République Algérienne Démocratique Et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ ABDELHAMID IBN BADIS – MOSTAGANEM
FACULTÉ DES SCIENCES EXACTES ET DE L'INFORMATIQUE
DEPARTEMENT DE MATHEMATIQUES-INFORMATIQUE

.....

DONNÉES SEMI-STRUCTURÉES

COURS ET EXERCICES CORRIGÉS

.....

Polycopié pédagogique réalisé par:

Dr. HOCINE Nadia

Maître de Conférences -classe A
Département de Mathématiques et Informatique

Polycopié destiné aux étudiants en 3ème année Licence Informatique
Filière Informatique
Option: SI

Année universitaire : 2024 - 2025

AVANT-PROPOS

Ce polycopié de cours a été réalisé pour la matière intitulée « *Données Semi-Structurées* » pour les étudiants en troisième année Licence informatique, option: Systèmes Informatiques (SI) de l'université d'Abdelhamid Ibn Badis, faculté des sciences exactes et informatique, département de mathématiques et informatique.

Ce cours se focalise sur l'étude des modèles et techniques d'exploitation des données semi-structurées en particulier à travers l'étude du langage XML (Extensible Markup Language). Le polycopié est structuré suivant le programme de la matière comme suit:

Chapitre I présente les principes de base de la structuration des données en XML (ou noyau XML) ainsi que les méthodes de validation associées.

Chapitre II explore en détail divers langages, interfaces de programmation et spécifications associés à XML, tels que XPath, XSLT, SAX, SVG, RDF, et d'autres. Ce chapitre permet aux étudiants de se familiariser avec cet ensemble de technologies XML (ou galaxie XML) et de comprendre comment XML est utilisé dans divers domaines d'application, tels que la description des ressources Web, la définition des images vectorielles et la description des services Web.

Chapitre III se concentre sur les bases de données XML et les langages de requêtes utilisés pour interroger ces bases, en particulier le langage XQuery.

En complément, le **Chapitre IV** offre une vue d'ensemble de quelques formats de données semi-structurées contemporains, tels que JSON et CSV. Des exemples de traitement de ces formats à l'aide du langage Python sont également présentés pour illustrer les différences de traitement par rapport à XML.

ACRONYMES

CSV: Comma-Separated Values

DOM: Document Object Model

DTD: Document Type Definition

HTML: HyperText Markup Language

INI: Initialization File

JSON: JavaScript Object Notation

Lorel: Language for Object Retrieval

RDF: Resource Description Framework

RELAX NG: Regular Language for XML Next Generation

RSS: Really Simple Syndication

SOAP: Simple Object Access Protocol

SAX: Simple API for XML

TOML: Tom's Obvious, Minimal Language

W3C: World Wide Web Consortium

WSDL: Web Services Description Language

XLink: XML Linking Language

XPath: XML Path Language

XPointer: XML Pointer Language

XQuery: XML Query Language

XQL: XML Query Language

XSD: XML Schema Definition

XSL: eXtended Style Language

XSLT: Extensible Stylesheet Language Transformations

XML: Extensible Markup Language

SVG: Scalable Vector Graphics

Quilt: Query Language for Unstructured Information

LISTE DES FIGURES

Figure	Page
Figure 1. Représentation arborescente des données semi-structurées	10
Figure 2. Exemple d'un document XML	11
Figure 3. Structure générale d'une DTD interne	12
Figure 4. Structure générale d'une DTD externe	13
Figure 5. Terminologie des noeuds en XPath	31
Figure 6. XML, XSLT et HTML	37
Figure 7. Principe de fonctionnement d'un SGBD-XML natif	59

LISTE DES TABLEAUX

Table	Page
Table 1. Exemples d'expression générales XPath	33
Table 2. Exemples d'expression XPath avec des prédicats	34
Table 3. Exemples d'expression XPath avec différents axes	35
Table 4. Commandes XSL pour décrire les règles	38

TABLE DES MATIÈRES

AVANT-PROPOS **1**
ACRONYMES **2**
LISTE DES FIGURES **3**
LISTE DES TABLEAUX **3**
TABLE DES MATIÈRES **4**
INTRODUCTION **6**
1. Données 6
1.1 Les données structurées 6
1.2 Les données non structurées 6
1.3 Les données semi-structurées 7
2. Documents multimédias 7
3. Hypermédia et hyperdocuments 7
4. Modélisation des documents 8
Chapitre I: Noyau XML **9**
1. XML: définition et utilisation 9
2. Syntaxe XML 9
3. Validation d'un document XML 12
4. DTD 12
4.1 Déclaration des éléments 14
4.2 Déclaration des entités dans la DTD 15
4.3 Les attributs 16
4.4 Exemple d'une DTD 17
4.5 Limitations des DTD 17
5. Schéma XML (XSD) 17
5.1 Syntaxe générale d'un XSD 18
5.2 Les éléments simples 19
5.3 Les attributs 20
5.4 Les éléments complexes 20
5.5 Les indicateurs 22
5.6 Exemple d'un XSD 23
6. Valider un XML à partir d'un langage évolué 25
7. Exercices 26
Chapitre II: Galaxie XML **31**
1. XPath 31
1.2 Syntaxe de l'expression de chemin 32
1.3 Opérateurs et fonctions 35
2. XSLT 36
2.1 Les règles XSLT 37
2.2 Exemple XSLT 38
3. Processeurs et interfaces de programmation pour XML 40

3.1 Processeur basé sur DOM et SAX....40
3.2 API Stax....41
4. Applications XML: RDF et SVG....47
4.1 RDF....47
4.2 SVG....48
5. Galaxie XML: en savoir plus....49
5.1 Xlink....49
5.2 WSDL....49
6. Exercices....52
Chapitre III: Base de données XML et XQuery....59
1. Base de données XML et SGBD....59
2. Langages de requêtes....60
2.1 Lorel....60
2.2 XQL....60
2.3 Quilt....61
2.4 XQuery....61
3. Le langage XQuery....61
3.1 Syntaxe générale....61
3.2 Clauses de XQuery....62
3.3 Expressions conditionnelles....66
3.4 Fonctions utilisateur....66
4. Exercices....68
Chapitre IV: Autres langages pour les données semi-structurées....72
1. YAML....72
2. JSON....73
3. CSV....75
SOLUTION DES EXERCICES....77
CHAPITRE I....77
CHAPITRE II....88
CHAPITRE III....94
CONCLUSION....98
BIBLIOGRAPHIE....99

INTRODUCTION

Données, Documents et Hyperdocuments

Avec le développement des technologies d'information et de communication, les données qualitatives et non structurées ont suscité l'intérêt de plusieurs travaux de recherche qui souhaitent rendre les systèmes plus intelligents et personnalisables. Ces données sont issues par exemple des courriels électroniques et de l'analyse des opinions, des comportements et des émotions des utilisateurs lors de leur interactions avec des systèmes et les contenus multimédia. L'analyse et le traitement de données et de documents qui ne sont pas souvent structurés à l'aide des enregistrements dans des bases de données classiques ou des tableaux peut s'effectuer à travers de nouvelles techniques et langages de traitement de données dites, semi-structurées.

1. Données

Les données sont devenues aujourd'hui de plus en plus primordiales pour la prise de décision et le fonctionnement des systèmes. Par exemple, des données numériques, textuelles, images et des traces d'interaction des utilisateurs avec les systèmes peuvent constituer une mine d'informations qui doivent être traitées, transformées, stockées et analysées pour sélectionner les informations les plus pertinentes pour l'utilisateur. Les données sont généralement décrites par leur type (numérique, image, texte, vidéo, …) et par leur structuration: les données structurées, non structurées et les données semi-structurées.

1.1 Les données structurées

Ces données sont généralement quantitatives et stockées dans des bases de données telles que NoSql et MongoDB ou possèdent une structure tabulaire tel que Excel avec des lignes et colonnes bien structurées. L'accès et le stockage de ces données est souvent facilité par les systèmes de gestion de base de données. Il est basé sur des enregistrements et des fichiers binaires lisibles par ces derniers. Afin de faciliter leur gestion, ces données doivent respecter une structuration rigide limitant par exemple le type et le volume des informations stockées. Par exemple, une base de données pour un système de gestion des étudiants doit clairement identifier des attributs des étudiants à stocker dans la base tels que leur identifiant, noms, date d'inscription, etc. Même si les données structurées sont faciles et rapides à gérer, elles sont généralement non extensibles et limitent l'évolution des applications.

1.2 Les données non structurées

Les données non structurées ou libres sont généralement des textes riches qui ne suivent pas des contraintes sur le contenu, des graphiques et images, des contenus hétérogènes incluant plusieurs types de média, vidéos, etc. Ces données sont souvent filtrées et transférées dans des formats structurés pour qu'elles puissent être analysées par la

machine. Les textes riches par exemples peuvent êtres traités par des techniques de traitement de langage naturel en construisant des arbres complets basés sur l'analyse de la structure de la phrase et la sémantique.

1.3 Les données semi-structurées

Même si les données structurées sont faciles à stocker et à analyser, elles ne permettent pas de stocker l'intégralité des données qui sont parfois primordiales pour la prise de décision. Les données qui sont basées sur des contenus multimédia, ne respectant pas une structure tabulaire mais possèdent une certaine structuration (comme des balises, des point virgules) permettant d'identifier la relation entre les éléments et/ou la sémantique sont appelées des données semi-structurées. Elles peuvent aussi êtres communiquer en réseau pour faciliter l'échange entre les applications. A la différence avec les données structurées, les données semi-structurées peuvent contenir des éléments de structure avec différentes hiérarchies et différents types de données. Ces données nécessitent un traitement et une analyse particulière qui est différente de celle des bases de données classiques. Plusieurs formats de données semi-structurées existent tels que XML, JSON, CSV, YAML, TOML, INI, etc. Ce cours concerne l'étude de formats d'échange de données entre applications Internet ou en réseau en général notamment à travers l'étude du langage XML.

2. Documents multimédias

La structuration ou la semi-structuration des données dépend de leur type et l'objectif de leur traitement. Parmi les données les plus concernées par le processus de structuration et de transformation sont les données multimédia issues des plateformes et applications d'éducation, de communication, des services à la personne, et de divertissement. Ces données sont généralement organisées sous forme de documents, appelés aussi documents multimédia.

Un document multimédia peut inclure un ou plusieurs types de données tels que le texte, l'audio, l'image et la vidéo. Sa structuration dépend de l'infrastructure de l'application ou du système et de l'environnement d'utilisation. Le contenu d'un document multimédia peut être linéaire, dont la structure ne nécessite pas une grande prise en compte de l'interactivité avec l'utilisateur. C'est le cas par exemple de la présentation d'un cours textuel ou une vidéo d'un film qui est passive et dont l'interaction ne change pas la structure du contenu. Le contenu non linéaire, également connu sous le nom de contenu hypermédia, dépend de l'interaction de l'utilisateur et sa structure peut être complètement modifiée selon plusieurs facteurs liés par exemple à l'environnement de l'utilisateur, son contexte, sa progression et son rythme. C'est le cas par exemple des cours personnalisables, des jeux vidéo, des applications de santé basées sur la réalité virtuelle et la réalité augmentée, etc.

3. Hypermédia et hyperdocuments

Les documents hypermédia sont des documents constitués de différents éléments multimédia tels que le texte, l'audio, les graphiques,les images et les vidéos ainsi que des textes avec liens entre ces éléments, nommées des hypertextes. Le contenu est donc non linéaire et dépend de la navigation de l'utilisateur. Les hypermédias et les hypertextes

peuvent être aussi considérés comme des hyperdocuments. Ces derniers font référence à tout document qui nécessite des lectures interactives et qui est divisé généralement en nœuds reliés entre eux par des liens permettant la navigation entre tous les éléments du contenu tels qu'un texte, graphiques, son, ou vidéo.

4. Modélisation des documents

Parmi les techniques les plus communes de modélisation de documents est de séparer leur structure logique (contenu) de leur structure physique (présentation ou vue). Ainsi, pour une seule structure logique, un document peut avoir plusieurs structures physiques qui dépendent de la plate-forme et du périphérique de sortie. Plusieurs modèles logiques et physiques ont été proposés lors de ces deux dernières décennies. Les deux modèles sont composés d'éléments, d'objets, de modules. Dans le cadre des applications Web, le lien entre ces deux modèles se fait souvent à travers de feuilles de style (stylesheets). C'est le cas par exemple de HTML avec des feuilles de styles CSS et XML avec par exemple de feuilles de styles XSLT.

L'objectif de ce cours est d'apprendre à structurer les données en utilisant le langage XML. Nous aborderons également les différents langages et technologies associés à XML, ainsi que les langages de requêtes pour les bases de données traitant des documents XML. Enfin, ce polycopié se termine par une présentation d'autres formats de données semi-structurées utilisés actuellement, tels que JSON et CSV. Pour résumer, les chapitres suivants sont présentés:

- Chapitre I: Noyau XML: vérification et validation d'un document XML
- Chapitre II: Galaxie XML
- Chapitre III: Base de données XML et XQuery
- Chapitre IV: Autres langages pour les données semi-structurées

Chapitre I: Noyau XML

Vérification et validation d'un document XML

Ce chapitre est consacré à la description du langage XML à travers la spécification de sa syntaxe et les langages les plus communs pour sa validation. En effet, les documents XML sont souvent utilisés pour échanger des données en réseau. La structuration de ces données doit suivre un certain schéma pour limiter l'ambiguïté lors du traitement de données et valider la pertinence de données récupérées. Plusieurs langages peuvent être utilisés pour valider l'XML, on abordera plus particulièrement dans ce chapitre DTD et XSD.

1. XML: définition et utilisation

XML (Extensible Markup Language) est un langage de description de données semi-structurées et un standard pour l'échange et la publication de données en réseau. Il propose un format à la fois simple (de balisage) et riche permettant d'assurer l'intégrité des données. XML facilite aussi l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes. De plus, la plupart des systèmes de gestion de base de données et les langages de programmation supportent XML en proposant des processeurs et des bibliothèques pour faciliter la l'accès et le traitement de fichiers XML.

2. Syntaxe XML

Un document XML permet de définir la logique des données. Un fichier contenant ce document porte l'extension ".xml". Il est composé essentiellement d'un ensemble **d'éléments** décrits par des balises. Un document XML contient aussi la déclaration, les attributs, les commentaires et les entités. Un **élément** dans un document XML est désigné par une balise ouvrante, une valeur et une balise fermante : `<element> valeur </element>`. Il peut être aussi composé de sous-éléments et/ou des attributs. Voici un exemple d'un document XML:

```
0. <?xml version="1.0" encoding="UTF-8"?>
1. <lettre>
2. <émetteur priorité="p1" sécurité="normale"> Amine Belkacem</émetteur>
3. <destinataire> Ahmed Belkacem </destinataire>
4. <adresse-destination> 23 rue des Lions, Mostaganem 27000
5. </adresse-destination>
6. <objet> Demande </objet>
7. <contenu> texte de la demande </contenu>
    8. <-- ceci est un commentaire-->
9. </lettre>
```

Le document XML peut contenir une première balise de déclaration de la version XML utilisée ainsi que l'encodage de données en UTF-8 ou ISO. Cette ligne de déclaration est appelée aussi le **prologue** (voir ligne 0)**.** Le document contient la description des éléments suivant une structure **arborescente**. Dans cet exemple, l'élément `lettre` de la ligne 1 est composé de cinq **sous-éléments**. Le premier sous-élément `émetteur` (ligne 2) est décrit

par deux **attributs** qui sont `priorité` la priorité de la demande et `sécurité` le niveau de sécurité de la lettre donnés par l'émetteur ainsi que la valeur du sous-élément `Amine Belkacem`. Les autres sous-éléments (lignes 3, 4, 6, 7) sont des éléments simples décrits par leurs **valeurs**. La ligne 8 est un exemple de **commentaires** en XML (optionnel). Il faut noter que la structuration de ce document et le nom des balises et des attributs ainsi que leurs valeurs sont complètement **libres**. Néanmoins, certaines règles de syntaxe sont exigées.

Les règles de syntaxe d'un document XML sont très similaires à celles du langage HTML. La différence entre les deux est que les balises XML sont:

- Les noms des balises sont libres, ce qui rend la structure du document extensible. Nous pouvons par exemple, ajouter des balises additionnelles pour décrire un nouvel élément de la lettre sans que cela influence les entrées précédemment structurées.
- A toute balise ouvrante doit correspondre une balise fermante
- Le nom d'une balise est sensible à la casse et peut contenir des caractères alphanumériques, des tirets, et doit contenir au moins une lettre.

Des règles de structuration sont également exigées:

- Les balises de contenu doivent toutes être liées à un seul élément, nommé la racine (*root* ou *document*). En effet, un document XML possède une structure d'arbre. Tous les éléments y compris la racine peuvent être composés de sous-éléments appelées aussi des nœuds enfants de l'arbre XML.
- un prologue optionnel qui mentionne la version de XML utilisée.
- Les attributs XML sont placés uniquement à l'intérieur de la balise d'ouverture. Un élément peut être vide et constitué uniquement des attributs.

Un document XML est dit **bien formé** s'il respecte les règles précédentes.

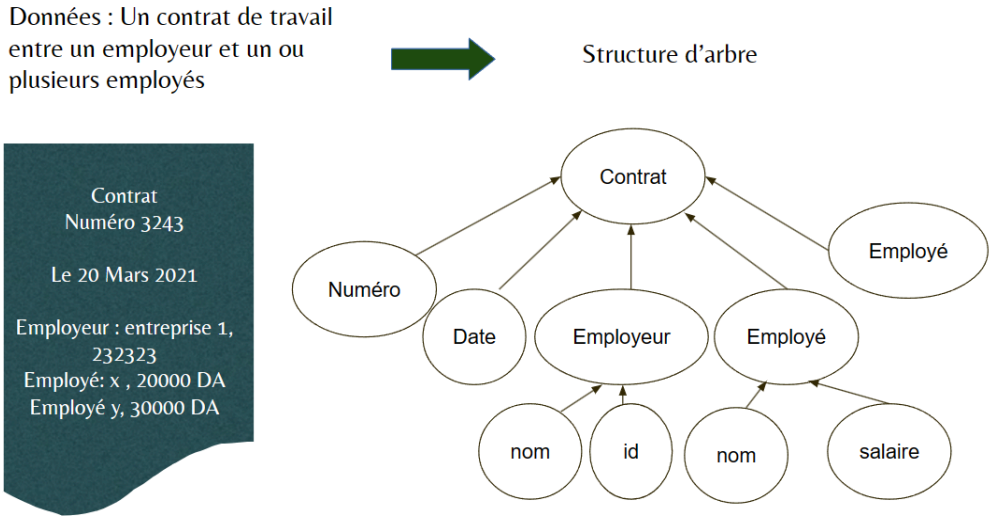


Figure 1. Représentation arborescente des données semi-structurées

Les éléments d'un document XML peuvent être hétérogènes ou ne possédant pas la même représentation logique. Dans un second exemple, on souhaite représenter un contrat entre un employeur et plusieurs employés. Un exemple de modélisation de l'arbre XML générique est donné dans la Figure 1. Dans cet exemple, nous avons choisi de limiter le nombre d'éléments et de sous-éléments (la balise "nom" est utilisée pour l'employeur et l'employé). Cela permettra de faciliter le traitement de document et sa validation (voir section suivante). La monnaie "DA" est une information qui est par défaut pour les contrats, on a choisi de la représenter par un attribut à la place d'un sous-élément.

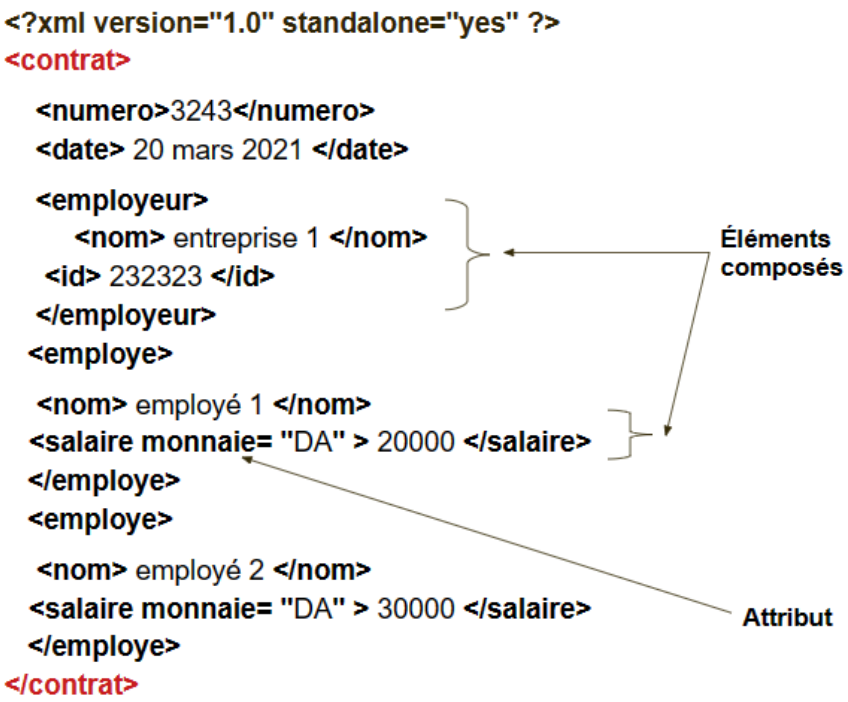


Figure 2. Exemple d'un document XML (contrat.xml)

La figure 2 illustre le document XML qui suit la structure d'arbre précédente. En général, les **sous-éléments** sont utilisés quand le contenu de l'élément est complexe possèdent plusieurs instances (valeurs) possibles. Ils sont aussi utilisés quand l'ordre des composantes est important et/ou l'extension de l'élément est envisagée. Les **attributs** sont favorisés dans les autres cas pour offrir une meilleure lisibilité au document. Ils sont souvent utilisés pour représenter des métadonnées.

Il est également possible d'introduire des **entités** au niveau des valeurs des éléments du document XML. Les entités sont généralement utilisées pour représenter les caractères spéciaux et des symboles réservés du langage qui ne peuvent pas être utilisés dans le document. Ces entités sont principalement < et > représentent les chevrons < et >, l'entité " qui remplace les guillemets, & pour le symbole & et ' pour ' (apostrophe simple). Il est également possible de définir d'autres entités par l'utilisateur (voir plus loin en Section 4.2).

3. Validation d'un document XML

Un document XML bien formé est un document qui respecte les règles de syntaxe vues précédemment. Cependant, les règles de syntaxes ne permettent pas de mettre des contraintes sur la structure de données échangées en réseau ou autre des applications Web. Par exemple, si un serveur doit utiliser un document XML reçu d'un client, il doit pouvoir l'analyser en identifiant les éléments et leurs sémantiques. Il est donc primordial de valider le document XML reçu pour vérifier si les contraintes préalablement établies avec le client dans la structuration du document ont été respectées. Les contraintes peuvent concerner par exemple la composition des éléments, l'ordre des éléments, le type de leur valeurs, etc. Plusieurs langages ont été proposés pour valider un document XML, parmi lesquels: DTD, XSD, RELAX NG et Schematron. Nous développons dans ce qui suit les deux langages: DTD et XSD.

4. DTD

DTD (Document Type Definitions) permet de définir la structure, les éléments et les attributs d'un document XML. Elle permet de spécifier les règles de validité et la hiérarchie des éléments. Elle est facultative, interne ou externe au document et contient des déclarations pour les éléments, attributs et notations utilisés. Elle permet de vérifier automatiquement des documents qui ont la même structure.

La syntaxe de la DTD est relativement simple. Elle commence par le mot clé <!DOCTYPE suivie par la racine et une définition des éléments et des attributs qui doivent figurer dans le document XML. La DTD peut être interne ou externe. Une DTD est interne si des éléments sont déclarés dans les fichiers XML. L'attribut "standalone" dans le prologue doit être défini sur "yes". Dans le cas d'une DTD externe, elle est définie dans un document externe portant l'extension ".dtd". La DTD est accessible depuis le document XML en spécifiant la référence vers le fichier .dtd et en donnant la valeur "no" à l'attribut "standalone" dans la déclaration XML. Voici la structure général d'un document xml contenant une DTD interne:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>

 <!DOCTYPE racine
 [
 <!ELEMENT racine (s-element1, s-element2, ...)>
 <!ELEMENT s-element1 (#PCDATA)>
 <!ELEMENT s-element2 (#PCDATA)>
 ]>

 <racine>
   <s-element1> val1</s-element1>
   <s-element2> val2</s-element1>
 </racine>
```

DTD

XML

Figure 3. Structure générale d'une DTD interne

On peut définir une DTD externe pour valider plusieurs documents XML qui ont la même structure comme suit:

racine.dtd

```
<!ELEMENT element1 (s-element1, s-element2, ...)>
<!ELEMENT s-element1 (#PCDATA)>
<!ELEMENT s-element2 (#PCDATA)>
```

racine.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>

<!DOCTYPE racine SYSTEM "racine.dtd">

<racine>
  <s-element1> val1</s-element1>
  <s-element2> val2</s-element1>
</racine>
```

Figure 4. Structure générale d'une DTD externe

Afin d'écrire la DTD, il faut commencer par la description de l'élément racine en utilisant le mot clé **ELEMENT** suivi par le nom de l'élément. Par exemple pour ce document XML:

```
<a>
<b>
<d> un texte </d>
</b>
<c att=”val”> un second texte </c>
</a>
```

La DTD externe suivante peut être proposée pour le valider:

```
 <!DOCTYPE a
 [
<!ELEMENT a (b,c)>
<!ELEMENT b (d)>
<!ELEMENT d (#PCDATA)>
<!ELEMENT c (#PCDATA)>
<!ATTLIST c att CDATA “val”>
 ]>
```

Les parenthèses expriment la composition des éléments ordonnés. Les feuilles de l'arbre qui représentent les éléments simples du document sont de type textuel ou PCDATA. Le mot clé **ATTLIST** est utilisé pour déclarer un attribut d'un élément de type textuel ou CDATA. La syntaxe utilisée pour décrire cet exemple sera expliquée en détail dans les sous sections suivantes.

4.1 Déclaration des éléments

Il existe plusieurs définitions d'éléments dans la DTD selon le type de ces derniers: simple, composé, mixte, libre et vide.

Elément simple

Un élément simple représente un élément feuille de l'arbre XML comme l'exemple suivant:

```
<!ELEMENT d (#PCDATA)>
```

La valeur de cet élément est de type texte "PCDATA". Le type indique à l'analyseur que la valeur de l'élément est une succession de caractères mais qui ne doit pas contenir les signes: &, <, >.

Elément composé

Un élément composé se constitue d'un ou plusieurs sous-éléments. Il existe deux types de composition des éléments. La première est **séquentielle** qui indique que l'ordre des sous-éléments est important, par exemple:

```
<!ELEMENT a(b,c)>
```

les balise b et c sont obligatoires et la balise b doit précéder la balise c.

La seconde composition est de type **alternative**, par exemple:

```
<!ELEMENT e(f, g,(h|j))>
```

dans ce cas la balise **h** ou la balise **j** succède la balise **g**. Les deux documents XML qui sont validés par cette DTD sont:

```
<e>                    <e>
<f> … </f>       <f> … </f>
<g> … </g>       <g> … </g>
<h> … </h>             <j> … </j>
</e>                 </e>
```

Il est possible d'utiliser aussi des **indicateurs d'occurrence**: **"*"** pour exprimer 0 ou plusieurs sous-éléments; ; **"+"** pour un ou plusieurs sous-éléments et **"?"** pour exprimer 0 ou 1 sous élément, par exemple:

```
<!ELEMENT chapitre (titre,intro?,section+)>
<!ELEMENT section (p|f)*>
```

Dans cet exemple, un élément **chapitre** est composé de l'élément **titre**, suivi éventuellement par l'élément **intro** et un ou plusieurs éléments **section.** L'élément **section** est composé éventuellement d'une succession d'éléments **p** ou **f**.

Élément mixte

Un élément mixte est un élément qui contient un texte libre et des sous-éléments, par exemple :

Ceci est un texte
 2

La seule syntaxe possible pour exprimer ce type d'éléments en DTD est la suivante selon l'exemple précédent:

```
<!ELEMENT p (#PCDATA|b)*>
```

Ou encore la solution suivante dans le cas de plusieurs sous-éléments:

```
<!ELEMENT p (#PCDATA|a|b|c)*>
```

Elément libre

Un élément libre est un élément dont le contenu est bien formé, mais sans restrictions particulières, par exemple :

```
<p> val </p>
<p> <a> val </a> </p>
```

```
<!ELEMENT p ANY>
```

Elément vide

Un élément est dit vide s'il ne possède pas un contenu (une valeur) mais contient un ou plusieurs attributs, par exemple: <k x='val' />

```
<!ELEMENT k EMPTY>
<!ATTLIST k x CDATA "val">
```

4.2 Déclaration des entités dans la DTD

Les entités générales sont référencées à l'aide du symbole "&" comme décrit dans la Section 2. Nous pouvons définir des entités pour représenter des caractères spéciaux. Par exemple, dans le document XML suivant on définit des entités pour é (eacute) et un espace insécable (nbsp).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE exemple[
    <!ENTITY eacute "&#233;">
    <!ENTITY nbsp " ">
]>
```

```
<exemple>
   <texte> Voici un exemple d'utilisation des caractères spéciaux en XML
           &eacute; (é) et   (espace insécable).
   </texte>
</article>
```

Les entités de type paramètre (ou définies par l'utilisateur) sont référencées avec le signe pourcent "%" et peuvent être déclarées dans la DTD. Pour cela il suffit d'utiliser le mot clé **`ENTITY`** suivie du "**`%`**", du nom de l'entité et les éléments qui ont été référencés par l'entité. Par exemple, voici la déclaration de deux entités de type paramètre:

```
<!ENTITY % personne "nom, prénom, age">
<!ENTITY % pnom "psuedo-nom">
```

Ces entités peuvent être utilisées dans le fichier XML ou la DTD comme montre l'exemple suivant:

```
…
 <!ELEMENT etudiant (%personne;, datenaissance)>
 <!ELEMENT utilisateur(%pnom;)>
…
```

4.3 Les attributs

Comme décrit précédemment, le mot clé **`ATTLIST`** est utilisé pour décrire les attributs en mentionnant l'élément puis le nom de l'attribut et ensuite son type. CDATA signifie que la valeur de l'attribut est constituée de simples caractères qui ne vont pas êtres examinés par un analyseur. Par exemple,

```
<!ATTLIST k x CDATA "val">
```

signifie que l'attribut **`x`** est un attribut de l'élément **`k`** et dont la valeur est une chaîne de caractères et qui est par défaut **`val`**.

Le type de l'attribut peut être aussi énuméré contenant plusieurs valeurs, par exemple:

```
<!ATTLIST k x (val1|val2)"val3">
```

L'attribut peut être aussi de type tokenisé, par exemple: ID pour identifier l'élément, IDREF pour référencer un ID préalablement nommé pour un autre élément, ENTITY qui représente une entité externe utilisée dans le document.
Il est également possible de spécifier quelques restrictions sur les attributs en ajoutant les attributs suivants dans la syntaxe:

- **#REQUIRED** : attribut obligatoire, valeur à être précisée dans le document
    - Exemple: **`<!ATTLIST k x CDATA #REQUIRED>`**
- **#IMPLIED** : attribut facultatif, valeur à être précisée dans le document

- Exemple: `<!ATTLIST k x CDATA #IMPLIED>`

- **#FIXED** (suivi de la valeur): valeur de l'attribut fixée pour tout élément instance
  - Exemple: `<!ATTLIST k x CDATA #FIXED "val">`

4.4 Exemple d'une DTD

Soit le document XML de la Figure 2 (contrat.xml). Voici une DTD qui valide ce document XML:

```
<!DOCTYPE contrat
 [
 <!ELEMENT contrat (numero, date, employeur, employe+)>
 <!ELEMENT numero (#PCDATA)>
 <!ELEMENT date (#PCDATA)>
 <!ELEMENT employeur (nom, id)>
 <!ELEMENT nom (#PCDATA)>
 <!ELEMENT id (#PCDATA)>
 <!ELEMENT employe (nom, salaire)>
 <!ELEMENT salaire (#PCDATA)>
 <!ATTLIST salaire monnaie CDATA "DA">
 ]>
```

La DTD dans cet exemple doit décrire et valider l'ensemble de documents XML représentant des contrats. Dans ce cas, il est possible d'ajouter ou de modifier certaines restrictions selon les spécifications des besoins. Par exemple, si les salaires dans les contrats peuvent être payés en euro ou en dollar l'attribut monnaie doit être modifié comme suit:

```
<!ATTLIST salaire monnaie (DA|dollar|euro)>
```

4.5 Limitations des DTD

DTD est un moyen simple de description des restrictions sur la syntaxe des documents XML. Elle définit la structure en arbre d'un document XML, mais elle possède une syntaxe spécifique qui est limitée. Le seul type de données est le texte PCDATA ou CDATA. On ne peut pas dans l'exemple précédent des contrats de spécifier par exemple que le salaire doit être numérique ou de restreindre ses valeurs possibles. On ne peut pas également préciser la cardinalité des instances telle que le nombre d'employés liés à un contrat. Enfin, le langage des DTD n'est pas extensible et ne permet pas de représenter les différentes structures de données exprimées dans un document XML. Néanmoins, il permet de définir rapidement et en quelques lignes les restrictions syntaxiques générales d'un document XML.

5. Schéma XML (XSD)

Afin de définir des contraintes syntaxiques sur un document XML, un schéma XML peut être utilisé. Plusieurs langages de schéma ont été définis pour exprimer des contraintes sur les documents XML, notamment le **XSD** (XML schema definition). XSD est un standard

pour la validation des documents XML qui utilise la syntaxe de balisage similaire à celle du XML. Il a l'avantage de faciliter l'expression des types de données primitifs (entiers, réels, dates, booléens, etc.) et les données de type complexe ou de nouvelles structures de données. Il permet d'exprimer également des espaces de noms (nom de balises bien définies) et d'ajouter des contraintes de cardinalité pour les éléments XML.

5.1 Syntaxe générale d'un XSD

Les règles d'écriture d'un schéma XML se basent sur le même principe que la DTD en définissant chaque élément et attribut du document XML. Dans XSD, il est important de faire la distinction entre un élément simple et un élément composé à travers son type. Prenant l'exemple suivant d'un document XML:

```
<a>
  <b> ceci est un texte </b>
  <c> ceci est un second texte </c>
</a>
```

Le document XSD simplifié est le suivant:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name = "a">
 <xs:complexType>
    <xs:sequence>
        <xs:element name = "b" type = "xs: string"/>
        <xs:element name = "c" type = "xs: string"/>
    </xs:sequence>
 </xs:complexType>
</xs:element>
```

Un document XSD doit contenir la définition de l'élément schéma **`xs:schema`** avec des attributs de déclaration tels que **`xmlns:xs`** qui indique l'URL des types de données référencés dans le schéma. Dans l'exemple précédent les éléments `c` et `b` sont des éléments simples **`xs:element`** définies par leur nom **`name`** et type **`type`** qui doit être primitif (dans cet exemple `xs: string`). L'élément `a` est un élément complexe. Afin de définir son type, il faut donc utiliser le mot clé **`xs:complexType`** qui indique que l'élément contient des sous-éléments et/ou des attributs et le mot clé **`xs:sequence`** qui indique que l'ordre d'apparition des sous-éléments `c` et `b` est séquentiel. Même si l'élément `a` représente dans cet exemple la racine du document, la même syntaxe est appliquée si c'était un élément complexe quelconque du document.

XSD est externe au document XML et son fichier porte l'extension "**`.xsd`**". Il est donc essentiel de lier le document XML à son schéma comme suit:

```
<a xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/chemin/..//classe.xsd">
```

On décrira par la suite les spécifications de syntaxe pour définir les éléments et les attributs en XSD.

5.2 Les éléments simples

Un élément simple ne peut contenir que des valeurs atomiques (ne contient pas des attributs et des sous-éléments). Sa syntaxe générale est décrite par son nom et type comme suit:

```
<xs:element name="..." type="..."/>
```

Voici par exemple quelques éléments XML avec leur description XSD:

- `<nom>Ali</nom>`
    - **`<xs:element name="nom" type="xs:string"/>`**
- `<age>51</age>`
    - **`<xs:element name="age" type="xs:integer"/>`**
- `<date-maissance>2000-02-21</date-maissance>`
    - **`<xs:element name="date-maissance" type="xs:date"/>`**

XSD définit un grand nombre de types simples prédéfinis, les plus connus sont: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time. Un type simple peut définir une **valeur par défaut** à l'aide de l'attribut **`default`** ou une **valeur fixe** à l'aide de l'attribut **`fixed`**, par exemple:

```
<xs:element name="couleur" type="xs:string" default="rouge"/>
<xs:element name="couleur" type="xs:string" fixed="rouge"/>
```

XSD permet de définir des restrictions sur les valeurs d'un élément en utilisant le mot clé **`xs:restriction`**. Différentes restrictions sont possibles dont les plus communes sont:

- **totalDigits**: précise le nombre exacte de numéros autorisés pour un type décimal (nombre total avant et après la virgule)
- **fractionDigits**: indique le nombre maximal de valeurs autorisés après la virgule pour un type décimal
- **maxExclusive** et **maxInclusive**: valeur maximale des valeurs numériques (< pour exclusive et <= pour inclusive)
- **minExclusive et minInclusive**: valeur minimale des valeurs numériques (> pour exclusive et >= pour inclusive)
- **pattern**: définit l'expression régulière ou la séquence de caractères qui sont acceptables
- **length**: nombre de caractères ou des items
- **maxLength et minLength**: nombre maximal et minimal de caractères ou les items autorisés
- **enumeration**: définit une liste de valeurs limitée

Voici quelques exemples d'éléments simples avec des restrictions sur leurs valeurs: Dans le premier exemple, on définit un élément âge dont la valeur est de type entier entre 0 et 100.

```
<xs:element name="age">
<xs:simpleType>
      <xs:restriction base="xs:integer">
         <xs:minInclusive value="0"/>
         <xs:maxInclusive value="100"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Dans le second exemple, on définit un élément mot de passe `pwd` qui est une chaîne de caractères composée d'une suite de 6 lettres et chiffres.

```
<xs:element name="pwd">
<xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9]{6}"/>
   <xs:length value="6"/>
 </xs:restriction>
</xs:simpleType>
</xs:element>
```

Le troisième exemple, trois valeurs possibles des marques sont définies pour l'élément voiture: Mercedes ou Kia:

```
<xs:element name="voiture">
<xs:simpleType>
 <xs:restriction base="xs:string">
         <xs:enumeration value="Mercedes"/>
         <xs:enumeration value="Kia"/>
      </xs:restriction>
</xs:simpleType>
</xs:element>
```

5.3 Les attributs

Les attributs peuvent être définis pour les éléments complexes à l'aide du mot clé **`xs:attribute`** et en précisant leur nom et type qui est primitif. Il est également possible d'indiquer si l'attribut possède une valeur par défaut **`default`**, fixe **`fixed`**, obligatoire **`required`** ou optionnelle **`optional`**. Voici quelques exemples d'attributs définis en XSD:

```
<xs:attribute name="a" type="xs:string"/>
<xs:attribute name="b" type="xs:string" default="DA"/>
<xs:attribute name="c" type="xs:string" fixed="Euro"/>
<xs:attribute name="d" type="xs:string" use="optional"/>
<xs:attribute name="e" type="xs:string" use="required"/>
```

5.4 Les éléments complexes

Un élément complexe peut être: un élément qui se compose d'un ou plusieurs

sous-éléments, un élément vide qui ne possède pas une valeur mais définit qu'avec un ou plusieurs attributs, ou bien un élément possédant une valeur et des attributs. Le type de cet élément est défini avec le mot clé xs:complexType. Il existe deux méthodes pour définir un élément de type complexe. La première consiste à le définir avec un type anonyme (n'est pas nommé). Par exemple, on décrit un élément employé `employe` par son nom et prénom comme suit:

```
<xs:element name="employe">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="nom" type="xs:string"/>
     <xs:element name="prenom" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

La seconde méthode de définition d'un élément complexe consiste à nommer le type complexe pour qu'il soit utilisé dans la définition de types de plusieurs autres éléments de notre document XML comme suit:

```
<xs:element name="employee" type="personne"/>
<xs:complexType name="personne">
 <xs:sequence>
   <xs:element name="nom" type="xs:string"/>
   <xs:element name="prenom" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Dans cet exemple, l'élément `employe` est de type **`personne`** un nouveau type qu'on crée et on définit à travers la balise `xs:complexType`.

On peut réutiliser ce nouveau type nommé pour définir d'autre éléments, par exemple:

```
<xs:element name="etudiant" type="personne"/>
<xs:element name="enseignant" type="personne"/>
```

Si l'élément complexe contient uniquement des attributs et ne possède pas une valeur (**élément vide**), la même syntaxe est appliquée, par exemple:

```
<produit reference="04503"/>
```

```
<xs:element name="produit">
    <xs:complexType>
   <xs:attribute name="pid" type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Si l'élément complexe contient une valeur textuelle et des attributs, il faut obligatoirement définir une restriction (ou extension) comme l'exemple qui suit:

```
<a b="val"> 2 </a>
```

```
<xs:element name="a">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
       <xs:attribute name="b" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Un élément complexe qui contient un texte et des sous éléments doit être défini avec le type **mixte** en utilisant l'attribut **mixed** dans la définition de `xs:complexType`.

```
<demande>
A Monsieur le directeur <institut> institut de santé </institut>,
<num_demande>20240213</num_demande>,
stage court <date>2024-10-15</date>
</demande>
```

```
<xs:element name="demande">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="institut" type="xs:string"/>
      <xs:element name="num_demande" type="xs:positiveInteger"/>
      <xs:element name="date" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Enfin, il est possible d'utiliser **<any>** et **<anyAttribute>** pour définir des éléments sans spécification particulière des sous-éléments et des attributs comme montré dans l'exemple suivant:

```
<xs:element name="personne">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prénom" type="xs:string"/>
      <xs:any minOccurs="0"/>
          <-- on peut rajouter n'importe quel élément ici-->
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

5.5 Les indicateurs

Les indicateurs sont utilisés pour contrôler l'occurrence et l'ordre des éléments d'un document XML. Il existe trois types d'indicateurs: Les indicateurs d'ordre (All, Choice, et

Sequence) et les indicateurs d'occurrence (maxOccurs et minOccurs)

Dans l'exemple suivant, on a utilisé l'indicateur **<xs:all>** qui signifie que les éléments "nom" et "prénom" peuvent apparaître une seule fois et dans n'importe quel ordre.

```
<xs:element name="employe">
 <xs:complexType>
   <xs:all>
     <xs:element name="nom" type="xs:string"/>
     <xs:element name="prenom" type="xs:string"/>
   </xs:all>
 </xs:complexType>
</xs:element>
```

Si on a utilisé l'indicateur **<xs: choice>** à la place de <xs:all>, cela signifie qu'un des éléments enfants peuvent apparaître (nom ou prénom). Enfin, l'indicateur d'ordre **<sequence>** spécifie que les sous-éléments doivent apparaître dans un ordre spécifique (le nom avant le prénom).

Les indicateurs d'occurrence sont utilisés pour limiter le nombre d'instances d'un élément. Dans l'exemple suivant, on fixe le maximum d'éléments produits à 100.

```
<xs:element name="magasin">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="propriétaire" type="xs:string"/>
     <xs:element name="produit" type="xs:string" maxOccurs="100"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Il est également possible d'utiliser plusieurs attributs d'indicateurs, par exemple:

```
 <xs:element name="produit" type="xs:string" minOccurs= 1
             maxOccurs="100"/>
```

Dans cet exemple, le nombre de produits est limité dans un intervalle entre 1 et 100.

5.6 Exemple d'un XSD

Soit le document XML suivant représentant une entreprise, sachant que l'entreprise possède un seul PDG (directeur) et au moins un ingénieur.

```
<entreprise>
      <pdg>
            <nom> Ahmed </nom>
            <prenom> Ali </prenom>
      </pdg>
      <ingenieur>
            <nom> Samir </nom>
            <prenom> Samir </prenom>
            <experience> 3 </experience>
      </ingenieur>
```

```
      <ingenieur>
            <nom> Ahmed </nom>
            <prenom> Ahmed </prenom>
      </ingenieur>
      <comptable>
            <prenom> Otmane</prenom>
            <nom> Otmane</nom>
      </comptable>
</entreprise>
```

Le XSD qui valide ce document XML est le suivant:

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="entreprise">
      <xs:complexType>
      <xs:sequence>
      <xs:element name=« pdg" minOccurs="1">
      <xs:complexType>
      <xs:sequence>
            <xs:element type="xs:string" name="nom"/>
            <xs:element type="xs:string" name="prenom"/>
      </xs:sequence>
      </xs:complexType>
      </xs:element>
      <xs:element name="ingenieur" maxOccurs="unbounded" minOccurs="1">
      <xs:complexType>
      <xs:sequence>
            <xs:element type="xs:string" name="nom"/>
            <xs:element type="xs:string" name="prenom"/>
            <xs:element type="xs:float" name="experience" minOccurs="0"/>
      </xs:sequence>
      </xs:complexType>
      </xs:element>
      <xs:element name="comptable">
      <xs:complexType>
      <xs:sequence>
            <xs:element type="xs:string" name="prenom"/>
            <xs:element type="xs:string" name="nom"/>
      </xs:sequence>
      </xs:complexType>
      </xs:element>
      </xs:sequence>
      </xs:complexType>
      </xs:element>
</xs:schema>
```

Afin d'établir le schéma correspondant à un document XML, il est important de dessiner l'arbre afin de déterminer la structure des éléments ainsi que leur occurrence. Dans cet exemple, le sous élément expérience ne figure que dans un élément ingénieur, ce qui nécessite de préciser la cardinalité pour un minimum de zéro apparence. Les sous-éléments nom et prénom doivent être bien ordonnés selon les éléments (par exemple ingénieur et comptable) puisque le mot clé xs:séquence du type complexe exige que l'ordre

des sous-éléments doit être respecté. Enfin, il est essentiel de prendre en compte les spécifications de besoins afin que le XSD puisse valider l'ensemble des documents XML qui suivent la structure convenue.

6. Valider un XML à partir d'un langage évolué

Il est souvent pratique d'utiliser la validation d'un XML à partir d'un langage évolué, tels que Java, Python, C, etc. La plupart des langages de programmation proposent des bibliothèques de fonctions pour manipuler et valider les documents XML. Ils sont souvent basés sur des fonctions de traitement de fichiers de type texte, auquel est ajouté des fonctionnalités permettant de faciliter la validation et le traitement des données semi-structurées.

Dans le pseudo code python suivant par exemple, on vérifie un document XML et on le valide à travers un document XSD. Nous aurons besoin de Python 3 et de la bibliothèque lxml.

```
import os
from lxml import etree
# arguments :
# xml_file: le chemin vers le document XML
# xsd_file: le chemin vers son schéma XSD.
# vérifier l'XML
    try:
        xml_tree = etree.parse(xml_file)
        except etree.XMLSyntaxError as err:
        print("Erreur pour parser l'XML: {err}")
        return None
    # Créer l'objet XSD
    try:
        schema = etree.XMLSchema(etree.parse(xsd_file))
        except etree.XMLSchemaParseError as err:
        print("Erreur de parser l'XSD: {err}")
        return None
    # Valider l'XML avec le XSD
    if not schema.validate(xml_tree):
        print(" XML non valide ! ")
        return None
```

Enfin, ce chapitre a présenté le format XML pour la représentation des données semi-structurées. Il a également introduit deux exemples de sa validation à travers DTD et XSD. Le chapitre suivant permet de présenter quelques outils, spécifications et langages dédiés à XML.

7. Exercices

Les exercices suivants peuvent être réalisés à travers l'outil XML COPY EDITOR[1]. C'est un éditeur gratuit et open source permettant d'éditer, vérifier et valider des documents XML.

Exercice 1:

Soit le document XML (classe.xml) suivant:

```
<?xml version="1.0" encoding="UTF-8"?>

 <classe>

<eleve ident="E01">
      <nom> Mansar </nom>
      <prenom>Ali</prenom>
      <adresse>
            <numero>23</numero>
            <voie cat="rue">Aban Ramdhan</voie>
            <ville> Oran </ville>
      </Adresse>
      <resultats.concours->
            <ar?>12</ar?>
            <math>9</math>
            <phys>6</phys>
      </resultats.concours>
</eleve>

<eleve ident:"E02">
      <nom> Alaribi </nom>
      <prenom>Amina</prenom>
      <adresse>
            <numéro>16</numéro>
            <voie cat="allée">des artistes</voie>
            <ville =Mostaganem</ville>
      </adresse>
      <resultats>
            <ar>11
            <phys>16</phys>
      </resultats>>
</eleve>
```

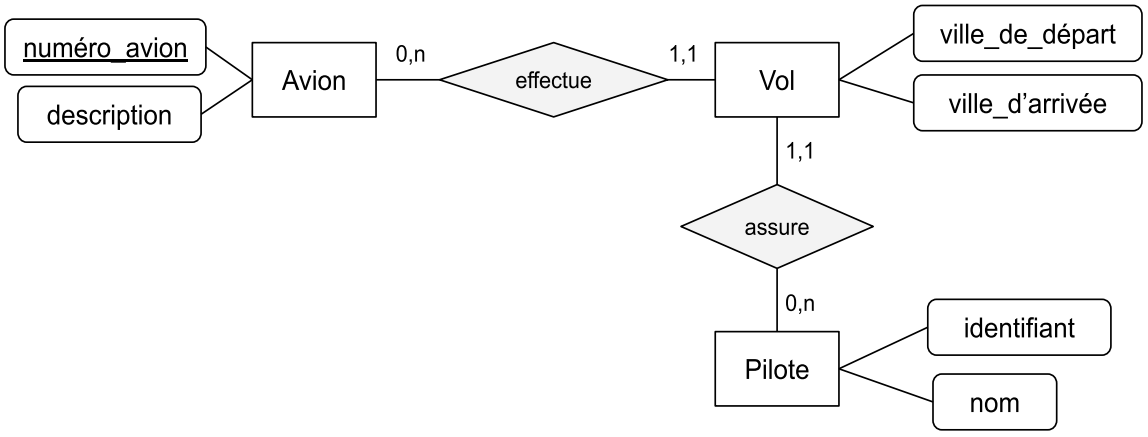
1. A l'aide de XML Copy Editor, sauvegarder votre fichier (avec la bonne extension) puis vérifiez si votre document XML est bien formé à l'aide de XML COPY EDITOR.
2. Listez et corrigez les erreurs trouvées. Proposer une amélioration de la modélisation XML utilisée.

[1] XML COPY EDITOR: https://xml-copy-editor.sourceforge.io/

3. Quel est le type des éléments suivants: classe, adresse, ville, cat, E01, Mostaganem.
4. Complétez le document XML précédent en ajoutant deux élèves:
    a. Amer Mohammed qui habite à 3 avenue Mostaganem à Oran et qui a obtenu les notes suivantes: 10 en arabe, 19 en mathématique et 12 en physique. Son adresse email est amer.mohammed@umab.dz
    b. Balim Houda qui habite à 20 rue Khemisti à Mostaganem et qui a obtenu les notes suivantes: 10 en mathématique, 15 en arabe et 20 en physique.

Exercice 2:

On souhaite représenter des vols organisés par des compagnies aériennes en se basant sur le modèle entité-association suivant:



La compagnie Air Algérie propose trois vols. Les trois vols sont nationaux, effectués entre Oran et Annaba, et sont assurés respectivement par le pilote Mohamed Mehdi dont l'identifiant est 0021dz; Belkacem Amin dont l'identifiant est 0022dz et Limam Fatima dont l'identifiant est 0020dz. L'avion utilisé est de type Airbus A330-200, portant le numéro 2745. Un quatrième vol est proposé par la compagnie TunisAir (Tunisie) effectué entre l'aéroport international d'Oran Ahmed Ben Bella et l'aéroport international Tunis Carthage le 22/03/2024 entre 09h et 10h30. Il est assuré par le pilote Mohamed Elghani dont l'identifiant est 0031tn. L'avion utilisé est de type Airbus A380, portant le numéro 0012.

Le document XML doit contenir les dates de départ et d'arrivée de chaque vol national ou international si l'information est disponible, ainsi que les aéroports de départ et d'arrivée. Il faut inclure également des informations sur la compagnie aérienne assurant le vol, notamment son nom et le pays d'origine.

1. Donnez le document XML (vols.xml) puis vérifiez si le document XML est bien formé à l'aide XML COPY EDITOR.
2. Donnez la DTD du document XML (vols.dtd). Vérifiez ensuite si votre document XML est valide à l'aide de la DTD. Sur XML COPY EDITOR, vous devez d'abord associer votre fichier xml à la DTD (Menu: XML-> Associer -> DTD system ->sélectionner), ensuite vérifier la validité du fichier XML.

Exercice 3:

Donnez la description DTD pour chaque document XML suivant (les documents sont indépendants):

1.
```
<films>
<film type="action" affiche="affiche1.jpg"/>
<film type="science fiction" affiche="affiche2.jpg"
      remarques="avant première"/>
<film type="comédie" affiche="affiche3.png"/>
  </films>
```
2.
```
<expression>
<exp1>
  <!--une séquence de a ou une séquence de b→
<b>y</b><b>y</b>
   </exp1>
      <exp2>
      <!--une séquence de ab précédée éventuellement par une séquence de
b-->
      <a>x</a><b>y</b><a>x</a><b>y</b>
   </exp2>
   <exp3>
     <!--une alternance de c ou de b, suivie éventuellement de a ou de d →
<b>y</b><c>z</c><c>z</c><a>x</a>
   </exp3>
 </expression>
```
1.
```
    <Lettre> À Monsieur le PDG de l'entreprise <objet> demande de stage
</objet> pour une durée de: <durée> 3 mois </durée> </Lettre>
```

Exercice 4:

On veut produire un document XML permettant de représenter les programmes d'une chaîne de télévision. On dispose du cahier des charges suivant:

- Une chaîne doit dérouler des programmes.
- Un programme peut être : divertissement, documentaire, journal, série ou film.
- Les programmes sont décrits par leur durée (exprimée en minutes), le nombre de répétitions dans la journée et une description.
- La description peut contenir une phrase qui résume l'objectif du programme. Elle peut contenir éventuellement des informations sur le producteur (nom et prénom) ainsi que l'année de production.
- Un journal est décrit également par son directeur et le journaliste responsable (optionnel), les deux sont décrits par leurs prénom et nom.
- Un documentaire est décrit également par sa catégorie qui peut être : nature, animaux, politique, société ou sport.

1. Donnez le document XML (chaine.xml) bien formé qui représente une chaîne de télévision et qui contient les trois programmes suivants :
    - Un documentaire sur la nature et les animaux qui dure 2h et qui passe à 10h et à 22h. C'est un documentaire sur l'Amazonie et le risque de sécheresse.

- Un journal qui passe à 20h qui dure une demi-heure dont le directeur est Ali Labouré, le producteur Amin Dimi et la journaliste Hala Salamé.

- Un film qui est programmé à 13h et qui dure 2h. C'est un film d'action qui date de 2016 et dont le producteur est George Ème.

2. Donnez La DTD (chaine.dtd). Vérifiez si votre document XML est valide à l'aide de la DTD.

Exercice 5:

On veut produire un document XML permettant de représenter une formation universitaire. On dispose du cahier des charges suivant:

- Une formation est décrite par sa spécialité et la description des semestres
- Un semestre est décrit par son numéro, date de début, niveau : Licence ou Master ainsi que les matières
- Une matière est décrite par son code, intitulé, type : fondamentale, méthodologique ou découverte, responsable, son contenu, pré-requis (optionnel) et son organisation en termes de cours, td et tp. Le contenu est décrit soit par un objectif soit par un texte (exp. paragraphe).

1. Donnez le document XML bien formé correspondant aux données suivantes d'une formation en informatique, semestre 6 Licence débutant 2022/01/01 et dont les matières sont :

code	intitulé	type	responsable	contenu	organisation	Pré-requis
UEF2	DSS	Fondamentale	Mme Hocine	Objectif : L'objectif est de se familiariser avec la structuration et le traitement des données non structurées	Cours (1h) TP (1h)	langage de programmation
UEM 1	Cloud	Méthodologique	M. Ali	cette matière offrira aux étudiants la base de supervision réseau en cloud	Cours (2h) TD (1h)	Réseau TCP IP

2. Donnez la DTD permettant de représenter les documents XML d'une formation universitaire, puis vérifiez si le document XML précédent est valide à l'aide de cette DTD.
3. Donnez le schéma XML (XSD). Vérifiez si votre document XML est valide à l'aide du schéma XML.

Exercice 6:

Ci-après un document XML donnant un exemple de factures d'une société qui propose plusieurs articles et services. On suppose que le prix unitaire est en dinar algérien par défaut mais peut être aussi en Euro. Le prix des services est réglé uniquement en dinar algérien.

```
<?xml version="1.0" encoding="UTF-8"?>
<facture>
<etabliepar> Société Méditerranéenne</etabliepar>
<date> 28 janvier 2024</date>
<areglerpar> UMAB </areglerpar>
 <article>
<description> cartouches imprimante epson </description>
<nombre> 200 </nombre>
 <prix_unitaire monnaie="da"> 1000 </prix_unitaire>
</article>
<article>
<description> rame de papier A4 80g</description>
 <nombre> 50 </nombre>
 <prix_unitaire monnaie="da"> 500 </prix_unitaire>
</article>
<service>
<description> aménagement nouveau bâtiment </description>
 <prix  monnaie="da"> 10000 </prix>
</service>
<total>235000</total>
</facture>
```

1. Donnez la DTD du document XML (facture.dtd). Vérifiez si le document XML est valide à l'aide de la DTD.
2. Modifiez la DTD pour valider des factures disposant uniquement d'articles et qui mentionnent la TVA. La nouvelle DTD (facture2.dtd) doit tout de même valider le document XML précédent.
3. En se basant sur la DTD (facture2.dtd), donnez le schéma XML (XSD) équivalent (facture.xsd). Vérifiez si votre document XML (facture.xml) est valide à l'aide du schéma XML.

Chapitre II: Galaxie XML

XML dispose d'une galaxie d'outils, infrastructures logicielles (frameworks) et de langages permettant de manipuler, analyser et transformer les documents XML. On présentera dans ce chapitre quelques éléments de cette galaxie tout en décrivant leur utilisation.

1. XPath

XPath ou **XML Path Language** est un langage d'expression de chemins dans un arbre XML. Il permet de rechercher et sélectionner des éléments et des attributs dans un document XML. Cependant, XPath ne donne pas la possibilité de mémoriser un résultat de recherche ou d'une sélection puisqu'il ne permet pas de définir des variables. En effet, XPath n'est pas un langage autonome et il est utilisé uniquement au sein d'un autre langage hôte.

Afin d'exprimer des chemins, XPath considère un document XML comme un arbre formé de nœuds qui sont: la racine (appelé aussi le nœud document), attribut, texte, les espaces de noms, les instructions de traitement, les commentaires, etc. Des exemples sur ces nœuds sont illustrés dans la figure 5.

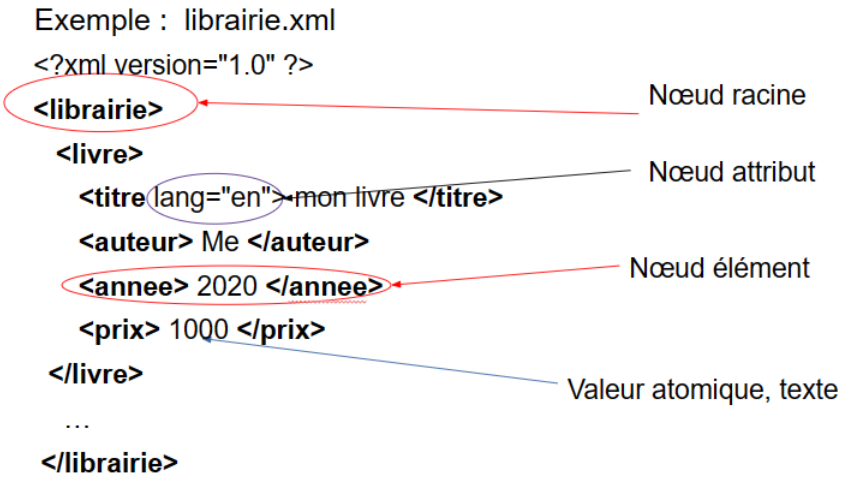


Figure 5. Terminologie des noeuds en XPath

XPath utilise également une terminologie pour nommer les noeuds et le lien entre eux dans la hiérarchie de l'arbre, par exemple:

- **Parent**: chaque nœud possède un parent, sauf le nœud racine. Dans l'exemple de la figure 5, le noeud livre est le parent du noeud auteur.
- **Children**: les noeuds peuvent avoir des noeuds enfants, sauf les feuilles de l'arbre. Dans l'exemple, auteur et année sont les enfants du noeud livre.
- **Siblings**: se sont les nœuds qui se trouvent sur le même niveau de hiérarchie et qui ont donc le même parent. Dans l'exemple auteur et année sont des frères ou siblings.
- **Ancestors**: se sont les ascendants du nœud (le parent, le parent du parent, etc).

Dans l'exemple précédent, le noeud année possède les ascendants livre et librairie.

- **Descendants**: se sont les descendants du nœud (les enfants, les enfants de enfant, etc). Les descendants de librairie dans l'exemple sont: livre, auteur, année, etc.

1.2 Syntaxe de l'expression de chemin

Une expression de chemin décrit le parcours d'un nœud initial à des nœuds cibles à travers l'arborescence du document. La syntaxe générale de l'expression de **chemin absolu** est la suivante: **/étape$_1$/étape$_2$/…/étape$_N$** Dans le cas d'un **chemin relatif**, l'expression commence à un nœud courant: **étape$_1$/étape$_2$/…/étape$_N$**

Chaque **étape** est exprimée comme suit: **axe::filtre[prédicat$_1$]…[prédicat$_n$]**

- **axe**: optionnel, il représente l'axe de recherche dans l'arbre qui est par défaut "child". Cela signifie que la recherche se fera à partir du nœud courant vers les nœuds descendants (verticale).
- **filtre**: obligatoire, il contient le nom ou le type des nœuds recherchés
- **prédicat**: optionnel, il permet d'exprimer des conditions logiques lors de la sélection. Un prédicat assure un filtrage supplémentaire sur le(s) nœud(s) sélectionné(s).

Afin d'expliquer la syntaxe XPath, on utilise dans ce qui suit l'exemple suivant d'un document XML d'une entreprise qui propose des services en informatique:

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
	<service>
		<name>Web Design</name>
		<description>Custom website design </description>
		<price currency="DA">5000.00</price>
		<duration unit=”day”>14</duration>
	</service>
	<service>
		<name>Content Writing</name>
		<description>Content creation for websites </description>
		<price currency="DA">20000.00</price>
		<duration unit=”day”>7</duration>
	</service>
	<service>
		<name>SEO Consultation</name>
		<description>Search engine optimization analysis</description>
		<price>70000.00</price>
		<duration unit=”day”hour>3</duration>
	</service>
</services>
```

1.2.1 Les filtres

Les filtres représentent l'expression du(es) nœud(s) élément(s) à sélectionner. Parmi les filtres les plus générales sont:

- nom d'un noeud: permet de sélectionner tous les nœuds dont le nom est donné dans le filtre
- /: Sélection à partir du nœud racine. / permet aussi d'exprimer la relation parent/ enfant dans le chemin. Par exemple: A/B signifie qu'on sélectionne B qui est le descendant (fils) de A.
- //: Sélection de nœuds du document à partir du noeud courant peu importe leur position dans l'arbre
- @: Sélection des attributs
- . : Sélectionne le nœud actuel
- .. : Sélectionne le parent du noeud courant

Il est également possible d'exprimer des filtres en fonction du type de nœuds: text() pour le nœud Texte, comment() pour le nœud commentaire et node() qui exprime tous les types de nœuds. Voici des exemples de l'application des filtres:

Expression XPath	**Description**
services	Sélection de tous les nœuds avec le nom "services"
/services	Sélection de l'élément racine services. Le chemin ici est absolue puisqu'il commence avec /
/services/service	Sélectionne tous les éléments service qui sont des enfants de services
//service	Sélectionne tous les éléments service, peu importe où ils se trouvent dans le document
services//service	Sélectionne tous les éléments service qui sont des descendants de services peu importe où ils se trouvent dans le document
//@unit	Sélectionne tous les attributs qui sont nommés unit
*	Correspond à n'importe quel nœud
@*	Correspond à n'importe quel d'attribut
/services/*	Sélection de tous les nœuds enfants de l'élément services
//*	Sélectionne tous les éléments dans le document
//service/name | //service/price	Sélection de tous les nom ET les prix de tous les éléments service
//name | //price	Sélection de tous les nom ET les prix dans le document

Table 1. Exemples d'expression générales XPath

1.2.2 Les prédicats

Les prédicats sont des expressions logiques permettant d'exprimer des conditions sur la sélection des nœuds. Il suffit d'utiliser des crochets pour exprimer le prédicat après un filtre. La table 2 donne des exemples d'expression XPath avec des prédicats. Il faut noter qu'il est possible d'utiliser des fonctions dans les prédicats comme par exemple la fonction last() qui donne les nœuds feuilles et position() qui donne la position du nœud dans l'arbre.

Expression XPath	**Description**
/services/service[2]	Sélectionne le deuxième élément service qui est l'enfant de l'élément services
/services/service[last()]	Sélectionne le dernier élément service qui est l'enfant de l'élément services
/services/service[position()<3]	Sélectionne les premiers deux services qui sont les enfants de services
//price[@currency]	Sélectionne tous les éléments prix qui ont un attribut nommé "currency"
//duration[@unit="hour"]	Sélectionne tous les éléments "duration"qui ont un attribut unit avec la valeur 'hour'
//duration[@*]	Sélectionne tous les éléments duration qui ont n'importe quel attribut
/services/service[price>10000]	Sélectionne tous les éléments service de l'élément services qui ont un élément price d'une valeur supérieure à 10000
/services/service[price>10000]/ description	Sélectionne les éléments description des éléments service de l'élément services qui ont un élément price d'une valeur supérieure à 10000

Table 2. Exemples d'expression XPath avec des prédicats

1.2.3 Les axes

Dans l'expression de chemin, l'axe est optionnel et il est par défaut "child" ce qui signifie que la recherche se fait à partir du nœud courant exprimé par le chemin vers ses nœuds enfants. L'axe child est l'axe le plus simple à utiliser pour exprimer la sélection de nœuds. Selon le type de données et le nombre de nœuds, il est parfois utile de faire des recherches avec des axes particuliers, comme par exemple du nœud courant vers ses

nœuds frères gauches ou droites (siblings) ou du nœud courant vers ses ascendants (ascendants).

Parmi les axes on peut citer les suivants: parent, ancestor, descendant, descendant-or-self preceding , preceding-sibling, following, following-sibling, etc. La table 3 donne quelques exemples d'expressions XPath utilisant différents axes.

Expression XPath	**Description**
ancestor::service	Sélectionne tous les noeuds service ascendants du noeud courant
child::service	Sélectionne tous les noeuds service qui sont des enfants du noeud courant
descendant::service	Sélectionne tous les noeuds service descendants du noeud courant
attribute::unit	Sélectionne l'attribut unit du noeud courant
attribute::*	Sélectionne tous les attributs du noeud courant
child::text()	Sélectionne tous les noeuds textes de l'élément courant
child::node()	Sélectionne tous les enfants de l'élément courant
child::*	Sélectionne tous les éléments enfants du noeud courant

Table 3. Exemples d'expression XPath avec différents axes

1.3 Opérateurs et fonctions

Les quatres types de données possibles dans une expression XPath sont: le type numérique, booléen, chaîne de caractères et ensemble de nœuds. Les principaux opérateurs utilisés pour manipuler ces données sont:

- **Les opérateurs de comparaison**: <, <=, >, >=, =, != (différent de)
- **Les opérateurs arithmétiques**: +, -, *, div (division), mod (modulo)
- **Les opérateurs logiques**: or, and, not
- **Les opérateur d'union d'ensembles de noeuds**: |

XPath propose également certaines fonctions pour faciliter l'utilisation des expressions dans un langage hôte. Par exemples:

Les fonctions de sélection: par exemple la fonction:

- **node()** qui permet de sélectionner tous les nœuds enfants d'un élément (sous-éléments, attributs, etc). Par exemple, /services/service/node()
- **text()** qui sélectionne la valeur atomique d'un nœud (le contenu texte sans balises).

Par exemple, /services/service/name/text()

Les fonctions de manipulation des booléens: comme par exemple la fonction :

- **not(expression)** qui retourne la négation de l'expression logique passée en paramètre.

Les fonctions de manipulation des numériques: telles que la fonction:

- **sum(expression)** qui retourne la somme des valeurs numériques des nœuds sélectionnés par l'expression passée en paramètre
- **avg(expression)** qui retourne leur moyenne.

Les fonctions de conversions de types:

- **boolean(), string() et number()** qui permettent de convertir le résultat respectivement en booléen, chaîne de caractères ou un nombre. Dans le cas d'impossibilité de conversion de type, ces fonctions retournent la valeur NaN.

Les fonctions de manipulation de nœuds: par exemple la fonction :

- **count(expression)** qui retourne le nombre de nœuds de l'expression passée en paramètre.

Les fonctions de manipulation de chaînes de caractères: par exemple la fonction:

- **concat(chaine1, chaine2, …)** qui permet la concaténation des chaînes de caractères passées en paramètres; **contains(chaine1, chaine2)** qui vérifie si chaine1 contient chaine2
- **string-length(chaine)** qui retourne la longueur de la chaîne passé en paramètre;
- **starts-with(chaine1, chaine2)** qui vérifie si une chaîne de caractères chaine1 commence par une sous-chaîne chaine2

Les fonctions de date et d'heure :

- current-date() : retourne la date courante sous forme de chaîne de caractères YYYY-MM-DD.
  current-time() : retourne l'heure courante sous forme de chaîne de caractères HH:MM:SS.
- date() : convertit une chaîne de caractères en une valeur de date.

2. XSLT

Pour publier un document XML en réseau, il est souvent recommandé de publier avec la DTD ou le schéma pour décrire ses balises ainsi que le XSL ou XSLT pour adapter le format de sa présentation.

XSL (eXtended Style Langage) est un langage de style pour les documents XML qui permet plus particulièrement de convertir les données XML en un autre format plus lisible et

exploitable tel que HTML. XSL regroupe deux langages: XSLT (XML Stylesheet Language Transformation) qui est un langage de transformation simple de documents XML en un autre format tels que HTML et XSL-FO (XML Stylesheet Language - Formatting Objects) qui est un langage de mise en page de document.

Le langage XSLT est basé sur un ensemble de règles de transformation qui permettent de sélectionner des données d'un XML pour pouvoir les afficher par exemple à travers une page HTML. Une feuille de style XSLT est un document XML utilisant des balises spécifiques ayant l'espace de noms **xsl**. Elle se compose d'un élément racine **<xsl:stylesheet>** et les éléments de contenu qui représentent les règles de transformation **<xsl:template>.** La figure 6 donne un aperçu sur le lien entre une page HTML, un document XML et sa feuille de style XSLT.

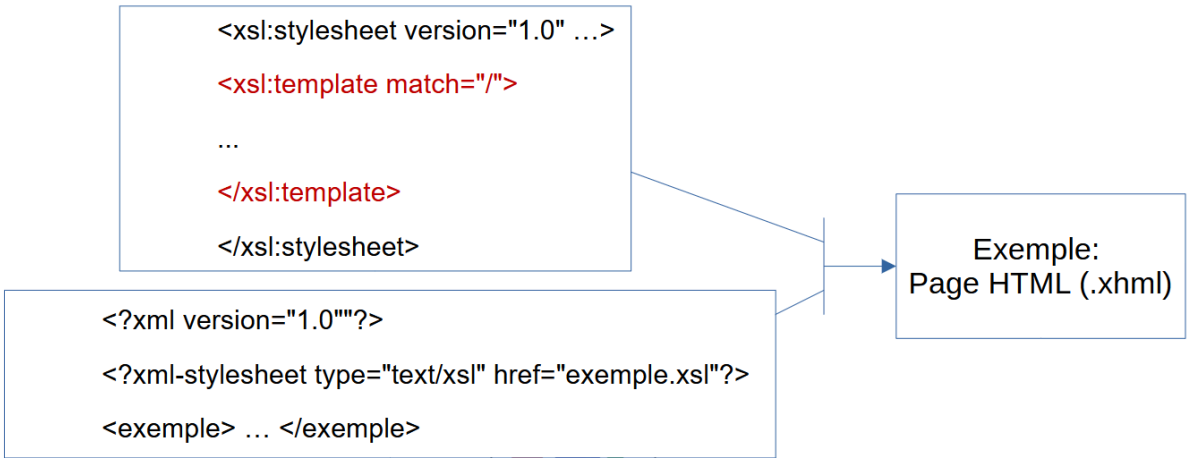


Figure 6. XML, XSLT et HTML.

2.1 Les règles XSLT

Il est possible d'écrire des règles différentes selon l'objectif de la transformation tel que l'exportation de données XML et leur affichage en HTML ou la sélection de quelques données XML pour un affichage plus personnalisé. Par exemple, afin de transformer tous les éléments d'un document XML en HTML, il faut définir pour chaque balise XML une règle qui traduit cette balise et son contenu. On commence par exemple par la définition d'une règle qui gère la racine. Cette règle doit produire la structure complète de la page HTML, autrement dit, elle doit produire les balises "html" "head" et "body". Ensuite, la ou les règles suivantes doivent être appliquées selon ce que vous voulez obtenir comme résultat d'affichage.

En général, une règle peut être appliquée pour: l'extraction de données, la génération de texte, la suppression, la duplication de contenu, le tri et/ou l'appel d'autres règles. La règle doit mentionner les nœuds surlequels elle va être appliquée: à partir de la racine (/) ou bien des nœuds spécifiés à travers une expression XPath.

<xsl:template> est utilisé pour définir une règle. La syntaxe générale d'une règle XSLT est la suivante:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
 …
 </xsl:template>
</xsl:stylesheet>
```

<xsl:stylesheet indique le début de la définition des règles. **xsl:template** est la déclaration d'une règle avec les principaux attributs suivants:

- **match**: la règle est appliquée sur la sélection des nœuds exprimée à travers XPath
- **name**: nom de la règle, souvent utilisé dans le cas d'une invocation de la règle par la balise <call-template>
- **priority**: priorité appliquée en cas de conflit entre deux règles

La table 4 illustre l'ensemble de commandes (balises XSL) les plus importantes permettant de décrire les règles.

Commande XSL	**Description**
<xsl:for-each>	sélectionner tous les éléments d'un ensemble de nœuds préciser à travers une expression XPath passée comme paramètre
<xsl:value-of>	extraire la valeur d'un élément et l'afficher
<xsl:sort>	trier les éléments
<xsl:if>	condition sur le contenu XML
<xsl:choose>, <xsl:when> et <xsl:otherwise>	utilisées pour exprimer des conditions multiples

Table 4. Commandes XSL pour décrire les règles

2.2 Exemple XSLT

Reprenant l'exemple XML de services d'une entreprise de la section 1.2 de ce chapitre sur lequel on applique une feuille de style services.xsl comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="services.xsl"?>

<services>
      <service>
            <name>Web Design</name>
            <description>Custom website design </description>
            <price currency="DA">5000.00</price>
```

```
            <duration unit=”day”>14</duration>
      </service>
      <service>
            <name>Content Writing</name>
            <description>Content creation for websites </description>
            <price currency="DA">20000.00</price>
            <duration unit=”day”>7</duration>
      </service>
      <service>
            <name>SEO Consultation</name>
            <description>Search engine optimization analysis</description>
            <price>70000.00</price>
            <duration unit=”hour”>3</duration>
      </service>
</services>
```

Un exemple d’une feuille de style XSLT qui affiche dans une page HTML le nom et la description des services de l’entreprise triés par ordre alphabétique est la suivante:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
   <html> <body> <h2>Les services</h2>
     <table border="1">
      <tr>  <th> Nom du service</th> <th>Description</th> </tr>
       <xsl:for-each select="services/service">
           <xsl:sort select="nom"/>
             <tr> <td><xsl:value-of select="nom"/> </td>
      <td><xsl:value-of select="description"/></td>
       </tr>
        </xsl:for-each>
      </table>
      </body> </html>
</xsl:template>
</xsl:stylesheet>
```

Pour exprimer des conditions, il suffit d’utiliser xsl: if dans l’exemple précédent comme suit:

```
 <xsl:template match="/">
    <xsl:for-each select="services/service">
          <xsl:if test="price &gt; 10000">
              …
          </xsl:if>
```

Dans le cas de multiple conditions, on peut utiliser la commande **choose** comme suit:

```
 <xsl:template match="/">
    <xsl:for-each select="services/service">
<xsl:choose>
      <xsl:when  test="price  &gt;  10000"> …  un  premier  affichage  ...
</xsl:when>
<xsl:otherwise> ... un autre affichage …</xsl:otherwise>
 </xsl:choose>
```

3. Processeurs et interfaces de programmation pour XML

Dans cette section, on traite les principaux modèles de programmation utilisés pour traiter et manipuler des documents XML à partir d'un langage évolué. En effet, en tant que standard de communication de données sur le réseau, XML bénéficie d'une large galaxie de bibliothèques et framework intégrés à plusieurs outils et langages de programmation existants. Un programme traitant des documents XML contient généralement deux parties: l'analyse du document (parsing) afin de récupérer les données à partir du fichier XML, et la partie traitement des données pour rechercher, ajouter, modifier des données d'un fichier XML.

Une application traitant des documents XML doit se baser sur un «Processeur XML» qui facilite l'analyse et le traitement d'un ou plusieurs documents XML. Un processeur permet la décomposition d'un document XML pour faciliter la récupération des éléments, attributs, les entités, etc. Il peut faire appel à des bibliothèques ou outils permettant de vérifier si le document XML est bien formé et valide à l'aide par exemple d'une DTD ou un XSD puis faire une représentation interne des informations (via une structure de données). Il existe deux types de modèles de programmation XML: un processeur basé sur un arbre ou DOM et un processeur basé sur un flux (document streaming) ou SAX. On décrira dans ce qui suit ces deux modèles de programmation puis on présente un exemple d'une interface de programmation récente permettant de manipuler des documents XML en se basant sur ces modèles de programmation.

3.1 Processeur basé sur DOM et SAX

DOM (Document Object Model) est une norme W3C pour représenter et stocker le document XML dans un arbre d'objets. L'utilisation d'une structure d'arbre, ou DOM, pour représenter un document XML en mémoire permet de faciliter le parcours, la modification et la transformation du document. L'avantage de cette structuration est qu'elle est indépendante des plates-formes et des langages de programmation. Plusieurs interfaces de programmation ont été conçues en se basant sur DOM. Même si ce type de processeur est simple à utiliser, il est généralement très coûteux en termes d'utilisation de la mémoire, notamment dans le cas d'un ensemble de gros documents XML à traiter.

Le second type de processeurs est le processeur **SAX (Simple API for XML)** qui est basé sur la gestion des événements. Dans SAX, les données de document XML ne sont pas sauvegardées en mémoire (comme c'est le cas de DOM). Il utilise le principe d'événements codés en fonction du besoin d'extraction ou de manipulation de données. Ces événements sont émis lors de la lecture séquentielle du flux qui contient le document XML. SAX nécessite moins de ressources mémoire mais il est moins pratique lors de la manipulation du document puisque les données ne sont pas conservées en mémoire.

Le choix entre SAX et DOM dépend alors de la taille des fichiers XML et du type de manipulation dans le programme. On utilise souvent le modèle DOM qu'on a besoin de faire différentes navigations et types de parcours du document, qu'on a de petits

documents (<1 Mo) ou des documents peu structurés. On préfère utiliser SAX qu'on a des documents XML bien structurés, pour le filtrage et l'extraction de données. En effet, dans le cas de plusieurs modifications de données d'un document, le nombre d'instructions devient élevé.

3.2 API Stax

Stax (Streaming API for XML) est une interface de programmation en Java pour lire et écrire des documents XML. Elle a été introduite depuis Java 6.0 et considérée comme API de haut niveau par rapport à SAX et DOM puisqu'elle facilite le traitement des documents XML d'une manière performante. Elle se base sur le modèle SAX en représentant un document sous la forme d'un ensemble d'événements qui sont fournis à la demande de l'application. Afin d'améliorer la performance du modèle SAX, Stax propose deux API pour parcourir, lire et écrire un document XML: Curseur (Cursor) qui gère des événements codés sur un entier pour chaque type d'élément rencontré et Itérateur d'événement (Event Iterator) utilise un objet appelée XMLEvent afin de gérer les évenements associés aux éléments trouvés d'un document. La bibliothèque Java a considéré pour utiliser Stax est `javax.xml.stream`.

On expliquera dans ce qui suit l'**API Itérateur d'événement (Event Iterator)** qui est la plus facile à utiliser puisque toutes les données sont sauvegardées dans un objet **XMLEvent**. L'API propose également deux objets **XMLEventReader** pour lire un événement et **XMLEventWriter** pour insérer des données dans l'XML. Pour simplifier la gestion des événements, Stax propose des fabriques (factories) pour construire des objets: **XMLInputFactory**, **XMLOutputFactory** et **XMLEventFactory** permettant de créer des événements, créer des objets pour les écrire dans le document XML et parcourir les événements créés respectivement.

3.2.1 XMLEventReader et XMLEventWriter

XMLEventReader et XMLEventWriter sont des interfaces qui héritent de Iterator et proposent des méthodes pour parcourir un document XML et obtenir l'événement courant (début d'un élément, attribut, etc) de type XMLEvent. Elles proposent les deux principales méthodes suivantes:

- **nextEvent()** qui retourne le prochain événement lu XMLEvent
- **hasNext()** vérifie si le document contient encore du texte et donc encore un événement à traiter.

3.2.2 L'interface XMLEvent

Lors du parcours et la lecture d'un document XML, des objets XMLEvent sont retournés. Un XMLEvent peut être par exemple:

- **StartDocument** et **EndDocument**: indiquent le début et fin du document
- **StartElement** et **EndElement**: indiquent le début et fin d'un élément (balise ouvrante et balise fermante)

- **Attribute**: indiquent un attribut. les attributs peuvent être aussi récupérés par un événement StartElement
- **Characters**: une suite de caractères tel qu'une section de type CData ou des séparateurs
- **DTD**: les informations sur la DTD
- **Comment**: il s'agit d'un commentaire détecté

Chaque type de **XMLEvent** possède des propriétés. Par exemple, StartElement qui hérite de l'interface XMLEvent propose des méthodes pour obtenir les attributs de l'élément (Iterator getAttributes()) ou rechercher un attribut par son nom (Attribute getAttributeByName()),

3.2.3 Exemple de lecture d'un document XML

Prenant l'exemple de services d'une entreprise simplifié (services.xml):

```
<services>
	<service>
		<name>Web Design</name>
		<price currency="DA">5000.00</price>
	</service>
	<service>
		<name>Content Writing</name>
		<price currency="DA">20000.00</price>
	</service>
	<service>
		<name>SEO Consultation</name>
		<price>70000.00</price>
	</service>
</services>
```

Il faut suivre les étapes suivantes pour pouvoir le lire via l'API Stax:

1) Définir la classe Service pour stocker les entrées du document XML (services.xml)

```
public class Service{
   private String name, price, currency;
   // …. getters and setters
   // affichage
   @Override
   public String toString() {
       return "Service [name=" + name+ ", price=" + price+"]";
   }
}
```

2) Définir la classe qui analyse et traite les données de l'XML (Parser.java). Cette classe permet de lire le document XML et crée une liste d'objets Service à partir des entrées du fichier XML

```
// les import pour les utilitaires utilisés

import java.io.*;
import java.util.*;
```

```
import javax.xml.stream.*;
import javax.xml.stream.events.*;
public class Parser {

       // Les cas des éléments(balises) du document XML
       static final String SERVICE= "service";
       static final String NAME= "name";
       static final String PRICE= "price";
       static final String CURRENCY= "currency";


      // lire les services à partir du fichier XML et retourner une liste

       public List<Service> readServices(String servicesFile) {
      // Créer une liste vide de services
       List<Service> services = new ArrayList<Service>();

       try {
         // 1) créer une nouvelle fabrique
         XMLInputFactory inputFactory = XMLInputFactory.newInstance();

         // 2) Définir un nouveau eventReader
         InputStream in = new FileInputStream(servicesFile);
       XMLEventReader eventReader =
       inputFactory.createXMLEventReader(in);

         //3) lire le document XML
       Service service= null;

       while (eventReader.hasNext()) {

                   XMLEvent event = eventReader.nextEvent();
                   if (event.isStartElement()) {
                   StartElement startElement = event.asStartElement();
                   String elementName=
                   startElement.getName().getLocalPart();

              switch (elementName) {
                 // traiter dans ce qui suit chaque élément possible

                    case SERVICE:
                          service= new Service();
                          event = eventReader.nextEvent();
                          break;

                         case NAME:
                         event = eventReader.nextEvent();
                         service.setName(event.asCharacters().getData());
                         break;

                         case PRICE:
                         event = eventReader.nextEvent();
                         service.setPrice(event.asCharacters().getData());
                           // On lit les attributs de l'élément
                           Iterator<Attribute> attributes =
                           startElement.getAttributes();
```

```
                        while (attributes.hasNext()) {
                        Attribute attribute = attributes.next();
                          if (attribute.getName().toString().
                                equals(CURRENCY)) {
                             service.setCurrency(attribute.getValue());
                          }
                        }
                        break;
                  } // fin switch
                }
               // ajout du service à la liste
               if (event.isEndElement()) {
                     EndElement endElement = event.asEndElement();
               if (endElement.getName().getLocalPart().equals(SERVICE)) {
                     services.add(service);
               }
             }
           }
         } catch (FileNotFoundException | XMLStreamException e) {
                     e.printStackTrace();
       }
         return services;
       }
}
```

3) Tester la classe Parser: créer une classe contenant la méthode main() pour appeler la classe Parser et afficher la liste récupérée des services à partir du fichier XML

```
import java.util.List;
public class Main {
   public static void main(String args[]) {
       Parser read = new Parser ();
       List<Service> services= read.readServices("services.xml");
       for (Service service: services) {
           System.out.println(service);
       }
   }
}
```

3.2.4 Exemple d'écriture d'un document XML

On souhaite créer un fichier XML (services2.xml) à partir de notre programme Java qui contient ce qui suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<service>
<name>Web Design</name>
<price>5000.00</price>
</service>
```

Pour cela, il faut suivre les étapes suivantes:

1) Créer la classe Writer:

Une classe qui permet de produire un document XML. Il faut savoir que StaX ne fournit pas un moyen pour formater automatiquement un document XML. Il est donc recommandé d'ajouter les fin de lignes et des tabulations lors de la création du document.

```
// les import utilisés

import java.io.FileOutputStream;
import javax.xml.stream.XML*;
import javax.xml.stream.events.*;

public class Writer {

   private String servicesFile; // notre fichier XML à créer



   public void setFile(String servicesFile) {
       this.servicesFile = servicesFile;
   }

   // méthode pour écrire des services sur le fichier

   public void saveService() throws Exception {

      // 1) créer une fabrique XMLOutputFactory

       XMLOutputFactory outputFactory = XMLOutputFactory.newInstance() ;

      // 2) créer un XMLEventWriter

       XMLEventWriter eventWriter = outputFactory
         .createXMLEventWriter(new FileOutputStream(servicesFile));

       // 3) créer la fabrique EventFactory

       XMLEventFactory eventFactory = XMLEventFactory.newInstance();
       XMLEvent end = eventFactory.createDTD("\n");

       // créer et écrire le prolog

       StartDocument startDocument = eventFactory.createStartDocument();
       eventWriter.add(startDocument);

       // créer un service et l'ajouter

       StartElement serviceStartElement =
       eventFactory.createStartElement("", "", "service");
       eventWriter.add(serviceStartElement);
       eventWriter.add(end);

       // écriture les différents noeuds avec notre méthode createNode

       createNode(eventWriter, "name", "Web Design");
       createNode(eventWriter, "price", "5000.00");
```

```
        // créer et écrire le dernier élément
         eventWriter.add(eventFactory.createEndElement("", "", "service"));
         eventWriter.add(end);
         eventWriter.add(eventFactory.createEndDocument());
         eventWriter.close();

        }

        // méthode pour créer un noeud du document xml

         private void createNode(XMLEventWriter eventWriter, String name,
         String value) throws XMLStreamException {

            XMLEventFactory eventFactory = XMLEventFactory.newInstance();
            XMLEvent end = eventFactory.createDTD("\n");
            XMLEvent tab = eventFactory.createDTD("\t");
            // ...

         // créer un élément de début et l'ajouter comme événement

         StartElement sElement =
         eventFactory.createStartElement("", "", name);
         eventWriter.add(tab);
         eventWriter.add(sElement);

         // créer le contenu

         Characters characters = eventFactory.createCharacters(value);
         eventWriter.add(characters);

         // créer le noeud de fin

         EndElement eElement = eventFactory.createEndElement("", "", name);
         eventWriter.add(eElement);
         eventWriter.add(end);
    }
}
```

3) Tester la classe Writer: créer une classe contenant la méthode main() pour appeler la classe Writer permettant de créer un fichier XML

```
import java.util.List;

public class Main2 {
   public static void main(String args[]) {
        Writer writer = new Writer();
        writer.setFile(services2.xml);
        try {
            writer.saveService();
        } catch (Exception e) {
            e.printStackTrace();
        }
   }
```

}

4. Applications XML: RDF et SVG

XML a été utilisé dans différents domaines tels que la description des ressources Web à l'aide de RDF, des images SVG ou encore des formules mathématiques en MathML. Nous décrivons dans ce qui suit deux exemples de son application, notamment à travers RDF et SVG.

4.1 RDF

RDF, ou Resource Description Framework, est un langage utilisé pour décrire et échanger des ressources sur le Web. Chaque ressource est décrite par le triplet sujet, prédicat et objet. Le sujet est la ressource à décrire, par exemple un document à partager en réseau. Il est défini par un URI (identifiant), par exemple `www.univ-mosta.dz/mydocuments/`. Le prédicat est le type de propriété de cette ressource identifié par un URI, par exemple le nom du document "`mydocument`". Enfin, l'objet est la valeur de la propriété qui représente une donnée ou une ressource, par exemple "`my document`" ou `www.univ-mosta.dz`. Cette structure permet de garantir une certaine sémantique facilitant l'interprétation et l'échange des données et l'intégration des connaissances.

Pour décrire des ressources en RDF, il est possible d'utiliser XML. Ci-après un exemple permettant de décrire des produits vendus en ligne: une imprimante et un clavier :

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:produits="http://www.magasin.dz/produit#">
<rdf:Description rdf:about="http://www.magasin.dz/produit/Imprimante">
             <produit:nom>Imprimante laser</produit:nom>
 <produit:pays>Chine</produit:pays>
 <produit:entreprise>Canon</produit:entreprise>
 <produit:prix>23000</produit:prix>
 <produit:série>TS 5050</produit:série>
</rdf:Description>
<rdf:Description rdf:about="http://www.magasin.dz/produit/Clavier">
             <produit:nom> Clavier gamer </produit:nom>
 <produit:pays>Japon</produit:pays>
 <produit:entreprise>VGN</produit:entreprise>
 <produit:prix>8000</produit:prix>
 <produit:série>cr123</produit:série>
</rdf:Description>
…
</rdf:RDF>
```

Dans cet exemple, on a commencé par définir la racine `<rdf:RDF>` puis l'espace de noms `xmlns:rdf` qui spécifie les éléments avec le préfixe `rdf` à partir de

"http://www.w3.org/1999/02/22-rdf-syntax-ns#". Ensuite, on spécifie l'espace des noms de nos ressources à définir par exemple "http://www.magasin.dz/produit#" en utilisant le mot clé xmlns.

Afin de définir un élément, qui est dans notre exemple un produit, il faut utiliser <rdf:Description> permettant de définir la ressource en utilisant également son attribut rdf:about. La description d'un élément inclut la description des sous éléments à travers des balises avec le préfixe de l'élément. Dans notre exemple, une imprimante est définie par le nom, le pays de fabrication, l'entreprise, le prix et la série.

4.2 SVG

SVG ou Scalable Vector Graphics est un langage de définition des graphiques vectoriels en deux dimensions basé sur XML. Une image SVG est décrite en utilisant des éléments graphiques tels que les lignes, les courbes, du texte, etc. L'avantage de SVG est la maîtrise de la qualité de l'image en cas de redimensionnement car elle n'est pas définie par des pixels, comme c'est le cas des images PNG par exemple.

Pour définir le contenu d'une image SVG, on utilise des balises prédéfinies. Voici un exemple:

```
<?xml version="1.0" standalone="no"?>

<svg xmlns="http://www.w3.org/2000/svg" width="600"
     height="800" viewBox="0 0 600 800">
<rect x="0" y="440" width="200" height="400" fill="808080"/>

<circle cx="550" cy="100" r="50" fill="#8B4513"/>
<polygon points="0,400 200,200 400,400" fill="#808080"/>

<text x="300" y="750" text-anchor="middle"
font-family="Arial" font-size="24" fill="black"> image ! </text>

</svg>
```

Dans cet exemple, la balise <svg> permet de définir l'image en précisant l'espace des noms, la longueur hight et la largeur width de l'image, et les coordonnées du point de départ des axes x et y. Les sous-éléments de svg sont les représentations graphiques de l'image qu'on souhaite dessiner. Dans cet exemple, on a dessiné un rectangle <rect>, un cercle <circle>, un polygone <polygon> et un texte <text> "image!".

Dans toutes les formes, l'attribut fill permet de définir la couleur de remplissage, x et y les coordonnées de la position, et width et height la largeur et la longueur. Dans un cercle, il faut définir le rayon r, dans un polygone ses sommets, et dans un texte sa taille, police et position dans l'image.

5. Galaxie XML: en savoir plus

En tant que standard, XML possède plusieurs applications, certaines sont toujours utilisées et d'autres ne sont plus actuelles. Plusieurs langages, bibliothèques et structures ont été proposés pour traiter et manipuler des documents XML dans différents domaines d'application. On peut citer par exemple XForms qui permet la saisie de formulaires avec XML, Xpointer permettant de définir des pointeurs internes à un document et Xlink pour créer des liens entre des documents XML. XML a été également utilisé pour définir des protocoles d'accès au services Web comme SOAP, de décrire des services Web avec WSDL, et pour permettre de syndiquer le contenu d'un site Web avec RSS. Nous citons dans ce qui suit deux exemples de langages Xlink et WSDL.

5.1 Xlink

XLink (XML Linking Language) permettant de créer des hyperliens et gérer des liens hypertextes entre différentes ressources. Il peut servir aussi dans le cas de création de liens vers des ressources externes non XML. XLink permet de faciliter la navigation et l'interconnexion entre les ressources. Il est compatible avec d'autres technologies XML comme XSLT. Voici un exemple simple d'utilisation de XLink dans un document XML :

```
<services xmlns:xlink="http://www.w3.org/1999/xlink">
<service xlink:type="simple" xlink:href="http://example.com/service1">
    <name>Service 1</name>
    <price>2000</price>
  </service>
  <service xlink:type="simple" xlink:href="http://example.com/service2">
    <name>Service 2</name>
    <price> 3000</price>
  </service>
</services>
```

Dans cet exemple, nous définissons d'abord l'espace de noms XLink `xmlns:xlink="http://www.w3.org/1999/xlink"` afin de pouvoir créer des liens. Ensuite `xlink:type` et `href` spécifient le type de lien ainsi que l'URI de la ressource.

5.2 WSDL

WSDL (Web Services Description Language) est un langage pour la description des services Web et d'échange de données en réseau en utilisant la structure XML. La description des services se fait à travers la définition des opérations, des messages échangés, et les protocoles de communication tels que HTTP et SOA. L'un des avantages de WSDL est son interopérabilité, ou sa capacité de définir des services qui sont indépendants du langage de programmation, le système et l'infrastructure utilisés.

Ci-après un exemple simple d'utilisation de WSDL pour définir un service d'addition de

deux nombres décimaux en ligne en utilisant le protocole SOAP. Le document WSDL suivant permet de décrire un service nommé Calculator disponible sur le port Port et l'adresse http://example.com/calculator propose une opération Add permettant d'additionner deux nombres décimaux. Il décrit les types de données utilisées pour les paramètres et les opérations utilisées ainsi que les informations nécessaires pour le protocole SOAP, notamment le binding qui permet de lier le service à l'hôte à travers son adresse et son port.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://example.com/calculatorService"
             targetNamespace="http://example.com/calculatorService">

    <!-- Définition du service -->
    <service name="Calculator">
        <port name="Port" binding="tns:Binding">
            <soap:address location="http://example.com/calculator"/>
        </port>
    </service>

    <!-- Définition des types des opérandes et du résultat-->
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="AddRequest">
                <complexType>
                    <sequence>
                        <element name="operand1" type="decimal"/>
                        <element name="operand2" type="decimal"/>
                    </sequence>
                </complexType>
            </element>
            <element name="AddResponse">
                <complexType>
                    <sequence>
                        <element name="result" type="decimal"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <!-- Définition des opérations -->
    <portType name="CalculatorPortType">
        <operation name="Add">
            <input message="tns:AddRequest"/>
            <output message="tns:AddResponse"/>
        </operation>
    </portType>

    <!-- Définition du binding SOAP -->
    <binding name="CalculatorBinding" type="tns:CalculatorPortType">
     <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <operation name="Add">
      <soap:operation soapAction="http://example.com/calculator/Add"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
    </operation>
  </binding>
</definitions>
```

Pour résumer, en tant que standard de représentation des données semi-structurées, XML a donné lieu à une multitude de langages, structures et outils permettant de traiter ce format et de l'appliquer à divers domaines. L'un des aspects importants de XML est également sa gestion et son utilisation par les bases de données. Ce sera l'objectif du chapitre suivant.

6. Exercices

Exercice 1

Soit le document XML “centre.xml” qui contient des données d’un centre commercial

```
<?xml version="1.0" encoding="UTF-8"?>
<centre>

  <!--Liste des magasins -->
  <magasins>
    <magasin categorie="etranger" locataire="o1">
      <nom>HM </nom>
      <type> vetements adultes femme et homme</type>
      <reference> 23453344 </reference>
      <pays-origine>Espagne</pays-origine>
      <téléphone>+213 4455577</téléphone>
      <local>
        <numero>24</numero>
        <superficie>70 </superficie>
        <lot>120</lot>
      </local>
      <employé>
        <prénom>Mohamed</prénom>
        <nomFamille>Hadi </nomFamille>
        <fonction> responsable magasin</fonction>
      </employé>
      <employé>
        <prénom>Bouchra</prénom>
        <nomFamille>Amir</nomFamille>
        <fonction> responsable des ventes </fonction>
      </employé>
      <employé>
        <prénom>Ali</prénom>
        <nomFamille>Abdelhadi</nomFamille>
        <fonction> caissier </fonction>
      </employé>
    </magasin>
    <magasin categorie="local" locataire="o2">
      <nom>ilyes bijoux </nom>
      <type> accessoires femme et homme</type>
      <reference> 23453343 </reference>
      <pays-origine>Algérie</pays-origine>
      <téléphone>+213 4455555</téléphone>
      <local>
        <numero>12 </numero>
        <superficie>35 </superficie>
        <lot>130</lot>
      </local>
      <employé>
```

```
      <prénom>Amine</prénom>
      <nomFamille>Dani </nomFamille>
      <fonction> responsable magasin</fonction>
    </employé>
    <employé>
      <prénom>Ismail</prénom>
      <nomFamille>Amir</nomFamille>
      <fonction> responsable des ventes </fonction>
      <fonction> caissier </fonction>
    </employé>
  </magasin>
  <magasin categorie="etranger" locataire="o4">
    <nom>Nike </nom>
    <type> chaussures adultes et enfants</type>
    <reference> 23453340 </reference>
    <pays-origine>USA</pays-origine>
    <téléphone>+213 4455588</téléphone>
    <local>
      <numero>10</numero>
      <superficie>100 </superficie>
      <lot>40</lot>
    </local>
    <employé>
      <prénom>Imane</prénom>
      <nomFamille>Benali </nomFamille>
      <fonction> responsable magasin</fonction>
    </employé>
    <employé>
      <prénom>Amine</prénom>
      <nomFamille>Zakaria</nomFamille>
      <fonction> responsable des ventes </fonction>
    </employé>
  </magasin>
</magasins>

<!-- Liste des locataires -->
<locataires>
  <locataire idlocataire="o1">
    <nom>Salim</nom>
    <prénom>Salim</prénom>
    <adresse>
      <rue>20 rue des Martyis</rue>
      <codePostal>27000</codePostal>
      <ville>Mostaganem</ville>
      <pays>Algérie</pays>
    </adresse>
  </locataire>
  <locataire idlocataire="o2">
    <nom>Mohammed</nom>
    <prénom>Mohammed</prénom>
    <adresse>
      <rue>13 boulvard des Lions</rue>
      <codePostal>31000</codePostal>
      <ville>Oran</ville>
      <pays>Algérie</pays>
    </adresse>
```

```
  </locataire>
  <locataire idlocataire="o3">
    <nom>Amel</nom>
    <prénom>Amel</prénom>
    <adresse>
      <rue>3 rue des falaises</rue>
      <codePostal>16000</codePostal>
      <ville>Alger</ville>
      <pays>Algérie</pays>
    </adresse>
  </locataire>
</locataires>

<!-- Liste des matériels à louer -->

<matériels>
  <matériel idmatériel="m1" locataire="o3">
    <nom>comptoire de réception</nom>
    <caractéristiques>
      <caractéristique>bois</caractéristique>
      <caractéristique>fixation automatique</caractéristique>
      <caractéristique>barrière de sécurité</caractéristique>
    </caractéristiques>
  </matériel>
  <matériel idmatériel="m2">
    <nom>alarme surveillance magasin</nom>
    <caractéristiques>
      <caractéristique>catégorie supérieure</caractéristique>
      <caractéristique>digit code intégré</caractéristique>
    </caractéristiques>
  </matériel>
</matériels>

<!-- Liste des appartements du centre-->

<appartements>
  <appartement idAppartement="a1" comfort="F">
    <superficie>30</superficie>
    <prix>600</prix>
    <description>Un petit studio </description>
    <etage>1</etage>
    <enregistrement>
      <date>2022-05-05</date>
      <texte>problème d'humidité</texte>
    </enregistrement>
    <enregistrement>
      <date>2023-02-24</date>
      <texte>peinture effectuée</texte>
    </enregistrement>
  </appartement>
  <appartement idAppartement="a2" comfort="A" locataire=" o2">
    <superficie>120</superficie>
    <prix>1800</prix>
    <description>Grand appartement avec balcon</description>
    <etage>4</etage>
    <enregistrement>
```

```
        <date>2023-02-24</date>
        <texte>nettoyage effectuée</texte>
      </enregistrement>
    </appartement>
    <appartement idAppartement="a3" comfort="B" locataire="o4">
      <superficie>80</superficie>
      <prix>900</prix>
      <description> Appartement sans balcon.</description>
      <etage>4</etage>
    </appartement>
  </appartements>
</centre>
```

Exprimez les requêtes XPath suivantes:

1. Sélectionnez les numéros de téléphones disponibles.
2. Sélectionnez tous les locataires du centre (avec leurs sous arbres entiers).
3. Sélectionnez les noms des magasins du centre (uniquement la valeur atomique).
4. Sélectionnez les pays d'origine des locataires.
5. Afficher le nombre d'employés.
6. Afficher le nombre de personnes.
7. Sélectionnez l'identifiant du locataire du magasin Nike.
8. Sélectionner les informations sur ce locataire.
9. Sélectionner les références des magasins étrangers du centre commercial.
10. Sélectionnez les prix des appartements avec un niveau de confort B.
11. Sélectionnez les appartements qui ne sont pas encore loués.
12. Sélectionnez les biens (appartements et les matériels) qui ne sont pas encore loués.
13. Sélectionnez les biens (appartements et les matériels) qui ne sont pas loués par "o1".
14. Sélectionnez le nom et la deuxième caractéristique du premier matériel de la liste (uniquement la valeur atomique)
15. Sélectionnez les locataires qui ont loué un appartement.
16. Donnez la somme des prix des appartements

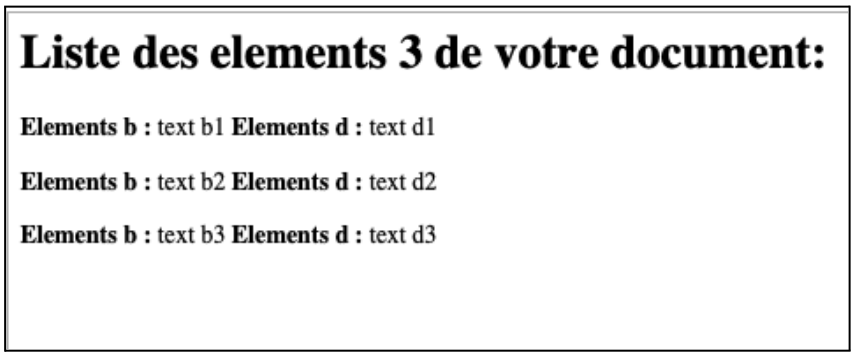
Exercice 2

Soit le document XML suivant:

```
<a>
  <aa>
         <b> text b1 </b>
         <c>
               <d> text d1 </d>
               <e kind="group"> 11 </e>
         </c>
  </aa>
  <aa>
        <b> text b2 </b>
```

```
        <c>
              <d> text d2 </d>
              <e kind="group"> 10 </e>
        </c>
  </aa>
  <aa>
        <b> text b3 </b>
        <c>
              <d> text d3 </d>
              <e kind="one"> 9 </e>
       </c>
  </aa>
</a>
```

Ecrire le programme XSLT qui permet de produire la page HTML suivante:



Exercice 3

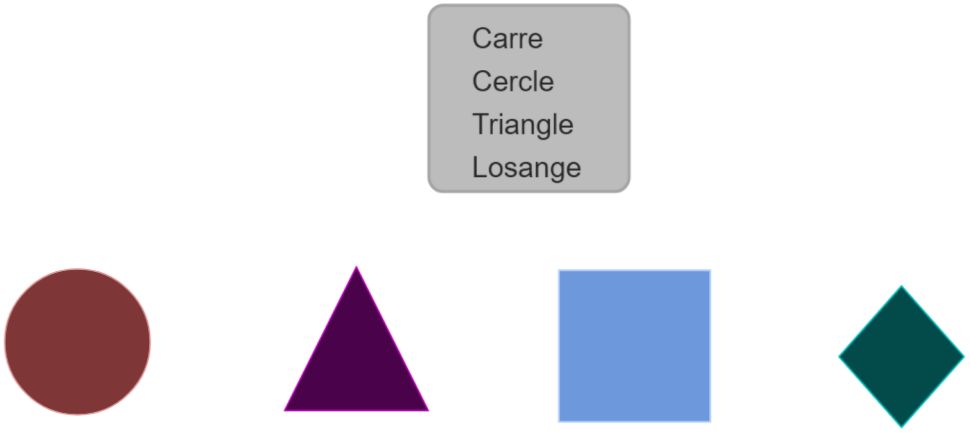
Soit le document XML suivant représentant le tableau d'affichages des vols d'un aéroport:

```
<?xml version="1.0" encoding="UTF-8"?>
<Departures>
  <flight>
    <Time>11:30</Time>
    <DeparturePoint>PARISCG</DeparturePoint>
    <Code>FR1613</Code>
    <Destination>PARIS Charle de Gaule</Destination>
    <Gate>10</Gate>
    <Status>AVAILABLE</Status>
  </flight>
  <flight>
    <Time>13:20</Time>
    <DeparturePoint>MadridASM</DeparturePoint>
    <Code>ES5453</Code>
    <Destination> Madrid (Adolfo Suárez Madrid-Barajas)</Destination>
    <Gate>02</Gate>
    <Status>DELAYED</Status>
  </flight>
</Departures>
```

Donnez la feuille XSLT qui permet de produire la page HTML contenant un tableau d'affichage des vols programmés.

Exercice 4

Donnez le programme qui permet à l'utilisateur de sélectionner une forme géométrique puis de l'afficher. Cette forme doit correspondre au format SVG.



Exercice 5

Donnez le code SVG permettant de définir et appliquer des filtres qui transforme une image en noir et balnc ou en couleurs plus intences et vifs. L'image peut être une ressource définie par son URL.

Par exemple:

Exercice 6

Soit un document XML qui inclut un ensemble de contacts:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<contacts>
<contact>
      <lastname>Mustapha</lastname>
```

```
      <firstname>Isac</firstname>
      <phone>010332435</phone>
</contact>
<contact>
      <lastname>Mohamed</lastname>
      <firstname>Otmane</firstname>
      <phone>010366635</phone>
</contact>
<contact>
      <lastname>Amine</lastname>
      <firstname>Islam</firstname>
      <phone>011112435</phone>
</contact>
</contacts>
```

Donnez le programme en Java qui utilise l'API Stax pour lire et afficher ce document XML.

Chapitre III: Base de données XML et XQuery

Afin de traiter et échanger des données semi-structurées telles que XML, il est souvent nécessaire d'utiliser des systèmes de gestion de base de données permettant de faciliter le traitement de ce type de données. Ce chapitre donne un aperçu sur les types de base de données XML et leur gestion ainsi que le langage de requêtes dédié à XML, XQuery.

1. Base de données XML et SGBD

On utilise généralement une base de données XML quand les données utilisées sont sous format XML et qu'on souhaite traiter et publier. Elle est également utilisée pour stocker des pages Web avec ses structures associées pour garantir la fonctionnalité de l'application dans le cas de dépendances multiples à des tierces frameworks.

Différents systèmes de gestion de base données (SGBD) et des techniques ont été proposés pour faciliter le traitement de données XML. La solution la plus simple consiste à transformer un XML en modèle relationnel. Cette solution est cependant coûteuse et pas toujours bijective. Selon la structure de l'XML, il est parfois nécessaire de mémoriser la hiérarchie entre les éléments, ce qui n'est pas possible avec un modèle relationnel. Néanmoins, cette solution reste envisageable dans le cas de trop peu de documents XML dont la structure est très simple. Une autre solution permet de considérer un document XML comme un attribut de la table relationnel (LOB, large object). Le problème avec cette solution est la difficulté d'exploiter l'XML et formuler des requêtes. Cette solution peut servir dans le cas d'une simple sauvegarde de document XML et dont l'utilisation ne nécessite pas un traitement ou une recherche approfondie de données.

En tant que standard, la plupart des systèmes de gestion de bases données proposent des extensions pour traiter, manipuler ou exporter les données en format XML. Les systèmes comme Oracle et SQL Server permettent de traiter directement des documents XML. Cette solution reste très pratique et flexible dans le cas des applications traitant des données structurées et semi-structurées. Il est également possible d'utiliser des bases de données dédiées uniquement à XML. Ces bases de données sont appelées des bases de données natives. C'est le cas par exemple de BaseX, eXist et Sedna.

Dans le cas de base de données natives, un SGBD possède souvent l'architecture fonctionnelle illustrée dans la Figure 7. Un SGBD-XML natif traite l'ensemble de document XML, ou forêt XML, en utilisant leur index qui est la racine de chaque document. La recherche se fait à l'aide de la formulation d'une requête XPATh ou XQuery par exemple. Le résultat est un ensemble de nœuds (arbre, sous arbre ou une forêt). La création et la modification des documents se fera à l'aide d'une requête de type XUpdate. L'avantage d'un SGBD XML natif est de permettre un traitement optimal des requêtes puisqu'il est

basé sur des recherches arborescentes optimisées. Il permet également une meilleure lisibilité des résultats des requêtes et des données XML, qui n'est pas le cas des SGBD non XML.

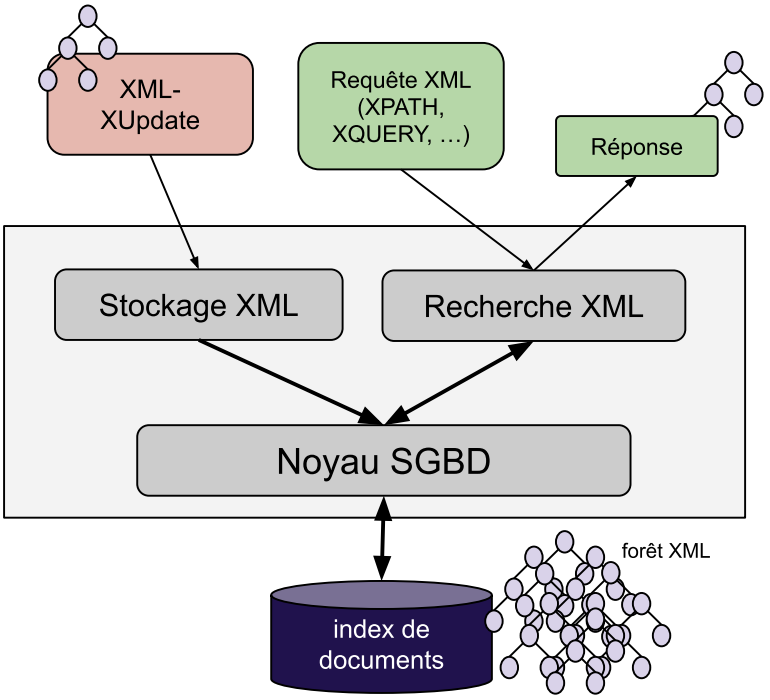


Figure 7. Principe de fonctionnement d'un SGBD-XML natif

2. Langages de requêtes

Différent langages ont été proposés pour formuler des requêtes XML tels que : Lorel, Quilt, XQL et XQuery.

2.1 Lorel

Lorel est l'un des premiers langages de requête pour les données semi-structurées. Il a été développé pour OQL (Object Query Language) afin de faciliter l'interrogation des éléments Parmi ses avantages, Lorel se distingue par une syntaxe simple, facilitant sa compréhension. Cependant, Lorel dépend de l'analyseur OQL et possède ainsi des fonctionnalités limitées.

2.2 XQL

XQL est un autre langage de requêtes pour les bases de données XML basé sur des expressions de chemin aligné sur les structures XML. Il prend en charge les jointures et certaines fonctions, facilitant la recherche et le tri dans les documents XML. Il utilise une syntaxe inspirée de XSL, permettant de sélectionner des éléments spécifiques de manière déclarative. Bien que ses expressions soient courtes et simples à utiliser, la sémantique peut parfois manquer d'intuitivité. Il possède également des fonctionnalités limitées par

rapport à l'expression de requêtes.

2.3 Quilt

Un autre langage qui est considéré comme très riche en termes de fonctionnalités est **Quilt**. C'est un langage fonctionnel permettant d'exprimer les requêtes sous forme d'expressions de plusieurs types incluant des expressions de chemin et des expressions utilisant plusieurs clauses, opérateurs et fonctions. Cependant, sa syntaxe est souvent considérée complexe. De plus, le traitement des requêtes Quilt peut poser des problèmes de performance en la comparant avec d'autres langages existants.

2.4 XQuery

Enfin, **XQuery** a été proposé comme un standard d'expression de requêtes XML qui est basé sur XPath pour l'expression de chemin dans les requêtes. Xquery est considéré comme un langage flexible et intégrant plusieurs fonctionnalités pour le traitement de différentes structures et types de données. Il peut également être facilement intégré avec d'autres langages et technologies. L'expression de requêtes avec XQuery sera l'objet du reste de ce chapitre.

3. Le langage XQuery

XQuery est le langage de requêtes pour XML défini et standardisé par le W3C. Il est utilisé pour les bases de données XML natives, les documents XML textuels (XQuery Text) ainsi que pour l'intégration de données virtuelles. XQuery permet de faire des projections sur les arbres XML de la base de données, sélectionner des nœuds ou des sous-arbres, appliquer des jointures d'arbres, etc. Il peut également permettre à l'utilisateur de créer ses propres fonctions intégrant des requêtes sur une base de données.

3.1 Syntaxe générale

Une requête XQuery est composée d'expressions et optionnellement de fonctions. La syntaxe générale d'une requête élémentaire est la suivante:

FOR $<var> **in** <forêt> **[, $**<var> **in** <forêt>**]** //itération

LET $<var> **:=** <sous-arbre> // assignation

WHERE <condition> // filtrage

RETURN <résultat> // construction

La première ligne (For) peut être utilisée pour accéder à un ou plusieurs document XML et sélectionner le(s) nœuds, l'arbre, ou la forêt sur laquelle la recherche sera effectuée. La recherche et l'expression de chemin se fait via XPath. Il faut par contre préciser le document XML en utilisant le mot clé doc. Par exemple:

```
doc("nomfichier.xml")/expressionXPath
```

Il est possible de mémoriser le résultat de la sélection dans une variable puis en utilisant **in** pour l'affectation. Cela est valable dans le cas de plusieurs sélections en utilisant une

multitude de variables séparés par des virgules. Les crochets sont utilisés dans la syntaxe pour exprimer la partie optionnelle et ils ne font pas partie de la syntaxe.

La deuxième ligne permet de définir des variables qui peuvent être utilisées pour exprimer des requêtes complexes. Une variable peut mémoriser un ensemble de nœuds (sous-arbre par exemple) ou bien le résultat du calcul ou de l'appel d'une fonction.

La troisième ligne permet d'appliquer un filtre supplémentaire sur la sélection en vérifiant une expression logique.

La dernière clause return, permet de retourner le résultat de la requête. C'est la seule clause **return** qui est obligatoire dans une requête XQuery. Le résultat est une valeur atomique, forêt (un ou plusieurs arbres), un nœud, un arbre ou un sous-arbre.

Lors de l'écriture d'une requête, les règles de syntaxe suivantes doivent être appliquées:

- Respecter le nommage de clauses et de variables en termes d'utilisation de la majuscule/minuscule puisque XQuery est sensible à la casse
- Les éléments, attributs et variables doivent avoir un nom valide en xml
- Il est possible d'exprimer des chaînes de caractères (`string`) avec les symboles `' '` ou bien `" "`
- Une variable est définie à l'aide d'un dollar qui précède le nom de la variable : `$nom_variable`
- un commentaire est défini comme suit: `(: Ceci est un commentaire :)`

3.2 Clauses de XQuery

Afin d'expliquer les clauses d'une requête XQuery, nous utilisons l'exemple de document XML de services en associant cette fois-ci à chaque service une catégorie et en rajoutant un service supplémentaire de l'entreprise:

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
      <service category="design">
            <name>Web Design</name>
            <description>Custom website design </description>
            <price currency="DA">5000.00</price>
            <duration unit="day">14</duration>
            <year> 2024 </year>
      </service>
      <service category="content">
            <name>Content Writing</name>
            <description>Content creation for websites </description>
            <price currency="DA">20000.00</price>
            <duration unit="day">7</duration>
            <year> 2023 </year>
      </service>
      <service category="analysis">
            <name>SEO Consultation</name>
            <description>Search engine optimization analysis</description>
            <price>70000.00</price>
```

```
            <duration unit="hour">3</duration>
            <year> 2022 </year>
      </service>
      <service category="design">
            <name>Mobile App Design</name>
            <description>Design of Mobile App interface</description>
            <price currency="DA">3000.00</price>
            <duration unit="day">2</duration>
            <year> 2024 </year>
      </service>
</services>
```

3.2.1 Les clauses for et where

Une requête simple inclut les clauses for, where et return. Par exemple, pour afficher la description des services proposés par l'entreprise et dont le prix est inférieur à 20000, on utilise la requête suivante:

```
for $x in doc("services.xml")/services/service
where $x/price<20000
return $x/description
```

Dans cet exemple, on a déclaré une variable `$x` a qui on a affecté le résultat de la sélection des services de notre document (arbre) XML à l'aide de `in`. Nous avons précisé le document XML à l'aide de la fonction `doc`. Le filtrage supplémentaire par rapport au prix des services a été effectué à l'aide de la clause where. Le résultat de filtrage donne en principe les sous-arbres services qui respectent la condition, mais nous avons choisi d'afficher uniquement la description des services.

L'exécution de cette requête donne le résultat suivant:

```
<description>Custom website design </description>
<description>Design of Mobile App interface</description>
```

De la même façon, nous pouvons choisir d'afficher le nom et le prix exacte de ses services comme suit:

```
for $x in doc("services.xml")/services/service
where $x/price<20000
return $x/(name, price)
```

L'exécution de cette requête donne le résultat suivant:

```
<name>Web Design</name>
<price currency="DA">5000.00</price>
<name>Mobile App Design</name>
<price currency="DA">3000.00</price>
```

Il est également possible de combiner plusieurs conditions de filtrage en utilisant les opérateurs and et or. Par exemple:

```
for $x in doc("services.xml")/services/service
where $x/@category= 'design' and number($x/price) < 4000
```

```
return $x
```

Cette requête sélectionne tous les services de la catégorie “design” dont le prix est inférieur à 4000.

Dans le second exemple, on souhaite sélectionner les services dont la description commence par "C” ou dont le prix est inférieur à 4000.

```
for $x in doc("services.xml")/services/service
where starts-with($x/description, ‘C’) or number($x/price) < 4000
return $x
```

Nous avons utilisé l’opérateur de comparaison dans les requêtes précédentes. D’autre opérateurs de comparaison sont possibles, notamment:

- A = B: vérifier l’égalité de valeur de A et B
- A != B: vérifier l’inégalité de valeurs
- Opérateurs de supériorité et infériorité: <, <=, >, >=.

Il faut noter qu’il est possible d'utiliser les entités pour exprimer ces opérateurs dans le cas d'ambiguïté ou de résultats multiple d’une requête: eq (=), ne (!=), lt (<), le (<=), gt(>), ge (>=).

Même si la clause for permet de faciliter le parcours d’une forêt ou d’un arbre pour sélectionner dans chaque itération un nœud ou un sous-arbre et l’affecter à une variable, elle peut être également utilisée pour d’autres types de variables de type primitive. Dans l’exemple suivant, la boucle for permet d’afficher de nouvelles balises créés avec des valeurs entre 1 et 5:

```
for $x in (1 to 5)
return <test> {$x} </test>
```

Le résultat de la requête est le suivant:

```
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

Il est également possible de combiner plusieurs expressions séparées par des virgules, comme dans l’exemple suivant :

```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

Cette requête affiche le résultat suivant:

```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
```

```
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

Le mot clé “at” peut être aussi utilisé pour compter le nombre d'itérations dans une requête. Par exemple:

```
for $x at $i in doc("services.xml")/services/service/name
return <service>{$i}.{data($x)}</service>
```

La requête donne ce résultat:

```
<service> 1. Web Design </service>
<service> 2. Content Writing </service>
<service> 3. SEO Consultation </service>
<service> 4. Mobile App Design </service>
```

Ici, la fonction `data` permet d’afficher uniquement la valeur atomique des balises (sans les balises name).

3.2.2 La clause order by

Cette clause permet d'ordonner les résultats d’une requête par ordre croissant. Par exemple, nous souhaitons ordonner le résultat d’une requête affichant les noms de service:

```
for $x in doc("services.xml")/services/service/name
order by $x
return $x
```

Cette requête affiche le résultat suivant:

```
<name>Content Writing</name>
<name>Mobile App Design</name>
<name>SEO Consultation</name>
<name>Web Design</name>
```

3.2.3 La clause let

Let permet d’assigner une valeur à une variable pour éviter la répétition des expressions. Dans l’exemple suivant, on déclare la variable nommé par exemple `currentYear` qu’on utilise pour faire le filtrage du résultat:

```
let $currentYear := 2024
for $x in doc("services.xml")//service
where xs:int($x/year) > $currentYear - 2
return <service>{data($x/name)}</service>
```

Cette requête affiche le résultat suivant:

```
<service>Web Design</service>
<service>Content Writing</service>
<service>Mobile App Design</service>
```

3.3 Expressions conditionnelles

Afin d'exprimer des conditions sur les résultats d'une requête, XQuery propose les instructions: `if … then … else`. Voici un exemple d'une requête qui affiche le nom des services selon leurs catégories en proposant une nouvelle structuration. Les nom de services de la catégorie `design` seront délimités par la balise `conception` et les autres services par la balise `other`.

```
for $x in doc("services.xml")/services/service
return
if ($x/@category="design")
then <conception>{data($x/name)}</conception>
else <other>{data($x/name)}</other>
```

Le résultat de la requête est le suivant:

```
<conception>Web Design</conception>
<other>Content Writing</other>
<other>SEO Consultation</other>
<conception>Mobile App Design</conception>
```

3.4 Fonctions utilisateur

Il est possible aussi de définir des fonctions utilisateur. La syntaxe générale d'une fonction est la suivante :

```
declare function prefix:fun_name($par as type) as returnType
{
 …  code ...
};
```

Le mot clé "`declare function`" est obligatoire pour indiquer qu'il s'agit d'une déclaration d'une nouvelle fonction. Ensuite, le préfixe (prefix) indique l'espace de nom utilisé ou bien par défaut "`local`". Ensuite il suffit de donner un nom à la fonction ainsi que les paramètres et leur type. XQuery utilise les mêmes types de données que XSD (`string`, `date`, `integer`, `decimal`, ….) . La fonction aussi peut avoir un type de retour si elle retourne une valeur.

La syntaxe utilisée pour appeler une fonction est la suivante:

```
<fun_name>{prefix:fun_name (…, …)}</fun_name>
```

Voici un exemple d'une fonction qui permet d'afficher un message selon la moyenne d'un étudiant:

```
declare function local: getMessage($moyenne as xs:decimal)
```

```
as xs:string {
if ($moyenne >= 10) then return concat("Admis", "*****")
else return concat("Ajourné, "****") };

(: Appel de la fonction :)
let $moy := 15
return local:getMessage($moy)
```

Il faut noter que dans une expression ou une fonction Xquery, on peut utiliser les fonctions prédéfinies dans XPath comme `concat` qui est utilisé dans cet exemple.

Dans une requête, on peut utiliser les données à partir d'une base XML. Dans l'exemple suivant nous décrivons une fonction qui calcul le prix du premier service après une promotion de 20%

```
declare function local:price($p as xs:decimal?,$d as xs:decimal?) as
xs:decimal?
{
let $temp := ($p * $d) div 100
return ($p - $temp)
};

(: exemple de l'appel de la fonction :)
<Price>{local:minPrice($/services/service[1]/price, 20)}</Price>
```

Enfin, XQuery est le langage de requêtes pour XML standardisé par le W3C. De plus de son utilisation pour formuler des requêtes sur des bases de données XML, il permet également de concevoir des pages Web et de transformer les documents XML en XHTML

4. Exercices

Pour tester les requêtes XQuery, vous pouvez utiliser BaseX[1], un SGBD de bases de données XML qui est libre et open source.

Exercice 1

Soit un document XML qui inclut un ensemble de contacts:

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts>
<contact type="sim">
      <lastname>Mustapha</lastname>
      <firstname>Isac</firstname>
      <phone>010332435</phone>
      <age>21</age>
</contact>
<contact type="phone">
         <lastname>Mohamed</lastname>
         <firstname>Otmane</firstname>
         <phone>010366635</phone>
         <age>54</age>
</contact>
<contact type="phone">
      <lastname>Amine</lastname>
      <firstname>Islam</firstname>

      <phone>011112435</phone>
      <age>14</age>
</contact>
<contact type="sim">
         <lastname>Ali</lastname>
         <firstname>Ali</firstname>
         <phone>0166632435</phone>
         <age>13</age>
     </contact>
</contacts>
```

1) Créer une nouvelle base de données BaseX (DataBase → New… → nommer votre base → Ok)

2) Ecrire fichier XML « contacts.xml » et vérifier s'il est bien formé (Utiliser l'éditeur, le bouton « New » → nommer puis vérifier l'extension de votre fichier, Ok)

[1] BaseX: https://basex.org/

3) Créer un fichier « contacts.xq » qui vous permettra de formuler les requêtes (Utiliser l'éditeur, le bouton «New» → nommer puis vérifier l'extension de votre fichier, Ok) . Une fois la requête introduite, il suffit de l'exécuter (Utiliser l'éditeur, le bouton « Run query »). Donner puis exécuter les requêtes suivantes :

1. Afficher les noms des contacts dont l'âge est moins de 15 ans.

2. Afficher les contacts dont l'âge est moins de 15 ans. Dans ce cas, il faut afficher le nom et le prénom de la personne ainsi que son numéro de téléphone.

3. Afficher les contacts sauvegardés uniquement en carte sim.

4. Afficher le nombre de contacts.

5. Donner la requête qui montre le résultat suivant :

```
<sim>Mustapha</sim>
<phone>Mohamed</phone>
<phone>Amine</phone>
<sim>Ali</sim>
```

6. Donner la requête qui affiche un nouveau document XML dont la racine est « contacts » et les sous éléments sont nommés « person ». Ce document contient uniquement les contacts âgés de plus de 11 ans et qui sont sauvegardés uniquement au téléphone.

Exercice 2

Soit une partie du document XML du centre commercial précédant modifié:

```
<?xml version="1.0" encoding="UTF-8"?>
<centre>
  <!--Liste des magasins -->
  <magasins>
    <magasin categorie="etranger" locataire="o1">
      <nom>HM </nom>
      <type> vetements adultes femme et homme</type>
      <reference> 23453344 </reference>
      <pays-origine>Espagne</pays-origine>
      <téléphone>+213 4455577</téléphone>
      <local>
        <numero>24</numero>
        <superficie>70 </superficie>
        <lot>120</lot>
      </local>
    </magasin>
    <magasin categorie="local" locataire="o2">
      <nom>ilyes bijoux </nom>
      <type> accessoires femme et homme</type>
      <reference> 23453343 </reference>
      <pays-origine>Algérie</pays-origine>
```

```
      <téléphone>+213 4455555</téléphone>
      <local>
        <numero>12 </numero>
        <superficie>70 </superficie>
        <lot>130</lot>
      </local>
      </magasin>
</centre>
```

Donnez le résultat des requêtes XQuery suivantes:

1.
```
<magasins>
  {
  for $b in doc("magasin.xml")//magasin
  where $b/superficie = "70" and  $b/@categorie = "local"
  return
     <magasin type="{$b/@categorie}">
        {$b/nom}
     </magasin>
  }
</magasins>
```

2.
```
<results>
  {
  for $b in doc("magasin.xml")//magasin,
      $t in $b/nom,
      $a in $b/type
  return
     <result>
        {$t}
        {$a}
     </result>
}
</results>
```

Exercice 3

Soit le document XML suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
      <service>
            <name>Web Design</name>
            <description>Custom website design </description>
            <price currency="DA">5000.00</price>
            <duration unit=”day”>14</duration>
            <date> 2020-02-08</date>
      </service>
      <service>
            <name>Content Writing</name>
            <description>Content creation for websites </description>
            <price currency="DA">20000.00</price>
            <duration unit=”day”>7</duration>
```

```
            <date> 2021-03-09</date>
      </service>
      <service>
            <name>SEO Consultation</name>
            <description>Search engine optimization analysis</description>
            <price>70000.00</price>
            <duration unit=”day”hour>3</duration>
            <date> 2022-04-10</date>
      </service>
</services>
```

1. Écrire la requête XQuery qui permet d’afficher les services payés par jour.
2. Écrire la requête XQuery qui permet d’afficher les services passées après le 2020-02-08 et dont le montant est compris entre 1000 et 80000

Exercice 4

Soit un document XML de l’exercice 2 du Chapitre II (vols.xml) . Modifier votre document pour inclure les tarifs de chaque vol pour la classe économique et première classe (first class) comme suit: vol 1: 7000, 15000; vol 2: 8000, 10000; vol 3: 7000, 14000 ; vol 4: 17000, 25000. Donner puis exécuter les requêtes suivantes :

1. la requête qui affiche le résultat suivant:

```
<vol>Oran Annaba</vol>
<vol>Oran Annaba</vol>
<vol>Oran Annaba</vol>
<vol>Oran Tunis</vol>
```

2. Utilisez la structure suivante pour vérifier s’il existe un vol dont la ville de destination est Mostaganem.

```
 some $var in expr1 satisfies expr2
(:il existe au moins un nœud retourné par l’expression expr1 qui satisfait l’expression expr2 :)
```

3. Utilisez la structure suivante pour vérifier si la ville de départ de chaque vol est Oran.

```
 every $var in expr1 satisfies expr2
(: tous les nœuds retournés par l’expression expr1 satisfont l’expression expr2:)
```

4. Affichez le prix moyen des vols en classe économique.

5. Écrire une fonction qui calcule le produit de deux entiers passés en paramètres. Appelez ensuite la fonction en affichant le résultat comme suit: <résultat> valeur </résultat>

Chapitre IV: Autres langages pour les données semi-structurées

D’autres langages pour les données semi-structurées ont suscité l'intérêt de développeurs, par exemple XML, JSON, CSV, YAML, TOML et INI. Ce chapitre permet d'introduire les formats les plus utilisés actuellement, notamment YAML, JSON, et CSV tout en donnant des exemples de leurs utilisations en Python.

1. YAML

YAML, ou YAML Ain’t Markup Language est un langage permettant de décrire des données semi-structurées sous forme de paires clé-valeur. Il est généralement utilisé en pratique pour décrire des fichiers de configuration. Avec une structure simple, YAML permet de décrire différents types de données, tels que les chaînes de caractères, les listes et les dictionnaires. Ci-après un exemple d’un fichier YAML qui représente les données d’une entreprise startup Facom situé à Mostaganem et possèdent deux employés:

```
company: Facom
employees:
    -name: Ahmed Laroui
    age: 30
    position: Software designer
   -name: Ines Amada
    age: 25
    position: Software engineer
location:
  city: Mostaganem
  country: Algeria
```

Pour parser des fichiers YAML en python, on peut utiliser la bibliothèque PyYAML, comme dans l’exemple suivant:

```
# bibliothèque pour parser YAML
 import yaml
# ouverture et chargement du fichier YAML dans une variable yaml_data
with open('data.yaml', 'r') as file:
    yaml_data = yaml.safe_load(file)

# Accès et affichage des données
print("Entreprise:", yaml_data['company'])
print("Localisation:", yaml_data['location']['city'],
                    ",", yaml_data['location']['country'])
print("\n Employés:")
for employee in yaml_data['employees']:
```

```
    print("Nom et prénom:", employee['name'])
    print("Age:", employee['age'])
    print("Poste:", employee['position'])
    print()
```

Dans le code de l'exemple, la méthode open('data.yaml', 'r') permet d'ouvrir le fichier en mode lecture (read) et utilise yaml.safe_load(file) pour charger le contenu du fichier dans la variable yaml_data. Cette variable est utilisée par la suite pour accéder aux différentes données, notamment le nom de l'entreprise, sa localisation et les employés. Même si la structure utilisée de YAML est simple, elle est souvent réservée en pratique pour les fichiers de configuration et non pas pour représenter des données.

2. JSON

JSON ou JavaScript Object Notation, est un format de données semi-structurées qui est largement utilisé pour l'échange de données sur le Web. Il est largement utilisé dans les API Web pour échanger des données entre serveurs et clients, ainsi que pour le stockage de configurations et de données structurées. Comme YAML, JSON utilise une structure de données basée sur des paires clé-valeur avec quelques différences dans la syntaxe. En particulier, les données sont organisées en objets qui sont encadrés par des accolades {} et des tableaux qui sont encadrés par des crochets [].

L'exemple précédent des données de l'entreprise est décrit en JSON comme suit:

```
{
  "company": "Facom",
  "employees": [
    {
      "name": "Ahmed Laroui",
      "age": 30,
      "position": "Software designer"
    },
    {
      "name": "Ines Amada",
      "age": 25,
      "position": "Software engineer"
    }
],
  "location": {
    "city": "Mostaganem",
    "country": "Algeria"
  }
}
```

Afin de charger et parser un fichier JSON en python, on utilise la bibliothèque json comme suit:

```
import json
# Ouvrir le fichier JSON et récupérer son contenu
```

```
with open('data2.json', 'r') as file:
    json_data = json.load(file)
# Accès et affichage de contenu
print("Entreprise:", json_data['company'])
print("Adresse:", json_data['location']['city'],
",", json_data['location']['country'])
print("\n Employés:")
for employee in json_data['employees']:
    print("Nom et prénom:", employee['name'])
    print("Age:", employee['age'])
    print("Poste:", employee['position'])
    print()
```

Si on souhaite convertir ce fichier JSON en XML, il suffit de récupérer son contenu puis de créer le fichier XML à l'aide de la bibliothèque xml de python. Pour cela il faut utiliser la fonction `ET.Element (name)` qui permet de créer l'élément racine d'un document XML avec le nom "`name`". Les sous-éléments sont ajoutés à l'aide de `ET.SubElement` en fonction des données extraites du JSON. Ensuite, il faut constituer l'arbre en utilisant `ET.ElementTree(root)` avec root l'élément racine précédemment créé. Enfin, il suffit d'écrire cet arbre sur un fichier texte portant l'extension .xml à l'aide de la fonction tree.write. Voici un exemple de script python permettant de transformer l'exemple de JSON précédent en un fichier XML:

```
import json
import xml.etree.ElementTree as ET

# ouvrir le fichier JSON et récupérer son contenu
with open('data.json', 'r') as file:
    json_data = json.load(file)

# Création de la racine du document XML
root = ET.Element("company_info")

# Ajouter les sous-éléments
company = ET.SubElement(root, "company")
company.text = json_data['company']

location = ET.SubElement(root, "location")
city = ET.SubElement(location, "city")
city.text = json_data['location']['city']
country = ET.SubElement(location, "country")
country.text = json_data['location']['country']

employees = ET.SubElement(root, "employees")
for employee in json_data['employees']:
    emp = ET.SubElement(employees, "employee")
    name = ET.SubElement(emp, "name")
    name.text = employee['name']
```

```
    age = ET.SubElement(emp, "age")
    age.text = str(employee['age'])
    position = ET.SubElement(emp, "position")
    position.text = employee['position']

# Création de l'arbre XML
tree = ET.ElementTree(root)

# créer un fichier XML en écriture et sauvegarder l'arbre XML
tree.write("data.xml",       xml_declaration=True,       encoding='utf-8',
method="xml")

# ouvrir et afficher le document XML précédent
with open("data.xml", "r", encoding='utf-8') as xml_file:
    print(xml_file.read())
```

Enfin, JSON a été largement adopté en raison de sa simplicité et lisibilité pour l'échange de données en Web. La plupart des langages de programmation proposent des bibliothèques pour manipuler ce type de données semi-structurées. Cependant, JSON reste un format simple pour les données qui ne supporte pas l'insertion des commentaires et les types de données complexes tels que les dates, les fonctions, etc.

3. CSV

CSV (Comma-Separated Values) est un autre format de données semi-structurées qui est et largement utilisé pour le stockage et l'échange de données tabulaires. Les données sont organisées sous forme de lignes et de colonnes. Les données dans chaque ligne sont séparées par des virgules (ou d'autres délimiteurs comme des points-virgules, des tabulations, etc.). CSV est très souvent utilisé en science de données. Les lignes représentent souvent les valeurs d'une variable selon les attributs représentés dans les colonnes. CSV est également utilisé pour l'import/export de données entre applications.

L'exemple précédent de l'entreprise peut être écrit en CSV comme suit:

```
company,employees_name,employees_age,employees_position,location_city,loca
tion_country
Facom,Ahmed Laroui,30,Software designer,Mostaganem,Algeria
Facom,Ines Amada,25,Software engineer,Mostaganem,Algeria
```

Afin de traiter les fichiers CSV, Python propose la bibliothèque csv. Pour lire le fichier, il suffit d'utiliser la fonction csv.DictReader(file). Cette fonction permet de lire chaque ligne du fichier CSV en tant que dictionnaire, où les noms de colonnes deviennent les clés.

```
import csv

csv_file = 'data.csv'
```

```
# Lecture du fichier
with open(csv_file, 'r', newline='') as file:
    csv_reader = csv.DictReader(file)

    # Initialisation des données
    csv_data = {
        'company': None,
        'employees': [],
        'location': {
            'city': None,
            'country': None
        }
    }

    # Lecture des lignes du fichier CSV
    for row in csv_reader:
        csv_data['company'] = row['company']
        csv_data['location']['city'] = row['location_city']
        csv_data['location']['country'] = row['location_country']

        employee = {
            'name': row['employees_name'],
            'age': int(row['employees_age']),  # Convertir en entier
            'position': row['employees_position']
        }
        csv_data['employees'].append(employee)

# Affichage
print("Entreprise:", csv_data['company'])
print("Localisation:", csv_data['location']['city'],
",", csv_data['location']['country'])

print("\Employés:")
for employee in csv_data['employees']:
    print("Nom et prénom:", employee['name'])
    print("Age:", employee['age'])
    print("Poste:", employee['position'])
    print()
```

Enfin, même si le CSV représente un format simple et léger pour représenter les données, il reste toutefois limité puisqu' il ne supporte pas les structures de données complexes comme les objets ou les relations.

SOLUTION DES EXERCICES

CHAPITRE I

Solution de l'exercice 1:

1. et 4. le document XML bien formé:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <!DOCTYPE classe SYSTEM "file:/C:/Users/nhoci/Desktop/xml/classe.dtd">
<classe>
      <eleve ident="E01">
            <nom> Mansar </nom>
            <prenom>Ali</prenom>
            <adresse>
                  <numero>23</numero>
                  <voie cat="rue">Aban Ramdhan</voie>
                  <ville> Oran </ville>
            </adresse>
            <resultats>
                  <ar>12</ar>
                  <phys>6</phys>
            </resultats>
      </eleve>
      <eleve ident="E02">
            <nom> Alaribi </nom>
            <prenom>Amina</prenom>
            <adresse>
                  <numero>16</numero>
                  <voie cat="allée">des artistes</voie>
                  <ville>Mostaganem</ville>
            </adresse>
            <resultats>
                  <ar>11</ar>
                  <math>9</math>
                  <phys>16</phys>
            </resultats>
      </eleve>

      <eleve ident="E03">
            <nom> Amer </nom>
            <prenom>Mohamed</prenom>
            <adresse>
                  <numero>12</numero>
                  <voie cat="avenue">Mostaganem</voie>
                  <ville>Oran</ville>
            </adresse>
            <email>amer.mohammed@umab.dz</email>
            <resultats>
```

```
                <ar>10</ar>
                <math>19</math>
                <phys>12</phys>
                </resultats>
          </eleve>
    <eleve ident="E04">
          <nom> Balim </nom>
          <prenom>Houda</prenom>
          <adresse>
                <numero>20</numero>
                <voie cat="rue">Khemisti</voie>
                <ville>Mostaganem</ville>
          </adresse>
          <resultats>
                <ar>15</ar>
                <math>10</math>
                <phys>20</phys>
          </resultats>
    </eleve>
</classe>
```

2. Une amélioration de la modélisation XML utilisée consiste à transformer l'identifiant de l'étudiant en un sous-élément pour avoir la possibilité de mieux gérer l'extension de ce document. En effet, l'identifiant peut avoir plusieurs instances et ne correspond pas alors à un attribut qui est utilisé souvent pour représenter des métadonnées. De plus, une seule balise résultats (au lieu de deux balises résultats-concours et résultats) puisque les valeurs possèdent la même sémantique et cela peut simplifier par la suite l'analyse et l'extraction des données du document XML.

3. Le type des éléments suivants est:

- classe: l'élément racine
- adresse, ville: sous-élément
- cat: attribut
- E01: valeur de l'attribut ou valeur atomique de l'attribut
- Mostaganem: valeur du sous-élément ou valeur atomique du sous-élément

Solution de l'exercice 2:

1. Le document XML (vols.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<vols>
      <vol type="national">
            <companie>
                  <nom>Air algerie</nom>
                  <pays_origine> Algérie</pays_origine>
            </companie>
            <avion>
                  <numéro_avion> 2745 </numéro_avion>
                  <description> Airbus A330-200 </description>
            </avion>
```

```
      <ville_de_départ> Oran</ville_de_départ>
      <ville_d_arrivé> Annaba</ville_d_arrivé>
      <date_de-départ>
            <date> 12-12-2024</date>
            <heure> 09:10</heure>
      </date_de-départ>
            <date_d_arrivée>
            <date> 12-12-2024</date>
      <heure> 10:10</heure>
      </date_d_arrivée>
      <aeroport_de-départ> l'aéroport international d'Oran Ahmed Ben
      Bella
      </aeroport_de-départ>
      <aeroport_d_arrivée>  Aéroport  international  d'Annaba  Rabah
      Bitat
      </aeroport_d_arrivée>
      <pilote>
            <identifiant> 0021dz </identifiant>
            <nom> Mohamed Mehdi</nom>
      </pilote>
</vol>

<vol type="national">
      <companie>
            <nom>Air algerie</nom>
            <pays_origine> Algérie</pays_origine>
      </companie>
      <avion>
            <numéro_avion> 2745 </numéro_avion>
            <description> Airbus A330-200 </description>
      </avion>
      <ville_de_départ> Oran</ville_de_départ>
      <ville_d_arrivé> Annaba</ville_d_arrivé>
      <date_de-départ>
            <date> 12-12-2024</date>
            <heure> 09:30</heure>
      </date_de-départ>
      <date_d_arrivée>
            <date> 12-12-2024</date>
            <heure> 10:30</heure>
      </date_d_arrivée>
      <aeroport_de-départ> l'aéroport international d'Oran Ahmed Ben
      Bella
      </aeroport_de-départ>
      <aeroport_d_arrivée>  Aéroport  international  d'Annaba  Rabah
      Bitat
      </aeroport_d_arrivée>
      <pilote>
            <identifiant> 0021dz </identifiant>
            <nom> Belkacem Amin</nom>
      </pilote>
</vol>

<vol type="national">
      <companie>
            <nom>Air algerie</nom>
```

```
            <pays_origine> Algérie</pays_origine>
      </companie>
      <avion>
            <numéro_avion> 2745 </numéro_avion>
            <description> Airbus A330-200 </description>
      </avion>
      <ville_de_départ> Oran</ville_de_départ>
      <ville_d_arrivé> Annaba</ville_d_arrivé>
      <date_de-départ>
            <date> 12-12-2024</date>
            <heure> 11:10</heure>
      </date_de-départ>
      <date_d_arrivée>
            <date> 12-12-2024</date>
            <heure> 12:10</heure>
      </date_d_arrivée>
      <aeroport_de-départ> l'aéroport international d'Oran Ahmed Ben
      Bella
      </aeroport_de-départ>
      <aeroport_d_arrivée> Aéroport international d'Annaba Rabah
      Bitat
      </aeroport_d_arrivée>
      <pilote>
            <identifiant> 0021dz </identifiant>
            <nom>Limam Fatima</nom>
      </pilote>
</vol>

<vol type="international">
      <companie>
            <nom> TunisAir </nom>
            <pays_origine> Tunisie </pays_origine>
      </companie>
      <avion>
            <numéro_avion> 0012 </numéro_avion>
            <description> Airbus A380, </description>
      </avion>
      <ville_de_départ> Oran</ville_de_départ>
      <ville_d_arrivé> Tunis</ville_d_arrivé>
      <date_de-départ>
            <date> 12-12-2024</date>
            <heure> 11:10</heure>
      </date_de-départ>
      <date_d_arrivée>
            <date> 12-12-2024</date>
            <heure> 13:10</heure>
      </date_d_arrivée>
      <aeroport_de-départ>l'aéroport international d'Oran Ahmed Ben
      Bella
      </aeroport_de-départ>
      <aeroport_d_arrivée> l'aéroport international Tunis Carthage
      </aeroport_d_arrivée>
      <pilote>
            <identifiant> 0031tn </identifiant>
            <nom>Mohamed Elghani</nom>
      </pilote>
```

```
      </vol>
</vols>
```

1. La DTD du document XML (vols.dtd):

```
<!ELEMENT vols (vol+)>
<!ELEMENT vol (companie, avion, ville_de_départ, ville_d_arrivé,
date_de-départ, date_d_arrivée, aeroport_de-départ,
aeroport_d_arrivée, pilote)>
<!ELEMENT companie (nom, pays_origine)>
<!ELEMENT date_de-départ (date, heure)>
<!ELEMENT avion (numéro_avion, description)>
<!ELEMENT date_d_arrivée (date, heure)>
<!ELEMENT pilote (identifiant, nom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT pays_origine (#PCDATA)>
<!ELEMENT numéro_avion (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT ville_de_départ (#PCDATA)>
<!ELEMENT ville_d_arrivé (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT heure (#PCDATA)>
<!ELEMENT aeroport_de-départ (#PCDATA)>
<!ELEMENT aeroport_d_arrivée (#PCDATA)>
<!ELEMENT identifiant (#PCDATA)>
<!ATTLIST vol type (national|international) #REQUIRED>
```

Solution de l'exercice 3

La DTD pour chaque structure donnée:

1
```
<!DOCTYPE films[

<!ELEMENT films(film*)>
<!ELEMENT film EMPTY>
<!ATTLIST film type CDATA #REQUIRED
                affiche CDATA #REQUIRED
                remarques CDATA #IMPLIED>
]>
```

2
```
<!DOCTYPE expression [
<!ELEMENT expression (exp1, exp2, exp3)>
<!ELEMENT exp1 (a+ | b+) >
<!ELEMENT exp2 (b*, (a, b)+) >
<!ELEMENT exp3 ((c|b)+, (a|d)?)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ELEMENT c (#PCDATA)>
<!ELEMENT d (#PCDATA)>
]>
```

1)

```
<!DOCTYPE Lettre [
<!ELEMENT Lettre(#PCDATA | objet| durée)*>
]>
```

Solution de l'exercice 4

1. Le document XML (chaine.xml) bien formé

```
<?xml version="1.0" encoding="UTF-8"?>
<chaine>
      <documentaire categorie='nature'>
            <duré> 120 </duré>
            <description>     l'Amazonie   et   le   risque   de   sécheresse
            </description>
            <nombre_repetition> 2 </nombre_repetition>
      </documentaire>
      <journal>
            <duré> 30  </duré>
            <description>
            <producteur>
                  <nom> Dimi </nom>
                  <prenom> Amin </prenom>
            </producteur>
            </description>
            <nombre_repetition> 1 </nombre_repetition>
            <directeur>
                  <prenom>Ali  </prenom>
                  <nom> Labouré  </nom>
            </directeur>
            <journaliste>
                  <prenom> Hala </prenom>
                  <nom>  Salamé </nom>
            </journaliste>
      </journal>
      <film>
            <duré> 120 </duré>
            <description> film d'action
                  <producteur>
                        <nom>  Eme </nom>
                        <prenom> Goerge </prenom>
                  </producteur>
                  <annee> 2016 </annee>
            </description>
            <nombre_repetition> 1 </nombre_repetition>
      </film>
</chaine>
```

2. La DTD (chaine.dtd)

```
<!ELEMENT chaine (documentaire| journal| film| divertissement|
série)+>
<!ELEMENT documentaire (duré, description, nombre_repetition)>
<!ELEMENT divertissement (duré, description, nombre_repetition)>
<!ELEMENT série (duré, description, nombre_repetition)>
<!ELEMENT journal (duré, description, nombre_repetition, directeur,
journaliste)>
<!ELEMENT producteur (nom, prenom)>
<!ELEMENT directeur (prenom, nom)>
<!ELEMENT description (#PCDATA|producteur| annee)*>
<!ELEMENT journaliste (prenom, nom)>
<!ELEMENT film (duré, description, nombre_repetition)>
<!ELEMENT duré (#PCDATA)>
<!ELEMENT nombre_repetition (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT annee (#PCDATA)>
<!ATTLIST documentaire categorie CDATA #REQUIRED>
```

Solution de l'exercice 5

1. XML pour la formation universitaire :

```
<?xml version="1.0" encoding="UTF-8"?>
<formation>
      <spécialité> Informatique</spécialité>
      <semestre niveau="Licence">
            <numero>6</numero>
            <date> 2022/01/01 </date>
            <matière type="Fondamentale">
                  <code> UEF2 </code>
                  <intitulé> DSS </intitulé>
                  <responsable> Mme Hocine </responsable>
                  <contenu>
                        <objectif> l'objectif est de se…..</objectif>
                  </contenu>
                  <organisation>
                        <cours> 1h </cours>
                        <tp> 1h </tp>
                  </organisation>
                  <prérequis> langage de programmation</prérequis>
            </matière>
            <matière type="Méthodologique">
                  <code> UEM1 </code>
                  <intitulé> Cloud </intitulé>
                  <responsable> M Ali </responsable>
                  <contenu> cette matière ….</contenu>
                  <organisation>
                        <cours> 2h </cours>
                        <td> 1h </td>
                  </organisation>
                  <prérequis> Réseau TCP IP </prérequis>
            </matière>
```

```
        </semestre>
</formation>
```

2. DTD

```
<!ELEMENT formation (spécialité, semestre+)>
<!ELEMENT semestre (numero, date, matière+)>
<!ELEMENT matière (code, intitulé, responsable, contenu,
organisation, prérequis?)>
<!ELEMENT contenu (#PCDATA|objectif)*>
<!ELEMENT organisation (cours?, td?, tp?)>
<!ELEMENT spécialité (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT intitulé (#PCDATA)>
<!ELEMENT responsable (#PCDATA)>
<!ELEMENT objectif (#PCDATA)>
<!ELEMENT td (#PCDATA)>
<!ELEMENT cours (#PCDATA)>
<!ELEMENT tp (#PCDATA)>
<!ELEMENT prérequis (#PCDATA)>
<!ATTLIST semestre niveau (Licence|Master) #REQUIRED>
<!ATTLIST matière type (Fondamentale|Méthodologique|Découverte)
#REQUIRED>
```

3. XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="annee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="semestre" maxOccurs="2">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="numero" type="xs:string"/>
              <xs:element name="date" type="xs:string"/>
              <xs:element name="matière" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="code">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:pattern value="[a-zA-Z0-9]{4}"/>
                          <xs:length value="4"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                    <xs:element name="intitulé" type="xs:string"/>
                    <xs:element name="responsable" type="xs:string"/>
```

```
                    <xs:element name="contenu" type="xs:string"/>
                    <xs:element name="organisation">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="cours" type="xs:string"
                                 minOccurs="0" maxOccurs="1"/>
                          <xs:element name="td" type="xs:string"
                              minOccurs="0"
                                 maxOccurs="1"/>
                          <xs:element name="tp" type="xs:string"
                              minOccurs="0"
                                 maxOccurs="1"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="prérequis" type="xs:string"
                      minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="type" type="xs:string"
                        use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="niveau" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="Licence"/>
                  <xs:enumeration value="Master"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Solution de l'exercice 6

1. facture.dtd

```
<!ELEMENT facture (etabliepar, date, areglerpar, article+,
service+, total)>
<!ELEMENT etabliepar (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT areglerpar (#PCDATA)>
<!ELEMENT article (description, nombre, prix_unitaire)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT prix_unitaire (#PCDATA)>
<!ELEMENT service (description, prix)>
<!ELEMENT prix (#PCDATA)>
<!ELEMENT total (#PCDATA)>
```

```
<!ATTLIST prix_unitaire monnaie (euro | da ) "da">
<!ATTLIST prix monnaie CDATA "da">
```

2. facture2.dtd

```
<!ELEMENT facture (etabliepar, date, areglerpar, article+,
service*, tva?,total)>
<!ELEMENT etabliepar (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT areglerpar (#PCDATA)>
<!ELEMENT article (description, nombre, prix_unitaire)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT prix_unitaire (#PCDATA)>
<!ELEMENT service (description, prix)>
<!ELEMENT prix (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT tva (#PCDATA)>
<!ATTLIST prix_unitaire monnaie (euro | da ) "da">
<!ATTLIST prix monnaie CDATA "da">
```

3. Le schéma XML (XSD) équivalent (facture.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
  <xs:element name="facture">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="etabliepar" type="xs:string"/>
        <xs:element name="date" type="xs:string"/>
        <xs:element name="areglerpar" type="xs:string"/>
        <xs:element name="article" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" type="xs:string"/>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="prix_unitaire">
                <xs:complexType mixed="true">
                  <xs:attribute name="monnaie" type="xs:string"
                      use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="service" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" type="xs:string"/>
              <xs:element name="prix">
                <xs:complexType mixed="true">
```

```
                    <xs:attribute name="monnaie" type="xs:string"
                         use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="tva" type="xs:string" minOccurs="0"/>
          <xs:element name="total" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

CHAPITRE II

Solution de l'exercice 1

Les expressions XPath:

1. Sélectionnez les numéros de téléphones disponibles.

//téléphone

//téléphone/text()

2. Sélectionnez tous les locataires du centre (avec leurs sous arbres entiers).

//locataire

/centre/locataires/locataire

3. Sélectionnez les noms des magasins du centre (uniquement la valeur atomique).

/centre/magasins/magasin/nom/text()

//magasin/nom/text()

4. Sélectionnez les pays d'origine des locataires.

//locataire/adresse/pays/text()

5. Afficher le nombre d'employés.

count(//employé)

6. Afficher le nombre de personnes.

count(//employé | //locataire)

7. Sélectionnez l'identifiant du locataire du magasin Nike.

//magasin[nom="Nike"]/@locataire

8. Sélectionner les informations sur ce locataire.

//locataire[@idlocataire="o3"]

9. Sélectionner les références des magasins étrangers du centre commercial.

//magasin[@categorie="etranger"]/reference/text()

10. Sélectionnez les prix des appartements avec un niveau de confort B.

//appartement[@comfort="B"]/prix/text()

11. Sélectionnez les appartements qui ne sont pas encore loués.

//appartement[not(@locataire)]

12. Sélectionnez les biens (appartements et les matériels) qui ne sont pas encore loués.

//appartement[not(@locataire)] | //matériel[not(@locataire)]

13. Sélectionnez les biens (appartements et les matériels) qui ne sont pas loués par "o1".

//appartement[@locataire!="o1"] | //matériel[not(@locataire="o1")]

14. Sélectionnez le nom et la deuxième caractéristique du premier matériel de la liste

(uniquement la valeur atomique)

//matériels/matériel[1]/nom/text()|
//matériels/matériel[1]/caractéristiques/caractéristique[2]/text()

15. Sélectionnez les locataires qui ont loué un appartement.

//locataire[@idlocataire = //appartement/@locataire]

16. Donnez la somme des prix des appartements

sum(//appartement/prix/text())

Solution de l'exercice 2

Le programme XSLT:

```
<?xml version="1.0" ?>
<xsl:stylesheet version='1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
 <xsl:template match="/">
   <html>
 <head> </head>
   <body>
     <h1> Liste des elements  <xsl:value-of select="count(//aa)"/>
           de votre document :
   </h1>
   <xsl:for-each select="//aa">
      <p> <b> Elements b : </b> <xsl:apply-templates select="b"/>
          <b> Elements d : </b> <xsl:apply-templates select="c/d"/>
      </p>
     </xsl:for-each>
   </body>
  </html>
  </xsl:template>
</xsl:stylesheet>
```

Solution de l'exercice 3

Le programme XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/Departures">
   <html>
      <head>
         <title>Departures</title>
           </head>
      <body>
           <h2>DEPARTURES</h2>
         <table>
               <tr >
                     <th>Time</th>
                     <th>DeparturePoint</th>
```

```
                    <th>Code</th>
                    <th>Destination</th>
                    <th>Gate</th>
                    <th>Status</th>
                </tr>
                <xsl:for-each select="/Departures/flight">
                <tr bgcolor="violet">
                    <td><xsl:value-of select="Time"/></td>
                    <td><xsl:value-of select="DeparturePoint"/></td>
                    <td><xsl:value-of select="Code"/></td>
                    <td><xsl:value-of select="Destination"/></td>
                    <td><xsl:value-of select="Gate"/></td>
                    <xsl:choose>
                        <xsl:when test="Status='CANCELLED'">
                            <td bgcolor="red">
                            <xsl:value-of select="Status"/></td>
                        </xsl:when>
                        <xsl:otherwise>
                            <td><xsl:value-of select="Status"/></td>
                        </xsl:otherwise>
                    </xsl:choose>
                  </tr>
                  </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Solution de l'exercice 4

SVG pour les formes géométriques:

```
<?xml version="1.0" standalone="no"?><!-- instructions de traitement XML
-->
<svg width="100%" height="150%"  xmlns="http://www.w3.org/2000/svg">
    <!-- Définition des formes -->
    <defs>
    <rect id="carre" width="100" height="100"
          fill="#7e36346" stroke="#dc9595" />
  <circle id="cercle" cx="50" cy="50"
          r="50" fill="#7e4536" stroke="#dc9595"/>
    <polygon id="triangle" points="50,0 100,100 0,100"
          fill="#7e0036" stroke="#dc9595" />
    <polygon id="losange" points="50,5 90,50 50,95 10,50"
          fill="#7e0936" stroke="#dc9595" />
    </defs>
    <!-- Liste de choix des formes -->
    <g id="choixFormes">
       <rect x="550" y="100" width="140" height="130"
             fill="#f0f0f0" rx="10" ry="10" />
<!--la zone qui contient les button-->
<text  x="580"  y="130"  onclick="document.getElementById('forme').innerHTML
= document.getElementById('carre').outerHTML;" >Carre</text>
```

```
<text x="580" y="160" onclick="document.getElementById('forme').innerHTML
= document.getElementById('cercle').outerHTML;" >Cercle</text>

<text x="580" y="190" onclick="document.getElementById('forme').innerHTML
= document.getElementById('triangle').outerHTML;">Triangle</text>

<text x="580" y="220" onclick="document.getElementById('forme').innerHTML
= document.getElementById('losange').outerHTML;">Losange</text>

<!-- onclick pour mettre à jour le contenu de l'élément <g id="forme">
avec le code SVG de la forme sélectionnée.-->
    </g>
    <!-- Affichage des formes-->
   <g id="forme" transform="translate(570,250)"></g>
         <line   x1="150"   y1="450"   x2="95%"   y2="450"   stroke="black"
stroke-width="2" stroke-dasharray="5,5" />
<!-- Styles CSS -->
    <style type="text/css">
   …
    </style>
</svg>
```

Solution de l’exercice 5

SVG en appliquant les filtres:

```
<defs>
<filter id="img1">
<feColorMatrix in="SourceGraphic" type="saturate" values="0">
</feColorMatrix>
  </filter>
  <filter id="img2">
<feColorMatrix in="SourceGraphic" type="saturate" values="2">
</feColorMatrix>
  </filter>
</defs>

<image  filter="url(#img1)"  href="https://emeple.com/img.svg"  width="40%"
height="60%" x="50" y="250"/>

<image  filter="url(#img2)"  href="https://emeple.com/img.svg"  width="40%"
height="60%" x="620" y="250"/>

<line    x1="100"    y1="1000"    x2="90%"    y2="1000"    stroke="black"
stroke-width="2" stroke-dasharray="5,5" />
```

Solution de l’exercice 6

Le programme permettant de lire et parser un document XML à l’aide de l’API Stax:

```
public class Contact {
    private String lastname;
    private String firstname;
    private String phone;
     public Contact(String lastname, String firstname, String phone, int
```

```
age) {
        this.lastname = lastname;
        this.firstname = firstname;
        this.phone = phone;
    }
    public String getLastname() {
        return lastname;
    }
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
 }
import java.io.*;
import java.util.*;
import javax.xml.stream.*;
import javax.xml.stream.events.*;

public class Parser {

      static final String CONTACT= "contact";
      static final String LASTNAME= "lastname";
      static final String FIRSTNAME= "firstname";
      static final String PHONE= "phone";

        public List<Contact> readContacts(String contactsFile) {
      List<Contact> contacts= new ArrayList<Contact>();

      try {
        // 1) créer une nouvelle fabrique
        XMLInputFactory inputFactory = XMLInputFactory.newInstance();

        // 2) Définir un nouveau eventReader
        InputStream in = new FileInputStream(contactsFile);
XMLEventReader eventReader =
inputFactory.createXMLEventReader(in);

        //3) lire le document XML
      Contact contact= null;

      while (eventReader.hasNext()) {

                  XMLEvent event = eventReader.nextEvent();
if (event.isStartElement()) {
                  StartElement startElement = event.asStartElement();
```

```
                String elementName=
startElement.getName().getLocalPart();

            switch (elementName) {
               // traiter dans ce qui suit chaque élément possible

                 case CONTACT:
                       contact= new Contact();
                       event = eventReader.nextEvent();
                      break;

                      case LASTNAME:
                      event = eventReader.nextEvent();

                      contact.setLastname(event.asCharacters().getData()
                      );
                      break;

                case FIRSTNAME:
                      event = eventReader.nextEvent();
                      contact.setFirstname(event.
                      asCharacters().getData());
                      break;

                case PHONE:
                      event = eventReader.nextEvent();
                      contact.setPhone(event.asCharacters().getData());
                      break;
              } // fin switch
             }
            // ajout du service à la liste
            if (event.isEndElement()) {
                EndElement endElement = event.asEndElement();
            if (endElement.getName().getLocalPart().equals(CONTACT)) {
                contacts.add(contact);
            }
          }
        }
       } catch (FileNotFoundException | XMLStreamException e) {
                e.printStackTrace();
      }
        return contacts;
      }
}

import java.util.List;
public class Main {
   public static void main(String args[]) {
       Parser read = new Parser ();
       List<Contact> contacts= read.readContacts("contacts.xml");
       for (Contact contact: contacts) {
           System.out.println(contact);
       }
   }
}
```

CHAPITRE III

Solution de l'exercice 1

Les requêtes XQuery:

1. Afficher les noms des contacts dont l'âge est moins de 15 ans.

```
for $x in doc("librairie.xml")/contacts/contact
where $x/age>15
return $x/lastname
```

Résultat

```
<lastname>Mustapha</lastname>
<lastname>Mohamed</lastname>
```

2. Afficher les contacts dont l'âge est moins de 15 ans. Dans ce cas, il faut afficher le nom et le prénom de la personne ainsi que son numéro de téléphone.

```
for $x in doc("contacts.xml")/contacts/contact
where $x/age>15
return $x/(lastname,firstname, phone)
```

Résultat :

```
<lastname>Mustapha</lastname>
<firstname>Isac</firstname>
<phone>010332435</phone>
<lastname>Mohamed</lastname>
<firstname>Otmane</firstname>
<phone>010366635</phone>
```

3. Afficher les contacts sauvegardés uniquement en carte sim.

```
for $x in doc("contacts .xml")/contacts/contact
where $x[@type='sim']
return $x
```

Résultat :

```
<contact type="sim">
 <lastname>Mustapha</lastname>
 <firstname>Isac</firstname>
 <phone>010332435</phone>
 <age>21</age>
</contact>
```

```
<contact type="sim">
 <lastname>Ali</lastname>
 <firstname>Ali</firstname>
 <phone>0166632435</phone>
 <age>13</age>
</contact>
```

4. Afficher le nombre de contacts.

```
for $x in doc("contacts.xml")/contacts
return count($x/contact)
```

Résultat: 4

5. La requête qui montre le résultat suivant :

```
<sim>Mustapha</sim>
<phone>Mohamed</phone>
<phone>Amine</phone>
<sim>Ali</sim>
```

```
for $x in doc("contacts.xml")/contacts/contact return
if ($x/@type="sim")
then <sim>{data($x/lastname)}</sim>
else <phone>{data($x/lastname)}</phone>
```

6. La requête qui affiche un nouveau document XML dont la racine est « contacts » et les sous éléments sont nommés « person ». Ce document contient uniquement les contacts âgés de plus de 11 ans et qui sont sauvegardés uniquement au téléphone.

```
<contacts> {
for $x in doc("contacts.xml")//contact
where $x/age>11 and $x/@type = "phone" return
<person> {$x/firstname}
</person> }
</contacts>
```

Résultat:

```
<contacts>
 <person>
 <firstname>Otmane</firstname>
 </person>
 <person>
 <firstname>Islam</firstname>
 </person>
</contacts>
```

Solution de l'exercice 2

Le résultat des requêtes XQuery:

1. `<magasins>`

```
     <magasin type="local">
     ilyes bijoux
   </magasin>
</magasins>
```

2.
```
<results>

     <result>

          <nom>HM </nom>
      <type> vetements adultes femme et homme</type>
     </result>
  <nom>ilyes bijoux </nom>
      <type> accessoires femme et homme</type>
  <result>
  </result>

</results>
```

Solution de l'exercice 3

1. Écrire la requête XQuery qui permet d'afficher les services payés par jour.

```
for $x in doc("services.xml")/services/service
where $x/duration[@unit="day"]
return $x
```

2. Écrire la requête XQuery qui permet d'afficher les services passées après le 2020-02-08 et dont le montant est compris entre 1000 et 80000

```
for $x in doc("services.xml")/services/service
where $x/data > xs:date('2020-02-08')
and $x/price ge 1000
and $x/price le 80000
return $x
```

Solution de l'exercice 4

1- la requête qui affiche le résultat suivant:

```
<vol>Oran Annaba</vol>
<vol>Oran Annaba</vol>
<vol>Oran Annaba</vol>
<vol>Oran Tunis</vol>
```

for $x in doc("vols_companies.xml")/vols/vol return

<vol> {$x/(ville_de_départ, ville_d_arrivé)/text()}</vol>

2. Utilisation de la structure some pour vérifier s'il existe un vol dont la ville de destination est Mostaganem.

```
 some $var in expr1 satisfies expr2
(:il existe au moins un nœud retourné par l'expression expr1 qui satisfait l'expression expr2 :)
some $x in doc("vols_companies.xml")/vols/vol satisfies $x/ville_d_arrivé="Mostaganem"
```

3. Utilisation de la structure every pour vérifier si la ville de départ de chaque vol est Oran.

```
 every $var in expr1 satisfies expr2
(: tous les nœuds retournés par l'expression expr1 satisfont l'expression expr2:)
every $x in doc("vols_companies.xml")/vols/vol satisfies $x/ville_de_départ="Oran"
```

4. Affichage du prix moyen des vols en classe économique.

```
for $p in doc("vols_companies.xml")/vols
return avg($p/vol/tarif-économique)
```

5. La fonction qui calcule le produit de deux entiers passés en paramètres:

```
declare function local:multiple($p as xs:integer?,$d as xs:integer?) as xs:integer?
{
let $disc := $p * $d
return $disc
};
 let $a:= xs:integer(20)
 let $b:=xs:integer(2)
return
<resultat>{
  local:multiple($a,$b)
}
  </resultat>
```

CONCLUSION

En conclusion, le langage XML reste un langage de structuration de données qui a historiquement suscité l'intérêt des développeurs et des chercheurs en raison de sa flexibilité, de sa simplicité, de son indépendance vis-à-vis des langages de programmation et de son extensibilité. Avec l'émergence de formats de données semi-structurés tels que JSON, parfois préférés pour des scénarios de données plus légers ou des API web, XML continue de jouer un rôle clé dans de nombreux systèmes et industries. Au cours de la dernière décennie, l'utilisation de XML s'est concentrée sur certaines applications, telles que les services Web SOAP, la configuration et le paramétrage des logiciels et frameworks de programmation, la gestion de contenu (CMS), l'affichage de flux d'actualités et de blogs comme RSS et Atom, ainsi que les applications éducatives et scientifiques contenant des données complexes.

BIBLIOGRAPHIE

- Jeff Friesen. Java XML and JSON : Document Processing for Java SE, Second Edition, ISBN 978-1484243299, Apress, 2019
- Kevin Williams, Michael Brundage, Patrick Dengler, Jeff Gabriel . XML et les bases de données; 2001, Paris : Eyrolles, ISBN 978-2-212-09282-0 (disponible à la bibliothèque d'UMAB)
- Gilles Chagnon, ; Florent Nolot,, 2007, XML : informatique ; synthèse de cours & exercices corrigés, Pearson Education, ISBN 978-2-7440-7236-9 (disponible à la bibliothèque d'UMAB)
- Richard Gaillard, Christian Soutou, Didier Lenquette, Laurent Navarro, Jean-Jacques Pagola. Programmer avec Oracle: SQL, PL-SQL, XML, JSON, PHP, Java 2020, Éditions Eyrolles, ISBN 9782212676297
- Jacques Le Maitre, Emmanuel Bruno, XQuery pour interroger des données XML : Éléments du langage et mise en oeuvre, Cours exercices corrigés, 2013, ELLIPSES, ISBN 2729883487