Données du Web : TP noté

XPath/XQuery

Alexandre Canton Condes

1) XQuery : Tweets

1. Indiquer le nombre de tweets et d'utilisateurs dans la base.

```
let $t := count(/descendant::Tweet)
let $a := count(/descendant::Author)
return ($t, $a)
```

2. Donner l'ensemble des hashtags contenus dans la base.

```
for $c in /descendant::Tweet/descendant::Content
return $c/Hashtag
```

3. Créer une liste de paires tweet-auteur, avec chaque paire contenue dans un élément result.

```
for $t in /descendant::Tweet
let $idrefa := $t/@idref_author
for $a in /descendant::Author
let $ida := $a/@id_author
where $idrefa = $ida
return
<result>{($t, $a)}</result>
```

4. Pour chaque utilisateur, lister le nom de l'utilisateur et la date de tous ses tweets, le tout regroupé dans un élément result.

```
for $a in /descendant::Author
let $a1 := $a/@id_author
let $a2 := $a/Name/text()
for $t in /descendant::Tweet
let $t1 := $t/@idref_author
where $a1 = $t1
return              <result>{($a2,              $t/@seconds/string(),
$t/@timer_zone/string())}</result>
```

5. Lister les utilisateurs qui ont publié un tweet qui a été retwitté.

```
for $a in /descendant::Author
```

```
where /descendant::Retweets > 0
return $a
```

6. Pour chaque tweet, indiquer la date de ses deux premières réponses. Rajouter un element vide <nonRetwitted/> s'il n'a pas été retwitté.

```
for $t in /descendant::Tweet
let $sort :=
  for $t1 in /descendant::Tweet[Content/Retweet/text() > 0]
    order by $t1/@seconds
    return $t1
  return
    if ($sort) then
      <result>($sort[1]/@seconds, $sort[2]/@seconds)</result>
    else
      <result><nonRetwitted></nonRetwitted></result>
```

7. Lister les utilisateurs de la plateforme en ordre alphabétique.

```
for $a in /descendant::Author
order by $a/Name
return $a
```

8. Lister les tweets contenants l'hashtag "#I<3XML".

```
for $t in /descendant::Tweet
where $t/Content/Hashtag/string() = "#I<3XML"
return $t
```

9. Trouvez le tweet le plus ancien ainsi que le plus recent.

```
let $t := /descendant::Tweet
let $recent := min($t/@seconds)
let $ancien := max($t/@seconds)
return ($t[@seconds = $recent], $t[@seconds = $ancien])
```

10. Pour chaque utilisateur, indiquer l'ensemble des hashtags qu'il a utilisés dans ses Tweets.

```
for $a in //Author
for $t in //Tweet
where $a/@id_author = $t/@idref_author and $t//Hashtag
return ($a/@id_author, $t//Hashtag)
```

11. Pour chaque tweet ayant des références utilisateur, retournez le tweet avec la liste des références utilisateur.

```
for $t in //Tweet
where $t//UserReference
return ($t/@id_tweet, $t//UserReference)
```

12. Déclarez la fonction local:aReponduAuTweet, qui, étant donné un tweet, retourne tous les utilisateurs qui ont répondu au Tweet.

```
declare function local:aReponduAuTweet($t, $r) {
  for $t1 in $r//Tweet
  where $t/@idref_author = $t1//UserReference
  return ($t1//UserReference)
};
local:aReponduAuTweet(//Tweet[1], /)
```

2) Génération de Pages HTML via XQuery

```
let $vcs := doc("https://data.montpellier3m.fr/sites/default/files/ressources/TAM_MMM_VELOMAG.xml")

let $sorted_alpha :=
  for $station in $vcs//si
  let $name := substring($station/@na, 5)
  order by $name
  return $station

let $sorted_capa :=
  for $station in $vcs//si
  order by xs:int($station/@to)
  return $station

let $low :=
  for $station in $vcs//si
  let $dispo := $station/@fr div $station/@to
  where $dispo < 0.3
  order by xs:decimal($dispo)
  return $station

let $medium :=
  for $station in $vcs//si
  let $dispo := $station/@fr div $station/@to
  where $dispo >= 0.3 and $dispo < 0.6
  order by xs:decimal($dispo)
  return $station

let $high :=
  for $station in $vcs//si
```

```
  let $dispo := $station/@fr div $station/@to
  where $dispo > 0.6
  order by xs:decimal($dispo)
  return $station

let $alpha :=
  <table>
    <tr>
      <th>Nom</th>
      <th>ID</th>
      <th>Latitude</th>
      <th>Longitude</th>
      <th>AV</th>
      <th>FR</th>
      <th>TO</th>
    </tr>
    {for $station in $sorted_alpha
    return
    <tr>
      <td>{substring($station/@na, 5)}</td>
      <td>{xs:int($station/@id)}</td>
      <td>{xs:decimal($station/@la)}</td>
      <td>{xs:decimal($station/@lg)}</td>
      <td>{xs:int($station/@av)}</td>
      <td>{xs:int($station/@fr)}</td>
      <td>{xs:int($station/@to)}</td>
    </tr>}
  </table>

let $capa :=
  <table>
    <tr>
      <th>Nom</th>
      <th>ID</th>
      <th>Latitude</th>
      <th>Longitude</th>
      <th>AV</th>
      <th>FR</th>
      <th>TO</th>
    </tr>
    {for $station in $sorted_capa
    return
    <tr>
      <td>{substring($station/@na, 5)}</td>
```

```
      <td>{xs:int($station/@id)}</td>
      <td>{xs:decimal($station/@la)}</td>
      <td>{xs:decimal($station/@lg)}</td>
      <td>{xs:int($station/@av)}</td>
      <td>{xs:int($station/@fr)}</td>
      <td>{xs:int($station/@to)}</td>
    </tr>}
  </table>

let $dispo_low :=
  <table>
    <tr>
      <th>Nom</th>
      <th>ID</th>
      <th>Latitude</th>
      <th>Longitude</th>
      <th>AV</th>
      <th>FR</th>
      <th>TO</th>
    </tr>
    {for $station in $low
    return
    <tr>
      <td>{substring($station/@na, 5)}</td>
      <td>{xs:int($station/@id)}</td>
      <td>{xs:decimal($station/@la)}</td>
      <td>{xs:decimal($station/@lg)}</td>
      <td>{xs:int($station/@av)}</td>
      <td>{xs:int($station/@fr)}</td>
      <td>{xs:int($station/@to)}</td>
    </tr>}
  </table>

let $dispo_medium :=
  <table>
    <tr>
      <th>Nom</th>
      <th>ID</th>
      <th>Latitude</th>
      <th>Longitude</th>
      <th>AV</th>
      <th>FR</th>
      <th>TO</th>
    </tr>
```

```
    {for $station in $medium
    return
    <tr>
      <td>{substring($station/@na, 5)}</td>
      <td>{xs:int($station/@id)}</td>
      <td>{xs:decimal($station/@la)}</td>
      <td>{xs:decimal($station/@lg)}</td>
      <td>{xs:int($station/@av)}</td>
      <td>{xs:int($station/@fr)}</td>
      <td>{xs:int($station/@to)}</td>
    </tr>}
  </table>

let $dispo_high :=
  <table>
    <tr>
      <th>Nom</th>
      <th>ID</th>
      <th>Latitude</th>
      <th>Longitude</th>
      <th>AV</th>
      <th>FR</th>
      <th>TO</th>
    </tr>
    {for $station in $high
    return
    <tr>
      <td>{substring($station/@na, 5)}</td>
      <td>{xs:int($station/@id)}</td>
      <td>{xs:decimal($station/@la)}</td>
      <td>{xs:decimal($station/@lg)}</td>
      <td>{xs:int($station/@av)}</td>
      <td>{xs:int($station/@fr)}</td>
      <td>{xs:int($station/@to)}</td>
    </tr>}
  </table>


return
<div>
  <h2>Ordre alphabétique</h2>
  {$alpha}
  <h2>Capacité</h2>
  {$capa}
```

```
  <h2>Niveau de disponibilité</h2>
  <h3>Faible</h3>
  {$dispo_low}
  <h3>Moyen</h3>
  {$dispo_medium}
  <h3>Haut</h3>
  {$dispo_high}
</div>
```

3) Propriétés des requêtes XPath

1. Reformuler les requêtes suivantes en utilisant exclusivement les axes child, descendant, descendant-or-self, following et following-sibling.

```
- //d/preceding-sibling::c
/descendant-or-self::*/child::c[following-sibling::d]

- //c/a/preceding-sibling::a/preceding::e
/descendant-or-self::*/child::c/child::e[following-sibling:a/followin
g-sibling::a]]

- //d[parent::b/c]
/descendant-or-self::*/child::b[child::c]/child::d

- /r/b/..//*/./../preceding::d
...

- //a/ancestor::c/child::d/parent::e
/descendant-or-self::*/child::c[descendant::a]/child::e[descendant::d
]

- //c[preceding::d]
/descendant-or-self::*/child::d/following::c
```

2. Reformuler les requêtes //a/following::b et //a/preceding::b en utilisant les axes descendant-or-self,
ancestor, following-sibling et preceding-sibling.

```
- //a/following::b
/descendant-or-self/child::b[preceding-sibling::a]

- //a/preceding::b
/descendant-or-self/child::b[following-sibling::a]
```

3. Pour chaque requête définie aux points 1 et 2, proposer un document XML pour lequel la réponse à la
requête n'est pas vide, sinon expliquer pourquoi un tel document n'existe pas.

- `//d/preceding-sibling::c`

```
<a>
    <c/>
    <d/>
</a>
```

- `//c/a/preceding-sibling::a/preceding::e`

```
<c>
    <e/>
    <a/>
    <a/>
</c>
```

- `//d[parent::b/c]`

```
<b>
    <d/>
    <c/>
</b>
```

- `/r/b/..//*/./../preceding::d`

Appeler «preceding» alors qu'on est à la racine ne renvoie aucun résultat.

- `//a/ancestor::c/child::d/parent::e`

On ne peut pas appeler «parent::e» car «d» a déjà «c» comme parent.

- `//c[preceding::d]`

```
<a>
    <d/>
    <c/>
</a>
```

- `//a/following::b`

```
<c>
    <a/>
    <b/>
</b>
```

- `//a/preceding::b`

```
<c>
```

```
    <b/>
    <a/>
</c>
```

4) L'égalité dans XQuery

1. Soient X,Y, Z des séquence d'éléments XML. Est il vrai que, dans le cadre du langage XPath, si X = Y et
Y = Z alors X = Z ? Est-ce le cas pour XQuery ?

-> Dans XPath :

L'égalité entre deux nœuds est vrai si la comparaison des valeurs de texte des deux nœuds est vrai.

Soit X le nœud possédant comme valeur texte «RiriFifi».
Soit Y le nœud possédant comme valeur texte «FifiLoulou».
Soit Z le nœud possédant comme valeur texte «Loulou».

X = Y vrai, car ils contiennent tout les deux le texte «Fifi».
Y = Z vrai, car ils contiennent tout les deux le texte «Loulou».
X = Z faux, car ils n'ont aucun texte en commun.

-> Dans XQuery :

Avec `eq` :
Si le type de X et Y est simple (`xs:string`, `xs:int`, etc...), alors `X eq Y` est vrai ssi leur valeurs sont égales.
Si le type de X et Y est complexe (défini par l'utilisateur), alors `X eq Y` est vrai ssi `xs:string(X) eq xs:string(Y)`.

Avec `=` :
Lorsque les opérandes X et Y sont des ensembles.
`X = Y` est vrai ssi il existe au moins un élément `x` dans `X` et un élément `y` dans `Y` tels que `x eq y`

Donc comme avec XPath, il suffit que les deux nœuds ai un élément texte en commun pour qu'ils soient égaux.
Mais l'égalité n'est pas transitive.

2. Donner une fonction XQuery qui renvoie vrai si et seulement si deux séquences sont identiques. Pour

simplifier, nous ne considérons pas les attributs (mais nous considérerons bien l'ordre des éléments).

```
declare function local:myEqual(
    $seq1 as item()*,
    $seq2 as item()*)
    as xs:boolean {
  every $i in 1 to max((count($seq1), count($seq2)))
  satisfies deep-equal($seq1[$i], $seq2[$i])
};

let                                $vcs                                :=
doc("https://data.montpellier3m.fr/sites/default/files/ressources/TAM
_MMM_VELOMAG.xml")
return local:myEqual($vcs/vcs/sl/*, $vcs/vcs/*/*)
```