XQuery exercises

A. Restaurants

Given the following DTD:

```
 <!ELEMENT restaurants (restaurant+)>
 <!ELEMENT restaurant (name, dish+)>
<!ELEMENT dish (name, ingredient+, price)>
<!ELEMENT ingredient (name, weight)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

A. Restaurants

1. List all dishes served by restaurant “UnibgFood”?
2. For each restaurant, return the number of available dishes
3. How many restaurants serve dishes with mushrooms?
4. For each restaurant identify the most expensive dish(es)
5. For each restaurant identify the dishes that cost at least 90% of the price of the most expensive dish (in desc. price order)
6. Which is the restaurant with the maximum number of dishes?
7. For each ingredient return the list of restaurants that use it
8. For each restaurant list the names of its dishes, sorting them by number of contained ingredients

A. Restaurants

1. List all dishes served by restaurant “UnibgFood”?

```
for $d in doc(“rest.xml”)/restaurants/restaurant[name=“UnibgFood”]/dish
return <dish> {$d/name} </dish>
```

```
for $r in doc(“rest.xml”)/restaurants/restaurant
let $d := $r/dish
where $r/name=“UnibgFood”
return $d
```

```
for $r in doc(“rest.xml”)/restaurants/restaurant
where $r/name=“UnibgFood”
return $r/dish
```

```
doc(“rest.xml”)/restaurants/restaurant[name=“UnibgFood”]/dish/name
```

A. Restaurants

2. For each restaurant, return the number of available dishes

```
for $r in doc(“rest.xml”)/restaurants/restaurant
return    <Restaurant>
              { $r/name }
              <NumberOfDishes> { count( $r/dish ) } </NumberOfDishes>
          </Restaurant>
```

```
for $r in doc(“rest.xml”)/restaurants/restaurant
let $nod := count( $r/dish )
return    <Restaurant name="{ $r/name/text() }">
              <NumberOfDishes> { $nod } </NumberOfDishes>
          </Restaurant>
```

```
<NumberOfDishesPerRestaurant> {
    for $r in doc(“rest.xml”)/restaurants/restaurant
    return <item> { $r/name , count( $r/dish ) } </item>
} <NumberOfDishesPerRestaurant>
```

A. Restaurants

3. How many restaurants serve dishes with mushrooms?

```
let $r := doc(“rest.xml”)/restaurants/restaurant[dish/ingredient/name=“mushroom”]
return      <count>
                 { count( $r ) }
            </count>

let $r := ( for $re in doc(“rest.xml”)/restaurants/restaurant
            where $re/dish/ingredient/name=“mushroom”
            return $re )
return      <count>
                 { count( $r ) }
            </count>

<count> {
count( doc(“rest.xml”)/restaurants/restaurant[dish/ingredient/name=“mushroom”] )
} </count>
```

A. Restaurants

4. For each restaurant identify the most expensive dish(es)

```
for $r in doc(“rest.xml”)/restaurants/restaurant
let $maxprice:= max($r/dish/price)
return    <Restaurant_MostExpensiveDishes>
          {
               <RestName> { $r/name/text() } </RestName>,
               <TopPrice> { $maxprice } </TopPrice>,
               <WhatYouBuy>
               {    for $x at $pos in $r/dish[price>=$maxprice]
                    return    if ( $pos = 1 )
                              then ( $x/name/text() )
                              else ( ", " , $x/name/text() )
               }
               </WhatYouBuy>
          }
          </Restaurant_MostExpensiveDishes>
```

A. Restaurants

5. For each restaurant identify the dishes that cost at least 90% of the price of the most expensive dish (in desc. price order)

```
for $r in doc(“rest.xml”)/restaurants/restaurant
let $maxprice:= max($r/dish/price)
return    <Restaurant_MostExpensiveDishes>
          {
                <RestName> { $r/name/text() } </RestName>
                <ListOfMED>
                      { for $d in $r/dish[price>= 0.9 * $maxprice]
                      order by $d/price descending
                      return <Dish> { $d/name/text(), $d/price/text() } </Dish>
                      }
                </ListOfMED>
          }
          </Restaurant_MostExpensiveDishes>
```

A. Restaurants

6. Which is the restaurant with the maximum number of dishes?

```
let $maxnd:= max( for $r in doc(“rest.xml”)/restaurants/restaurant
                  return count($r/dish) )
return     <restaurant>
           {
                doc(“rest.xml”)/restaurants/restaurant[count(dish)=$maxnd]/name
           }
           </restaurant>
```

```
doc(“rest.xml”)/restaurants/restaurant[
           count(dish)
           = max( for $r in doc(“rest.xml”)/restaurants/restaurant return count($r/dish)
                )]/name
```

A. Restaurants

7. For each ingredient return the list of restaurants that use it

```
for $in in distinct-values(
                            doc(“rest.xml”)/restaurants/restaurant/dish/ingredient/name
                            )
return <ingredient>
          <name> { $in } </name>
          <restaurants>
          {
               for $r in doc(“rest.xml”)/restaurants/restaurant
               where $r/dish/ingredient/name=$in
               return $r/name
          }
          </restaurants>
       </ingredient>
```

A. Restaurants

8. For each restaurant list the names of its dishes, sorting them by number of contained ingredients

```
<result>
{
    for $r in doc(“rest.xml”)/restaurants/restaurant
    return <restaurant>
            <name> { $r/name } </name>
            <dishes> {
                      for $d in $r/dish
                      order by count( $d/ingredient )
                      return $d/name
                    }
            </dishes>
</restaurant>
}
<result>
```

B. Santa Claus goes geek

The website of Good Old Santa allows children to submit letters with their wishlist for Christmas via Web Service invocations, whose WSDL exposes an interface with the following format for data encoding:

```
<!ELEMENT Letters (Letter*)>
<!ELEMENT Letter ( ChildName, Address, Country, Toy+ )>
<!ELEMENT NameChild (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ATTLIST Country TimeZone CDATA #Implied>
<!ELEMENT Toy (#PCDATA)>
<!ATTLIST Toy weight CDATA #Required>
```

B. Santa Claus goes geek

1. Find the names of all children who did not ask for a mobile phone
2. Find the name of the Italian child whose letter has the heaviest toy list
3. For each country extract the average number of toys asked by children

B. Santa Claus goes geek

1. Find the names of all children who did not ask for a mobile phone

```
doc(“...”)//Letter[ not(Toy/text()=“Mobile phone”) ]/ChildName

for $letter in doc(“...”)//Letter
where not( $letter/Toy/text() = “Mobile phone” )
return $letter/ChildName
```

B. Santa Claus goes geek

2. Find the name of the Italian child whose letter has the heaviest toy list

```
for $lel in doc(“...”)//Letter[Country=”Italy”]
where     sum( $lel/Toy/@weight ) =
          max(      for $lil in doc(“...”)//Letter[Country=”Italy”]
                    return sum( $lil/Toy/@weight )   )
return $lel/ChildName

let $lettersItaly := doc(“...”)//Letter[Country=”Italy”]
for $lel in $lettersItaly
where sum( $lel/Toy/@weight ) = max( for $lil in $lettersItaly
                                     return sum( $lil/Toy/@weight )   )
return $lel/ChildName
```

B. Santa Claus goes geek

3. For each country extract the average number of toys asked by children

```
for $n in distinct-values( doc(“santa.xml”)//Country )
let $letts_from_n := doc(“santa.xml”)//Letter[Country = $n ]
return     <CountryStats country=”{ $n }”>
                 <AvgToysNum>
                       { count($letts_from_n//Toy) div count($letts_from_n) }
                 </AvgToysNum>
           </CountryStats>

<AvgToysNum>
      {     avg(       for $x in $letts_from_n
                 return count($x/Toy) )        }
</AvgToysNum>
```

C. I have a drink

```
<!ELEMENT Cocktails ( Cocktail+, Ingredient+ )>
<!ELEMENT Cocktail ( Name, Component+, GlassType,
Garnish, Procedure )>
<!ELEMENT Component ( IngredientName, Quantity )>
<!ELEMENT Ingredient ( Name, CaloriesPerGram )>
```

This DTD describes a portion of the content of a Web-accessible barman course. Unspecified elements only contain PCDATA. Quantity is expressed in milliliters or grams (respectively for liquids and solids) and refers to the amount required for a cocktail for one person. The ingredient name identifies the ingredient.

C. I have a drink

1. For each ingredient, the list of the names of cocktails containing it
2. Build in XQuery a “summary” document that, for each cocktail, presents:
   - the total amount of calories in the cocktail (considering ALL the ingredients);
   - the list of the 4 ingredients used in the largest quantity (consider milliliters and grams as equivalent); in the case of a tie, always choose 4 ingredients based on the alphabetical order.

C. I have a drink

1. For each ingredient, the list of the names of cocktails containing it

```
<TOC> {
      for $i in //Ingredient/Name
      return <Ingredient name="{ $i }">
                  { for $n in //Cocktail[ Component/IngredientName = $i ]/Name
                  return <Cocktail> { $n } </Cocktail> }
            </Ingredient>
} </TOC>

<TOC> {
      for $i in //Ingredient/Name
      return <Ingredient name="{ $i }">
                  { //Cocktail[ Component/IngredientName = $i ]/Name }
            </Ingredient>
} </TOC>
```

C. I have a drink

2. Summary build

```
<Summary>{
    for $c in //Cocktail
        let $ordcomp :=     for $ingr in $c/Component
                            order by $ingr/Quantity, $ingr/IngredientName
                            return $ingr

        let $cals := for $i in $ordcomp
                         return $i/Quantity * //Ingredient[
                                                      Name =  $i/IngredientName
                                                      ]/CaloriesPerGram

        return <Cocktail name="{ $c/Name/text() }">
                    <Calories> { sum( $cals ) } </Calories>
                    <Top4> { $ordcomp[position() <= 4]/IngredientName } </Top4>
               </Cocktail >
} </Summary>
```

D. Olympic games

```
<!ELEMENT Collection (Country*) >
<!ELEMENT Country (Gold?, Silver?, Bronze?) >
<!ATTLIST Country      Name CDATA #REQUIRED >

<!ELEMENT Gold (Medal+)>
<!ELEMENT Silver (Medal+)>
<!ELEMENT Bronze (Medal+)>

<!ELEMENT Medal (Athlete+)>
<!ATTLIST Medal        Discipline CDATA #REQUIRED
                       Date CDATA #REQUIRED >
<!ELEMENT Athlete (#PCDATA)>
```

N.B. One single edition

Medals can be won by individuals or by teams, the names of athletes winning more than one medal are repeated in the document

D. Olympic games

1. Find countries that never won a team medal (a team medal is a medal won by more than one athlete)
2. Find the name of the athlete who won the highest number of medals, considering all possible disciplines and all kinds of medal.

D. Olympic games

1. Find countries that never won a team medal (a team medal is a medal won by more than one athlete)

```
for $n in //Country
where every $m in $n/*/Medal satisfies count( $m/Athlete ) = 1
return <OnlySingle>
              { $n/@name }
          </OnlySingle>

//Country[ count( ./*/Medal[count(./Athlete)>1] ) = 0 ]/@Name
```

D. Olympic games

2.a Find the name of the athlete who won the highest number of medals, considering all possible disciplines and all kinds of medal.

```
let $athletes := distinct-values( //Athlete/text() )
let $max := max ( for $at in $athletes
                  return count( //Medal[ Athlete/text() = $at ] )
for $a in $athletes
let $med := count( //Medal[ Athlete/text() = $a ] )
where $med = $max
return <MostMedals> { $a } </MostMedals >
```

D. Olympic games

2.b (same, but unfair in case of ties...)

```
let $list :=      for $at in distinct-values( //Athlete/text() )
                  let $nu := count( //Medal[ Athlete/text() = $at ] )
                  order by $nu
                  return <item>
                                <name> { $at } </name>
                                <num> { $nu ) </num>
                         </item>
return <champion> { $list[1] /name/text() } </champion>
```

D. Olympic games

2.c (in order to use the ranking in a fair way...)

```
let $list :=      for $at in distinct-values( //Athlete/text() )
                  let $nu := count( //Medal[ Athlete/text() = $at ] )
                  order by $nu
                  return <item>
                             <name> { $at } </name>
                             <num> { $nu ) </num>
                         </item>
for $a in $list
where $a / num = $list[1] / num
return <champion> { $a/name/text() } </champion>
```

E. Hierarchy of Components

A shop sells electronic components, described in document catalog.xml. For each component the cost is represented and, for complex components, the internal structure:

```
<!ELEMENT Catalog (Component+)>
<!ELEMENT Component (Description, Component*)>
<!ATTLIST Component     Code ID #REQUIRED,
                        Cost CDATA #REQUIRED>
```

Where Description only contains PCDATA.

E. Hierarchy of Components

1. Extract the component that has the greatest number of sub-components (direct and indirect)
2. Build in XQuery an XML document that shows, for each leaf component c, the list of the codes of the components that contain c, ordered from the "lower" one to the "higher" one in the containment hierarchy

E. Hierarchy of Components

1. Extract the component that has the greatest number of sub-components (direct and indirect)

```
let $max := max(   for $c in doc("comps.xml")/Catalog/Component
                   return count( $c//Component ) )
return doc("comps.xml")/Catalog/Component[ count( .//Component ) = $max ]
```

E. Hierarchy of Components

2. Path problem

```
declare function local:reversepath($lc as element(), $root as element()) as element()*
{
      for $c in $root/Component
      where $lc/@code = $c//Component/@code
      return ( local:reversepath($lc, $c) , <parent code="{ $c/@code }"/> )
};
<Hierarchies> {
      for $lc in doc("comps.xml")//Component
      where count($lc/Component) = 0
      return <LeafComponent code="{ $lc/@code }">
                  { local:reversepath($lc, doc("comps.xml")/Catalog) }
             </LeafComponent>
}</Hierarchies>
```

E. Hierarchy of Components

2. (another approach v1...)

```
let $leaves := doc("comps.xml")//Component[ count(./Component) = 0 ]
for $f in $leaves
      let $fathers := (
                        for $p in doc("comps.xml")//Component[ .//@code = $f/@code ]
                        let $numsucomp := count( $p//Component )
                        where $numsucomp > 0
                        order by $numsucomp
                        return <Container code="{ $p/@code }"/> )
return <LeafComponent code="{ $f/@code }">
            { $fathers }
         </LeafComponent>
```

E. Hierarchy of Components

2. (another approach v2...)

```
for $l in doc("comps.xml")//Component[ count(./Component) = 0 ]
return <LeafComponent code="{ $l/@code }">
            {
            reverse( for $c in doc("comps.xml")//Component[
                                                                   .//@code = $l/@code and
                                                                   count(./Component) > 0 ]
                         return data($c/@code)
                       )
             }
        </LeafComponent>
```

F. Sports Club

Consider the following DTD about the organization of a sports club. Elements not further specified contain only PCDATA. The team names and social security numbers (SSN) are unique.

```
<!ELEMENT Club ( Coach+, Team+ )>
<!ELEMENT Coach ( SSN, Name, Lastname )>
<!ELEMENT Team ( Name, CoachSSN, Player+ )>
<!ELEMENT Player ( SSN, Name, Lastname, Role )>
```

F. Sports Club

1. Write in XQuery the query that extracts the names of the coaches who train three or more teams and who also appear as player in some other team (i.e. not in a team coached by them)
2. Write in XQuery the query that for each coach retrieves the coach's data and the complete list in alphabetical order of all the players he coached (considering all the teams he trains)

F. Sports Club

1. Write in XQuery the query that extracts the names of the coaches who train three or more teams and who also appear as player in some other team (i.e. not in a team coached by them)

```
<BusyCoaches> {
    for $c in //Coach
    where   3 <= count( //Team[CoachSSN = $c/SSN] )
            and 0 < count( //Team[CoachSSN != $c/SSN]/Player[SSN = $c/SSN] )
    return $c/Lastname
} </BusyCoaches >
```

F. Sports Club

2. Write in XQuery the query that for each coach retrieves the coach's data and the complete list in alphabetical order of all the players he coached (considering all the teams he trains)

```
<CoachingLists> {
    for $c in //Coach
    return    <Coach>
                  { $c/*}
                  <CoachedPlayers> {
                      for $p in //Team[CoachSSN = $c/SSN]/Player
                      order by $p/Lastname, $p/Name
                      return $p
                  } </CoachedPlayers>
              </Coach>
} </CoachingLists >
```

F. Sports Club

2. (what if the same players play in more than one team coached by the same coach?)

```
<CoachingLists> {
    for $c in //Coach
    return <Coach> { $c/*}
            <CoachedPlayers> {
                let $HisPlayers := distinct-values(
                                    //Team[CoachSSN = $c/SSN]/Player/SSN )
                let $list := (  for $ssn in $HisPlayers
                                let $plrs := //Player[SSN = $ssn]
                                for $p in $plrs[1]
                                order by $p/Lastname, $p/Name
                                return $p)
                return $list
            } </CoachedPlayers> </Coach>
} </CoachingLists >
```