencing; R has provinces :)
```

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[id(@country)/@car_code = $c1/@car_code]/name
for $c2 in $europcountries
let $seas2 := /mondial/sea[id(@country)/@car_code = $c2/@car_code]/name
where $c1/name/text() < $c2/name/text()
  and exists($seas1)    (: not the empty set of seas :)
  and deep-equal($seas1,$seas2)
return  <result>{$c1/name} {$c2/name} {$seas1}</result>

(: 59 results :
   Med.Sea: MC/IT/SLO/HR/BIH/MNE/AL/GR/MAL/CY = 45 Pairs
   Black Sea: BG/RO 1 Pair
   Baltic: PL/SF/LT/LV/EW   10 Pairs
   North: B/NL      1 Pair
   Channel: GBG/GBJ 1 Pair
   Atl+Med: E/GBZ   1 Pair   = 59 Pairs
   :)
(: 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[id(@country)/@car_code = $c1/@car_code]/name
for $c2 in $europcountries
let $seas2 := /mondial/sea[id(@country)/@car_code = $c2/@car_code]/name
where $c1/name/text() < $c2/name/text()
  and exists($seas1)
  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)/@car_code = $c/@car_code]/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 $c2/seas/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>
```

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

(: result: 33 countries, 4.745858E6 sqkm :)
```

Exercise 3.7 ("Every" and "Some" - a Comparison)
Consider again Exercise 3.28. 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 at least one european member country.
- Give the names of all organizations that have no european member countries.
- Give the names of all organizations that have *only* member countries that are (at least partly) located in Europe.
- Give the names of all organizations where *all* european countries *which are members of at least 10 organizations* are members.

```
(: some europeans: 129 results -- three variants:  :)

/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 satisfies $c/name = "Europe"
return <answer>
         {$org/name}
         {$con/name}
       </answer>
```

```
(: no europeans: 39 results
```

```
   note that a country might be in europe and also on another continent (R,TR) :)

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

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

for $org in /mondial/organization
where every $c in $org/members/id(@country)
      satisfies (every $cont in $c/encompassed/id(@continent)
                 satisfies $cont/name != "Europe")
return $org/name
```

```
(: only europeans: 11 hits

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

(: the pure XPath variant is hard to read: :)
/mondial/organization
 [not (members/id(@country)[not(encompassed/id(@continent)/name = "Europe")])]/name

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

(:  the explit every-some makes it easy to understand :)
/mondial/organization
 [ every $c in members/id(@country)
   satisfies (some $cont in $c/encompassed/id(@continent)
              satisfies $cont/name="Europe")]/name

for $o in /mondial/organization
where every $c in $o/members/id(@country)
      satisfies (some $cont in $c/encompassed/id(@continent)
                 satisfies $cont/name="Europe")
return $o/name
```

```
(: all europeans with >= 10 memberships: 5 hits
   International Criminal Police Organization
   International Federation of Red Cross and Red Crescent Societies
   Organization for Security and Cooperation in Europe
   Organization for the Prohibition of Chemical Weapons
   United Nations

let $europeancountries :=
```

```
  /mondial/country[
    count(id(@memberships)) >= 10 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

(: SQL relational devision style with not(exists)-not(exists)
for $org in /mondial/organization
where not
  ( /mondial/country[
    count(id(@memberships)) > 10 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 condition is needed in the inner subquery, which requires variables.

Exercise 3.8 (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[last()])
   order by $pnum descending
   return $c/population[last()]
  )
return
<result>
  {$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: 117 items, Albania, 611257 :)

(: 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[last()]) descending
   return $c
  )
return
<result>
  {$country/name}
  <three-cities>
    {$cities[position()=1]/name[1]}
    {$cities[position()=2]/name[1]}
    {$cities[position()=3]/name[1]}
    <sum>{sum($cities[position()<=3]/population[last()])}</sum>
  </three-cities>
</result>
```

Exercise 3.9 (Mondial - Cities population above average)

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

```
(: result: 213 countries :)

(: first, just have a look for all countries (some have no city
   that ist bigger than average)

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

(: now to the desired output: all cities :)
for $c in //country[count(city/population/text())=count(city)]
```

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

Exercise 3.10 (Hamlet - Anzahl SPEECHes) Create an HTML table which lists for every person (use the `//PERSONAE` list at the beginning of the XML document) how many speeches he/she gives.

```
(: Note: for some persons like Claudius, SPEECHES lists "King Claudius" :)
<html>
<table>
{
for $p in //PERSONA/text()
let $person :=
 if (contains($p,",")) then substring-before($p,",")
 else $p
where //SPEECH[some $x in ./SPEAKER satisfies contains($x,$person)]
return
<tr>
  <td>
    { $person }
  </td>
  <td>
    { count(//SPEECH[some $x in ./SPEAKER satisfies contains($x,$person)]) }
  </td>
</tr>
}
</table>
</html>
```

Exercise 3.11 (Mondial & Hamlet)

Which countries (from Mondial) are mentioned in "Hamlet"? Give also the corresponding LINEs. Do the same for the cities. Where's a problem?

```
(: use Hamlet as context document and access Mondial from the Web :)
for $name in doc('http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml')//country/name
(: where contains(/,$name)  return $name    -- checks only for occurrence :)
for $line in //LINE
where contains($line,$name)
return <result>{$name/text, $line}</result>
```

```
(: use Hamlet as context document and access Mondial from the Web :)
for $name in doc('http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml')//city/name
(: where contains(/,$name)      -- checks only for occurrence :)
for $line in //LINE
where contains($line,$name)
return <result>{$name/text(), $line}</result>
```

Some city names occur as sub-words (like "Bern" in "Bernardo", "Gent" in "Gentlemen"). Thus, it must be checked for whole words (capitalization already guarantees the beginning of a word, but the end, including "end of string" must be checked):

```
for $name in doc('http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml')//city/name
for $line in //LINE[contains(.,$name)]
  (: the [contains ...] is now redundant, but makes it much more efficient
     since 'contains' is cheaper to evaluate than regex matching :)
where matches($line,concat($name,"\s"))  (: regex matching: whitespace :)
   or matches($line,concat($name,"$"))   (: regex matching: end of string :)
   or contains($line,concat($name,"."))
   or contains($line,concat($name,";"))
   or contains($line,concat($name,":"))
   or contains($line,concat($name,","))  (: and maybe some more :)
return <result>{$name/text(), $line}</result>
```

Exercise 3.12 (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)
```