# 1. Introduction

Commencez par créer un dossier `tp5` pour ce TP et travaillez dedans. Vous y mettrez les réponses aux questions.

Info: le TP est prévu pour durer 2 semaines. C'est le TP le plus important de ce module.

# 2. BaseX et XQuery

BaseX est un gestionnaire de bases de données XML. Il gère des données nativement représentées en XML. Les requêtes s'écrivent en XQuery. Relire le cours sur XQuery, en résumé, c'est l'intégration des requêtes XPath dans des structures de contrôle plus complexes.

- Télécharger le document `arbres.xml`.
- Lancer le logiciel `BaseX` (menu Applications, Bases de données).

**IMPORTANT** : la version de BaseX qui est installée actuellement par la distribution Debian est très buggée. Voir Version 9.0.1 critical bug – il reste d'autres bugs non corrigés. Il faut télécharger la dernière version de BaseX : Version 9.6.4. Cliquez sur BaseX964.jar (4Mo). Il suffit de double cliquer le jar obtenu pour lancer BaseX.

Avant tout, allez dans le menu Options, Préférences et choisissez le langage `French`. Redémarrez BaseX.

On va commencer par créer une base de données sur le fichier `arbres.xml`. Sélectionnez le menu `Base de données`, item `Nouveau...`. Désignez le fichier `arbres.xml` puis faites ok.

L'écran est divisé en plusieurs fenêtres. Au début, il y a :

- une zone éditeur avec une vue des fichiers,
- une zone pour les résultats des requêtes.

La vue éditeur permet de saisir des programmes XQuery. Tapez `(: TP5 sur BaseX :)`. Cette ligne bizarrement souriante est un commentaire dans le langage XQuery. Dans le menu `Éditeur`, choisissez `Enregistrer sous...`, sauvez-le sous le nom `tp5_arbres.xq`.

La vue éditeur s'est réduite et maintenant il y a une zone d'affichage du fichier de manière graphique. C'est le menu `Visualisation` qui gère ça. Deux autres visualisations peuvent être utiles : `dossier` et `tableau`.

Essayez la visualisation `Diagramme`. Choisissez `arbre` dans le combo du milieu, puis `@lat` dans celui de gauche et `@lon` dans celui de droite. Si on avait une carte de Paris à superposer (géolocalisée), on verrait la position des arbres dans la ville.

## 2.1. Requêtes XPath simples

Essayer de comprendre les requêtes suivantes. Si elle ne vous paraissent pas claires, tapez-les dans l'éditeur de texte. Lancez l'exécution avec le bouton flèche verte dans la barre d'outils de l'éditeur. Une fois qu'une requête a été saisie et essayée, mettez-la simplement en commentaire avant de taper la suivante puis enregistrez le fichier avec `CTRL S`. Un commentaire XQuery s'écrit entre `(:` et `:)`.

Vous pouvez aussi taper les requêtes dans la barre du haut, mais alors mettez-la en mode `XQuery`.

Voici quelques exemples, un par ligne à essayer :

```
//lieu/nom
//arbre[genre="Quercus" and hauteur>=30 and lieu//@arrondissement=12]
string(//arbre[@id=8]//@arrondissement)
//arbre[starts-with(nom_commun, "Chene")]/(espece|nom_commun)
distinct-values(//genre)
min(//arbre/annee)
```

Vous avez remarqué qu'on travaille avec XPath 2.0, puisqu'on peut utiliser des fonctions telles que `distinct-values`, `min`, `avg`... La liste complète est sur cette page.

## 2.2. Requêtes XQuery complexes

XQuery permet d'effectuer un travail similaire à XSLT : générer des données XML à partir d'un document. Voici différentes requêtes à essayer successivement, chacune étant à mettre en commentaires avant d'essayer la suivante, ou alors effacez tout avant d'essayer la suivante.

- Exemple 1

```
let $grands := //arbre[hauteur > 40.0]
return <geants>{$grands}</geants>
```

Cette requête affecte une variable avec une expression XPath, puis construit une valeur de retour sous forme d'un élément XML. Dans le `return`, les variables XQuery doivent être placées entre {}.

- Exemple 2

```
let $nombre := count( /arbres/arbre )
let $moyenne := avg( //hauteur )
return
<html>
    <body lang="fr">Il y a {$nombre} arbres, de {$moyenne}m de haut en moyenne</body>
</html>
```

XQuery permet de générer du XML quelconque, dont le HTML. Plusieurs affectations peuvent être écrites successivement. Attention à ne pas mettre de `;` entre elles.

- Exemple 3

```
element html {
    let $nombre := count( //arbre )
    let $moyenne := avg( //hauteur )
    return
        element body {
            attribute lang { "fr" },
            "Il y a", $nombre, "arbres, de",
            concat($moyenne, "m de haut en moyenne")
        }
}
```

C'est une variante syntaxique du précédent exemple. La génération des éléments ressemble à la syntaxe compacte imbriquée RelaxNG, à part les jokers. On peut imbriquer des éléments et des instructions. Également, le document est explicitement mentionné. On peut travailler avec plusieurs documents.

- Exemple 4

```
for $arbre in //arbre
let $arrond := $arbre//@arrondissement
order by number($arrond)
return element arbre { $arrond }
```

XQuery permet de faire des boucles pour générer plusieurs éléments et on peut placer des affectations dans le corps. Regardez la clause `order by` : on est obligé de convertir l'arrondissement en nombre, sinon le classement serait dans l'ordre alphabétique des chiffres du numéro.

- Exemple 5

```
for $genre in distinct-values(//genre)
let $nombre := count(//arbre[genre=$genre])
return <type genre="{$genre}">{$nombre}</type>
```

`distinct-values` retourne la liste des valeurs (textes) distincts trouvés dans les éléments sélectionnés par son paramètre.

- Exemple 6

```
max(for $genre in distinct-values(//genre) return count(//arbre[genre=$genre]))
```

Les fonctions et les boucles peuvent être combinées. En fait, ce ne sont que des expressions et non pas des instructions comme dans d'autres langages.

Ici, le `for` passe les différentes valeurs des éléments `<genre>` en revue, pour chacun, le `return` affiche le nombre d'arbres de ce genre. Le tout étant dans une fonction `max`, c'est seulement la plus grande valeur qui est produite. L'un des genres d'arbres possède 17 représentants (on ne sait pas lequel, voir l'exemple 10).

- Exemple 7

```
<nombres>{
    for $arr in distinct-values(//lieu//@arrondissement)
    let $nombre := count(//arbre[lieu//@arrondissement=$arr])
    order by $nombre descending
    let $arbres := if ($nombre>1) then " arbres" else " arbre"
    return element nombre { concat($arr,"e arrondissement : ", $nombre, $arbres) }
}</nombres>
```

La structure *For-Let-Order-Return* est appelée FLWOR. Pour le W, voir l'exemple suivant. Cette structure de contrôle donne sa puissance à XQuery : la possibilité de faire de boucles complexes bien plus simplement qu'en XSLT.

Attention, il y a un piège avec la conditionnelle : `if` est une expression et non pas une structure de contrôle. Pour exécuter une instruction sous contrôle d'un test, il faut utiliser `where`.

- Exemple 8

```
element nombres {
    for $arr in distinct-values(//lieu//@arrondissement)
    let $nombre := count(//arbre[lieu//@arrondissement=$arr])
    where $nombre > 10
    order by $nombre descending
    return element nombre {concat($arr,"e arrondissement : ", $nombre, " arbres")}
}
```

Si on veut utiliser une conditionnelle en tant que structure de contrôle, alors utiliser `where` et non pas `if`, et il n'y a pas de possibilité de `else`.

Ces instructions, For Let Where Order Return sont désignées par l'acronyme *FLWOR*.

- Exemple 9

```
let $max := max(for $arbre1 in //arbre return $arbre1/hauteur)
for $arbre in //arbre
where $arbre/hauteur = $max
return $arbre
```

Cette requête XQuery affiche l'arbre le plus haut. On peut également employer une requête XPath2 :

```
//arbre[hauteur = max(//arbre/hauteur)]
```

Les deux font la même chose. À vous de voir laquelle vous préférez. Il n'est pas toujours possible d'écrire des requêtes en pur XPath ; c'est le cas de l'exemple suivant.

- Exemple 10

```
let $max := max(for $genre in distinct-values(//genre)
                return count(//arbre[genre=$genre]))
for $genre in distinct-values(//genre)
let $nombre := count(//arbre[genre=$genre])
where $nombre = $max
return <max genre="{$genre}">{$max}</max>
```

Cette requête assez complexe affiche le genre d'arbre le plus représenté dans le document. Le `let` recherche le genre le plus présent, voir l'exemple 6. Le `for` parcourt les valeurs différentes des éléments `<genre>`. Le 2e `let` à l'intérieur du `for` compte les arbres de *ce* genre-là. Le `where` sélectionne les genres dont le nombre d'arbres est égal au maximum, et le `return` génère un élément pour afficher les deux informations.

- Exemple 11

```
for $siecle in distinct-values(//(annee idiv 100 + 1))
order by $siecle ascending
return element siecle {
    attribute nom { concat($siecle,"e") },
```

```
    for $arbre in //arbre
    where $arbre/annee idiv 100 + 1 = $siecle
    order by $arbre/annee
    return element arbre {
        attribute numero {$arbre/@id},
        attribute annee {$arbre/annee}
    }
}
```

C'est une double boucle. La boucle externe parcourt les années divisées par 100 + 1 (`idiv` fait une division entière, tandis que `div` fait la division entre réels). Alors regardez bien la manière dont la fonction `distinct-values` a été appelée : ce n'est pas `distinct-values(//annee idiv 100 + 1)` car ça ne marche pas, mais `distinct-values(//(annee idiv 100 + 1))` qui semble moins naturel. La raison est que `//annee` représente une collection, on ne peut pas lui appliquer une division, malheureusement XQuery n'a pas cette notion (une sorte de *map*).

La boucle interne parcourt les arbres et les sélectionne en fonction de leur année : elle doit correspondre au siècle de la boucle externe. Pour finir, il y a un classement des arbres selon leur année.

En fait, il y a une erreur avec le calcul des siècles : l'année 1900 est encore au 19$^{e}$ siècle et non pas au 20$^{e}$. Le 20$^{e}$ siècle commence en 1901. Comment faire pour corriger le problème ? Il faut écrire `((annee - 1) idiv 100 + 1)` à la fois dans le `for` et dans le `where` (en englobant `$arbre/annee`). Attention, il faut impérativement un espace entre `annee` et `- 1`.

## 2.3. Requêtes en modification

XQuery permet de modifier la base de données XML à l'aide de différentes requêtes. Alors il faut comprendre une chose concernant BaseX : le fichier XML fourni pour créer la base n'est pas concerné par les modifications. Elles ne sont appliquées que sur une copie gérée en interne par BaseX. Cette copie se trouve dans `~/BaseXData` mais est dans une forme binaire non lisible. Lorsque vous faites des changements, c'est comme avec une base SQL, ils sont définitifs. Si vous voulez retrouver l'état initial, vous devrez re-créer la base en écrasant l'existante.

- Exemple 12

```
insert node <visite>oui</visite> into //arbre[@id=6]/lieu
```

On doit donner l'emplacement exact du nouvel élément. Il ne peut y avoir qu'un seul élément désigné par l'expression XPath.

- Exemple 13

```
insert node <visite>oui</visite> into //arbre/lieu
```

Dans cet exemple, l'expression XPath désigne plusieurs emplacements possibles. Cela cause une erreur. Pour modifier plusieurs éléments, il faut faire une boucle :

```
for $lieu in //arbre/lieu
return insert node <visite>oui</visite> into $lieu
```

- Exemple 14

```
delete node //arbre[@id=6]/lieu/visite
```

BaseX n'affiche rien dans la fenêtre des résultats. Il suffit de cliquer sur l'icône en forme de maison pour voir l'état de la base XML.

- Exemple 15

```
delete node //arbre/lieu
```

Voila une action particulièrement destructrice. L'expression a sélectionné beaucoup d'éléments de la base et les a tous supprimés sans demander confirmation.

Re-créez la base pour continuer : `CTRL N` puis `Ok`.

- Exemple 16

```
replace value of node //arbre[@id=6]/genre with 'Macintosh'
```

Cette action remplace la valeur (le contenu texte) d'un élément par une autre chaîne.

Re-créez la base pour continuer.

- Exemple 17

```
let $genre := //arbre[@id=6]/genre/text()
return replace node //arbre[@id=6]/genre with <type>{$genre}</type>
```

Si on veut utiliser des variables pour se faciliter la vie, il faut inclure le `replace` dans une expression FLWOR, placer le `replace` dans la clause `return`, bien qu'elle ne retourne rien.

Ici, il faut faire attention à bien mettre `text()` pour récupérer le contenu de l'élément `<genre>`. Regardez bien ce qui a été remplacé et par quoi.

Si on veut faire le remplacement sur chaque élément, il faut employer une boucle :

```
for $genre in //genre
return replace node $genre with <type>{$genre/text()}</type>
```

- Exemple 18

```
let $arbre := //arbre[@id=6]
return (
    replace node $arbre/genre with <type>{$arbre/genre/text()}</type>,
    delete node $arbre/espece,
    delete node $arbre/famille,
    insert node element rare {} into $arbre
)
```

Pour effectuer plusieurs changements sur le même élément, on les regroupe dans un `return` avec des parenthèses.

# 3. Exercices XQuery sur `factbook.xml`

Télécharger le document `factbook.xml` et sa DTD `factbook.dtd`. C'est un document complexe et ancien qui provient de la CIA et qui regroupe des informations géographiques, politiques et éthniques sur les pays et villes du monde. Malheureusement, ce fichier est très incomplet et parfois incohérent.

Passez quelques minutes à étudier sa structure, éventuellement en regardant sa DTD. En particulier, notez l'emploi d'identifiants XML avec les types `ID`, `IDREF` et `IDREFS`. `ID` est le type d'une valeur qui doit être unique parmi les autres `ID` du document. `IDREF` désigne une valeur qui doit être l'un des `ID` du fichier. `IDREFS` désigne une liste de valeurs `IDREF` séparées par des espaces. Par exemple, chaque pays, chaque ville est identifiée par un code unique, de type `ID`. La capitale d'un pays est indiquée par l'attribut `capital` de type `IDREF` qui désigne la ville concernée.

- Faites-en une base de données XML.
- Téléchargez `ReponsesTP5a.xq`, vos réponses seront à écrire dedans. Ce fichier sera à déposer sur Moodle.
- Quand vous travaillez sur une réponse, `(:`commentez les autres`:)`

**IMPORTANT** : ne changez ni le nom, ni la structure du fichier `ReponsesTP5a.xq`. Vous devez seulement placer vos réponses en dessous des noms des questions, avec une ligne vide entre chaque question. Ne laissez pas le message « Mettez votre réponse ici » si vous répondez à la question, mettez votre réponse exactement à sa place. Laissez les commentaires `(:` et `:)` seuls sur leur ligne autour de votre réponse. Ne mettez aucune ligne vide dans le bloc de votre réponse. Si vous proposez plusieurs réponses, alors faites un bloc entre `(:` et `:)` pour chaque, sans mettre de ligne vide entre eux. Voici un exemple correct selon ces consignes :

```
(: Exercice 3.1 : Hauteur du Mont-Blanc :)
(:
string(//@elevation[mountain/@name="Mont-Blanc"])
:)
(:
//@elevation[mountain/@name="Mont-Blanc"]
:)

(: Exercice 3.2 : Capitale du Vatican :)
(:
for $city:=//city[name="Vatican"]
    return capital[.=$city]
:)
```

Comme toujours, faites ce que vous pouvez comme vous pouvez. Pensez à sauvegarder très régulièrement.

## 3.1. Longueur de la Tamise

Écrire une requête XPath qui affiche la longueur de la Tamise. Il faut afficher l'attribut `length` de l'élément `<river>` ayant l'attribut `name` valant « Thames ».

## 3.2. Capitale du Mali

Écrire une requête XPath qui affiche le nom de la capitale du Mali.

Voici comment faire cette requête :

- D'abord chercher l'élément `<country>` correspondant au Mali.
- Obtenir l'identifiant de la capitale : attribut `capital`
- Obtenir l'élément `<city>` ayant cet identifiant puis afficher l'élément `name` ou son contenu.

## 3.3. Îles du Japon

Écrire une requête XQuery qui affiche les noms des îles du Japon. Il y en a 4 dans ce fichier.

Voici comment faire cette requête :

- D'abord, il faut l'identifiant de l'élément `<country name="Japan">`
- Il faut trouver tous les éléments `<island>` contenant un sous-élément `<located country="id_japon"/>`
- Pour finir, il faut afficher le nom de ces îles, soit le nom seul, soit avec l'attribut `name`.

## 3.4. Nom de la mer la plus profonde

Écrire une requête XQuery qui affiche le nom de la mer la plus profonde.

Il y a sûrement plusieurs façons de faire. Vous pouvez d'abord rechercher la plus grande valeur pour l'attribut `depth` des éléments `<sea>`. Ensuite, cherchez quel est celui qui possède cette valeur et affichez son attribut ou sous-élément `name`.

## 3.5. Noms des pays qui n'appartiennent à aucune organisation

Écrire une requête XQuery qui affiche les noms des pays qui n'appartiennent à aucune organisation. Ce sont des territoires qui dépendent en réalité d'un autre pays, comme Mayotte et Gibraltar.

Voici comment faire cette requête :

- On s'intéresse aux éléments `<organization>`. Ils représentent les organisations internationales, comme l'ONU. Leur sous-élément `<member>` donne les identifiants des pays qui en font partie. Le but est de trouver les pays qui ne sont dans *aucun* de ces sous-éléments.
- Parcourir les pays et ne garder que ceux dont on ne trouve pas l'identifiant en tant que `<member>` de l'une des organisations.
- Afficher le nom en tant qu'attribut ou seulement le nom en tant que chaîne.

## 3.6. Classement des pays par mortalité infantile

Écrire une requête XQuery qui affiche le classement des pays par mortalité infantile croissante. Le résultat attendu est :

```
<classement>
  <pays name="Andorra">2.2</pays>
  <pays name="Man">2.4</pays>
  <pays name="Jersey">2.7</pays>
  <pays name="Iceland">4.3</pays>
  <pays name="Japan">4.4</pays>
  ...
</classement>
```

Voici comment faire cette requête :

- On s'intéresse à l'attribut `infant_mortality` des éléments `<country>`. Il ne faut pas traiter les pays n'ont pas renseigné cet attribut.
- Il faut classer les pays dans l'ordre croissant de la mortalité. Il y a un piège : si on classe simplement, ça sera dans l'ordre alphabétique des chiffres. Il faut trier sur la valeur numérique : `order by number(...infant_mortality)`.

## 3.7. Noms des pays bordés par au moins 3 océans ou mers

Écrire une requête XQuery qui affiche les pays qui sont bordés par au moins trois océans (ou mers) – à considérer tels que dans ce fichier qui est très incomplet. Il faut générer un élément `<pays>` avec le nom du pays en attribut, et des sous-éléments `<ocean>` dont le contenu est le nom de l'océan concerné. Le résultat attendu est :

```
<maritimes>
  <pays name="Russia">
    <ocean>Arctic Ocean</ocean>
    <ocean>Pacific Ocean</ocean>
    <ocean>Sea of Japan</ocean>
  </pays>
  <pays name="Cuba">
    <ocean>Atlantic Ocean</ocean>
    <ocean>Gulf of Mexico</ocean>
    <ocean>Caribbean Sea</ocean>
  </pays>
</maritimes>
```

Voici comment faire cette requête :

- L'information se trouve dans les éléments `<sea>`. Il y a un sous-élément `<located>` qui indique par son attribut `country` au bord de quel pays il se trouve.
- La méthode consiste à parcourir les pays, et pour chacun compter le nombre d'éléments `<sea>` qui mentionnent ce pays. Attention car le même pays peut être référencé plusieurs fois dans le même élément `sea`. Ce sont donc les éléments `<sea>` qu'il faut compter, pas les `<located>`.

# 4. Travail à rendre

Déposez seulement votre fichier `ReponsesTP5a.xq` (tel quel, non compressé) sur Moodle, dans la page de cours dédiée, `TP5 partie 1` L4IN121T Formats et traitements de données internet.