REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE D'ORAN
MOHAMED BOUDIAF - USTO-MB –
FACULTE DES MATHEMATIQUES ET INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Polycopié

Notes de cours et exercices corrigés

Module : Données semi-structurées

Niveau : 3ème année Licence Informatique SI et ISIL

Réalisé par : Dr. Mohamed Amine NEMMICH

Année universitaire

2021 -2022

Avant-propos

Ce polycopié est un support pédagogique ; destiné principalement aux étudiants de troisième année Licence Informatique, spécialités : "Systèmes informatiques (SI)" et "Ingénierie des Systèmes d'Information et du Logiciel (ISIL)". Ce document comporte des notes de cours du module "Données semi-structurées" avec exercices corrigés (travaux pratiques).

L'objectif vise particulièrement à étudier et maîtriser les modèles, les opérations et les techniques liés à l'exploitation des données semi-structurées comme une alternative aux modèles de données fortement structurées, à savoir : la définition de schémas et la vérification des documents XML, les langages de requêtes, le contrôle d'accès aux documents XML,… ainsi que les problèmes de recherche encore ouverts dans le domaine.

Ce polycopié est organisé en six chapitres :

- Le premier chapitre : Contexte et problématique.
- Le deuxième chapitre : Documents et hyper documents multimédias.
- Le troisième chapitre : Noyau XML.
- Le quatrième chapitre : Galaxie XML.
- Le cinquième chapitre : Bases de données XML et bases de données semi-structurées.
- Le dernier chapitre : XQUERY et les bases de données.

Table des matières

Liste des figures

1 Contexte et problématique

1.1 Rappels sur les bases de données

Les **données** sont des faits ou des chiffres bruts ou une entité. Lorsque des activités ont lieu dans l'organisation, l'effet de ces activités doit être enregistré, ce que l'on appelle des données. Les données traitées sont appelées informations. Le but du traitement des données est de générer les informations nécessaires à l'exercice des activités commerciales. Afin d'être transformées en informations significatives qui permettent et soutiennent la prise de décision, les données doivent être stockées, maintenues, traitées et analysées.

Définition de la base de données

La **base de données** (BDD) est une collection de données liées avec une structure logiquement cohérente et une certaine signification inhérente avec un objectif spécifique, un groupe d'utilisateurs et d'applications visés avec une taille très variable et une portée ou un contenu plus ou moins large.

Le **système de gestion de base de données (SGBD)** est un système logiciel à usage général qui facilite le processus de définition, de construction et de manipulation de BDD pour diverses applications.

L'objectif principal du SGBD est de fournir aux utilisateurs une **vue abstraite des données**, c'est-à-dire que le système cache certains détails sur la façon dont les données sont stockées et maintenues. Afin de simplifier l'interaction de l'utilisateur avec le système, la complexité dissimulée aux utilisateurs par le biais de 3 niveaux d'abstraction :

- **Niveau physique :** (le système de gestion de fichier) C'est le plus bas niveau d'abstraction qui décrit comment les données sont réellement stockées.
- **Niveau logique :** (SGBD interne) Ce niveau décrit et cache les données stockées dans la base de données et les relations entre ces données.
- **Niveau de vue :** (SGBD externe) La vue fournit un mécanisme de sécurité pour empêcher l'utilisateur d'accéder à certaines parties de la BDD. Les programmes d'application cachent les détails des types de données.

Instances et schémas

Instance est la collection d'informations stockées dans la BDD à un moment donnée. Également appelé état (ou occurrence). L'état de la BDD change chaque fois que la BDD est mise à jour. Le s**chéma,** également appelé intension, c'est la conception globale de la BDD. Il comprend des descriptions de la structure de la BDD et des contraintes qui doivent s'appliquer. Le SGBD définit trois schémas à trois niveaux :

- **Le schéma interne** au niveau interne pour décrire les structures de stockage des données et les chemins d'accès. Utilise généralement un modèle de données physique.
- **Le schéma conceptuel** au niveau conceptuel pour décrire la structure et les contraintes de l'ensemble de la BDD. Utilise un modèle de données conceptuel ou d'implémentation.
- **Le schéma externe** au niveau externe pour décrire les différentes vues des utilisateurs. Utilise généralement le même modèle de données que le niveau conceptuel.

Des mappages entre les niveaux de schéma sont également nécessaires. Les programmes font référence à un schéma externe et sont mappés par le SGBD sur le schéma interne pour exécution.

Modèles de données

Le **modèle de données** est à la base de la structure d'une base de données. Il s'agit d'un ensemble d'outils conceptuels permettant de décrire les données, leurs relations, leur sémantique et les contraintes de cohérence. Il existe différentes catégories de modèles de données :

- **Modèle relationnel** : il utilise une collection de tables pour représenter à la fois les données et les relations entre ces données. Chaque table a plusieurs colonnes et chaque colonne à un nom unique. Les tables sont également appelées relations.
- **Modèle Entité-Association** : (appelé parfois entité-relation) principalement pour la conception de BDD, il utilise une collection d'objets de base, appelés entités, et des relations (ou

associations) entre ces objets. Une entité est une « chose » ou un « objet » dans le monde réel qui se distingue d'un autre objet. Les entités sont représentées par leurs propriétés, appelées attributs. Tous les attributs ont des valeurs. Par exemple, une entité étudiant peut avoir le nom, la classe, l'âge comme attributs. Une Association est une liaison entre plusieurs entités, avec une signification précise. Exemple : un employé travaille dans un département. Elle peut être porteuse d'informations comme les cardinalités, les attributs et les contraintes.

- **Modèles de données basés sur des objets** : programmation orientée objet, en particulier en Java, C++ ou C#.
- **Modèle de données semi-structuré (XML) :** il permet la spécification de données où des éléments de données individuels du même type peuvent avoir différents ensembles d'attributs.
- **Autres modèles plus anciens :** comme le modèle de réseau et le modèle hiérarchique, …

Les Langages De Base De Données

Une base de données fournit un langage de définition de données (DDL) pour spécifier le schéma de la BDD et un langage de manipulation de données (DML) pour exprimer les requêtes et les mises à jour de la BDD.

Langage de manipulation des données (DML) : C'est un langage qui permet aux utilisateurs d'accéder ou de manipuler des données telles qu'elles sont organisées par le modèle de données approprié. Les types d'accès sont : la récupération d'informations stockées dans la base de données, l'insertion de nouvelles informations, et la suppression d'informations. Il en existe essentiellement deux types : les DML **procéduraux** nécessitent que l'utilisateur spécifie les données nécessaires et la manière d'obtenir ces données et les DML **déclaratifs** demandent à l'utilisateur de spécifier les données nécessaires sans préciser comment les obtenir.

Langage de définition des données (DDL) : il permet de spécifier un schéma de base de données par un ensemble de définitions. Le DDL est également utilisé pour spécifier des propriétés supplémentaires des données. SQL est un exemple qui fournit un DDL riche qui permet de définir des tables, des contraintes d'intégrité, des assertions, etc...

Par exemple pour trouver tous les instructeurs du département Comp. Sci. :

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

En outre, l'instruction DDL met à jour le dictionnaire de données, qui contient des métadonnées ; le schéma d'une table est un exemple de métadonnées.

Une **nouvelle génération de BDD** utilise des modèles de données plus riches permettant des structures de données plus complexes et définies par l'utilisateur : objets complexes, données semi-structurées (hyper documents), données distribués, gestion des données multimédia (Images, vidéos, sons, ...), aspects orientés objet, et gestion automatique de la persistance au niveau du SGBD.

NoSQL (Not only SQL) a été utilisé pour la première fois en 1998 et officialisé le 11 juin 2009 lors d'un NoSQL meetup à San Francisco.

Trop de données distribuées / trop de contraintes. Suggestion : Cependant, il y avait trop de contraintes et de données distribuées pour que cela soit entièrement réalisable. D'où les types de bases de données NoSQL : Colonne, Graphique, Document et Clé – valeur.

Donc, l'évolution des bases de données est loin d'être terminée et SQL survit. D'autres technologies continuent d'être créées et coexistent.

1.2 Multimédia et document

Le **multimédia** est devenu un élément incontournable de toute présentation. Il a trouvé une variété d'applications allant du divertissement à l'éducation. L'évolution d'Internet a également accru la demande de contenu multimédia.

Le **multimédia** est une collection d'unités multiples d'informations qui sont contraintes par des relations de synchronisation temporelle. Un **document** est constitué d'un ensemble d'informations structurelles qui peuvent se présenter sous différentes formes de médias, et lors de la présentation, elles peuvent être générées ou enregistrées. Un document est un ensemble d'informations identifiées comme une unité et destinées à la perception humaine. Un **document multimédia** est une collection

d'informations, qui peuvent être identifiées et traitées comme une unité, et sont exprimées dans deux ou plusieurs types de médias tels que le texte, l'image, l'audio et la vidéo. Un document multimédia est étroitement lié à son environnement d'outils, d'abstractions de données, de concepts de base et d'architecture de document.

Le multimédia peut être divisé en deux grandes catégories : **linéaire** et **non linéaire**. Le contenu actif linéaire progresse sans aucun contrôle de navigation, comme dans le cas d'une présentation cinématographique. Le contenu non linéaire offre à l'utilisateur une interactivité qui lui permet de contrôler sa progression, comme dans un jeu vidéo ou une formation en ligne à son propre rythme. Le contenu non linéaire est également connu sous le nom de contenu hypermédia.

Les présentations multimédias peuvent être en **direct** ou **enregistrées**. Une présentation enregistrée peut permettre l'interactivité via un système de navigation. Une présentation multimédia en direct peut permettre une interactivité par le biais d'une interaction avec le présentateur ou l'interprète.

Les **applications** et les plateformes de diffusion **du multimédia** sont pratiquement illimitées. Le multimédia trouve son application dans divers domaines, notamment le commerce électronique (publicités, achats à domicile), l'éducation (didacticiels hypermédias), l'art et le divertissement (vidéo à la demande, télévision interactive, jeux, systèmes de production et de montage vidéo numériques), l'ingénierie, la médecine, les systèmes de bases de données multimédias, les mathématiques, les affaires, la recherche scientifique, les applications spatio-temporelles (World Wide Web, vidéoconférence, groupware, réalité virtuelle).

1.3 Hypermédia, Internet et Web

L'hypertexte est un texte qui contient des liens vers d'autres textes et qui est donc généralement non linéaire. C'est le terme général appliqué au texte "cliquable". Lorsqu'un utilisateur clique sur un ou plusieurs mots, il est alors dirigé vers un autre document ou une autre zone du document actuel. Les **documents hypertextes** peuvent être statiques (préparés et stockés à l'avance) ou dynamiques (modifiant continuellement en réponse à l'entrée de l'utilisateur).

L'hypermédia est utilisé comme une extension logique du terme hypertexte, dans lequel les graphiques, l'audio, la vidéo, le texte brut et les hyperliens s'entremêlent pour créer un support d'information généralement non linéaire. Les **documents hypermédia** peuvent être définis comme un amalgame de documents hypertexte et de documents multimédia.

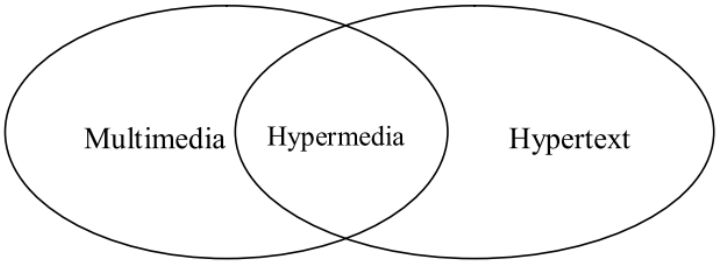


Figure 1 La relation hypertexte, hypermédia et multimedia

Le **World Wide Web** (WWW) est un exemple classique d'hypermédia, tandis qu'une présentation cinématographique non interactive est un exemple de multimédia standard en raison de l'absence d'hyperliens. La plupart des hypermédias modernes sont diffusés via des pages électroniques à partir de divers systèmes, notamment des lecteurs multimédias, des navigateurs Web et des applications autonomes. Le World Wide Web est l'application hypermédia la plus vaste et la plus couramment utilisée. Sa popularité est due à : la quantité d'informations disponibles sur les serveurs Web, la capacité de publier ces informations et la facilité de navigation dans ces informations avec un navigateur Web.

Beaucoup de gens confondent encore Internet et le World Wide Web. **Internet** peut être décrit comme un réseau de réseaux composé d'ordinateurs et de câbles. Internet transmet de petits paquets d'informations qui contiennent des adresses. Il existe de nombreux programmes différents qui utilisent Internet, tels que le courrier électronique et l'hypertexte. Le WWW, ou simplement **Web**, est un espace abstrait d'informations. Le Web est constitué de programmes et de documents qui communiquent entre ordinateurs sur Internet. Une série de liens hypertextes est utilisée pour connecter des documents sur le Web. Ainsi, le Web n'est qu'une partie d'Internet.

1.4 Problématique de ce cours

Ces dernières années, de nouveaux logiciels et techniques d'analyse de données se développent, vous permettant de recueillir des informations commerciales majeures à partir de données qualitatives ou non structurées et structurées d'e-mails, de sites Web, d'interactions avec le service client, ainsi que de données quantitatives ou structurées de statistiques et de feuilles de calcul.

Avec des données qualitatives, des moyens s'offrent à vous pour aller au-delà de ce qui s'est passé et des raisons pour lesquelles cela s'est produit avec différentes techniques, notamment l'exploration d'opinions et l'analyse de sujets.

Les données peuvent être définies comme les informations converties sous une forme très économique pour être traduites ou traitées. Les données, y compris les vidéos, les images, les sons et les textes, sont représentées par des valeurs binaires qui signifient 0 ou 1. À partir de ces deux chiffres, des modèles sont générés pour stocker différents types de données.

Les données peuvent être définies comme des informations converties sous forme numérique binaire dans les ordinateurs et les supports de transmission actuels. Avec l'augmentation du nombre d'utilisateurs d'ordinateurs, la quantité de données générées a également augmenté de manière significative au cours de la dernière décennie. Un nouveau terme a donc été créé pour désigner cet énorme volume de données généré à une vitesse fulgurante. On l'appelle big data. Ce n'est pas seulement le volume des données qui a augmenté au fil du temps.

Parallèlement à ce volume, la variété des données générées augmente rapidement. Il devient donc très important de classer les types de données qui sont générées. À l'ère de l'Internet, une grande quantité de données est générée. Ces données peuvent être du texte, des images, des vidéos, des documents, des fichiers pdf, des vidéos, des fichiers journaux, et bien d'autres encore.

Cette grande quantité de données peut être classée dans les catégories suivantes :

1. Les données structurées

Ce sont des informations qui ont été spécifiquement conçues pour être facilement consultables, et elles sont quantitatives et hautement organisées. Elles sont généralement stockées dans des bases de données relationnelles (SGBDR) et sont souvent écrites en langage d'interrogation structuré (SQL), un langage standard développé par IBM dans les années 1970 pour communiquer avec les bases de données.

Les humains ou les machines peuvent saisir des données structurées, mais celles-ci doivent respecter un cadre rigide avec des propriétés organisationnelles prédéfinies. Par exemple une base de données d'hôtel dans laquelle on peut effectuer des recherches par nom de client, numéro de téléphone, numéro de chambre et autres critères. Ou des fichiers Excel dont les données sont soigneusement organisées en lignes et en colonnes.

2. Les données non structurées

Ces données sont généralement des textes ouverts, des images, des vidéos et d'autres médias sans organisation ou conception prédéterminée. Par exemple, les documents et les autres sources de données qualitatives, les opinions et les sentiments. Ces données sont plus difficiles à analyser, mais elles peuvent être structurées pour en extraire des informations à l'aide de techniques d'apprentissage automatique, même si elles doivent d'abord être structurées pour que les machines puissent les analyser.

3. Les données semi-structurées

Les données semi-structurées sont un type de données structurées qui ne respectent pas la structure tabulaire des modèles de données associés aux bases de données relationnelles ou à d'autres types de tableaux de données, mais qui comprennent tout de même des balises ou d'autres marqueurs pour séparer les éléments sémantiques et renforcer les hiérarchies des enregistrements et des champs au sein des données. Par conséquent, on parle également de structure autodescriptive.

Dans les données semi-structurées, les entités d'une même classe peuvent avoir des caractéristiques différentes bien qu'elles soient regroupées les unes à côté des autres, et l'ordre des attributs est sans importance.

Depuis l'essor de l'internet, les données semi-structurées sont devenues plus courantes, car les documents en texte intégral et les bases de données ne sont plus les seuls types de données.

Diverses applications nécessitent un support pour l'échange d'informations, et les données semi-structurées sont courantes dans les bases de données orientées objet.

Par exemple, les courriers électroniques sont semi-structurés par expéditeur, destinataire, sujet, date, etc., ou sont automatiquement classés dans des dossiers tels que Boîte de réception, Spam, Promotions, etc. à l'aide de l'apprentissage automatique.

Les données semi-structurées sont un hybride d'images et de vidéos. Par exemple, elles peuvent comporter des balises Meta relatives au lieu, à la date ou à la personne qui les a prises, mais les informations qu'elles contiennent ne sont pas structurées. Prenons l'exemple des plateformes de médias sociaux telles que Facebook, qui organise les informations par utilisateurs, amis, groupes, marché, etc., mais les commentaires et le texte contenus dans ces classifications ne sont pas organisés.

Les données semi-structurées sont plus faciles à analyser que les données structurées car elles présentent un niveau d'organisation légèrement supérieur. Elles doivent toutefois être décomposées à l'aide d'outils d'apprentissage automatique avant de pouvoir être analysées sans intervention humaine. Elles comprennent également les données quantitatives qui, comme les données entièrement non structurées, peuvent également fournir des informations beaucoup plus utiles.

Exemples de données semi-structurées :

Les données semi-structurées se présentent sous une multitude de formats, chacun ayant son propre ensemble d'applications. Certaines sont à peine structurées, tandis que d'autres ont une structure hiérarchique assez sophistiquée.

CSV

Les trois principaux langages utilisés pour interagir ou transférer des données d'un serveur web à un client sont CSV, XML et JSON (c'est-à-dire ordinateur, Smartphone, etc.).

CSV signifie "valeurs séparées par des virgules", et les données sont exprimées sous la forme de Lucy, Jessica et Anthony. Il peut être exprimé de la même manière que les fichiers Excel, mais avec une seule colonne.

Courriel

Le courrier électronique est sans doute le type de données semi-structurées le plus courant, car nous en recevons tous régulièrement. Les messages électroniques comprennent des données structurées telles que le nom, l'adresse électronique, le destinataire, la date, l'heure, etc., et sont également organisés en dossiers tels que la boîte de réception, les messages envoyés, la corbeille, etc.

Même si la plupart des logiciels de messagerie vous permettent d'effectuer des recherches par mot-clé ou autre texte, les données contenues dans chaque e-mail ne sont pas structurées. Les courriels peuvent offrir une pléthore de possibilités d'exploration de données aux entreprises pour analyser les réactions des clients, s'assurer que le service à la clientèle est opérationnel et aider à la création de matériel de marketing.

HTML

HTML (Hyper Text Markup Language) ou (langage de balises pour l'hypertexte) est un langage hiérarchique. Le HTML est utilisé pour créer des sites Web et visualiser des informations. Les commentaires utilisés pour afficher du texte et des images sur un écran d'ordinateur fournissent la semi-structure du HTML, mais le texte et les images eux-mêmes ne sont pas structurés.

Les pages Web

Les pages Web sont conçues pour être facilement accessibles avec des onglets tels que Accueil, À propos de nous, Blog, Contact, etc., ou des liens vers d'autres pages dans le texte, pour aider les utilisateurs à trouver les informations dont ils ont besoin. Bien sûr, tout cela est écrit en HTML, mais on ne le voit pas sur l'écran de l'ordinateur. Et le texte et les données de chacune de ces pages ne sont pas structurés.

Bases de données NoSQL

Les bases de données non relationnelles sont communément appelées bases de données NoSQL ("not only structured query language" ou "non SQL"), les types les plus courants étant les bases de données de type document, clé-valeur, large colonne et graphe. Ce sont des dispositifs de stockage de données polyvalents, car ils peuvent stocker à la fois des données structurées et non structurées.

Elles sont idéales pour les données semi-structurées, car elles sont facilement modulables. Une seule couche supplémentaire de structure (sujet, valeur, type de données, etc.) peut faciliter la recherche et le traitement des données non structurées.

Échange de données informatisées (EDI)

L'EDI est la transmission électronique d'ordinateur à ordinateur de documents commerciaux tels que les bons de commande, les factures et les documents d'inventaire précédemment transmis sur papier. Comme l'EDI utilise de nombreux formats standard, notamment ANSI, EDIFACT, TRADACOMS et ebXML, les entreprises doivent utiliser le même format lorsqu'elles communiquent par EDI. L'EDI permet une transmission de documents beaucoup plus rapide et moins coûteuse. Bien que chaque format soit destiné à être facilement traité et compris par des machines, les données contenues dans chaque transmission ne sont pas structurées.

Analyser les données semi-structurées :

Il est moins difficile d'interagir avec des données semi-structurées qu'avec des données non structurées, mais cela crée tout de même des défis. Les conceptions d'analyse de texte peuvent désormais décomposer et analyser instantanément les données textuelles semi-structurées et non structurées afin d'en tirer de puissantes informations, grâce à la technologie d'apprentissage automatique guidée par l'IA.

Avantages et inconvénients des données semi-structurées :

Les données semi-structurées ne sont en effet pas limitées à une seule architecture. Ainsi, par exemple, une base de données NoSQL pourrait même contenir n'importe quel format de données et peut être facilement mise à l'échelle pour stocker des quantités massives de données. L'inconvénient est que cela rend l'analyse des données beaucoup plus difficile. Elles doivent être traitées manuellement (ce qui prend des centaines d'heures de travail) ou d'abord structurées dans un format compréhensible par les ordinateurs.

Les données semi-structurées sont beaucoup plus stockables et mobiles que les données entièrement non structurées, mais leur coût de stockage est généralement beaucoup plus élevé que celui des données structurées.

Les données semi-structurées sont polyvalentes dans la mesure où elles permettent de commencer à modifier le schéma. Toutefois, le schéma et les données sont souvent trop étroitement liés, de sorte que, lorsque vous effectuez des requêtes, vous devez avant tout savoir quelles données vous recherchez.

Ce cours propose d'étudier les modèles de données semi-structurées comme une alternative aux modèles de données fortement structurées, en particulier quand il s'agit de traiter des données hétérogènes. Il introduit également les modèles de contrôle d'accès aux documents XML ainsi que les problèmes de recherche encore ouverts dans le domaine.

Donc **les objectifs de ce cours** visent principalement à maîtriser les modèles, les opérations et les techniques liés à l'exploitation des données semi-structurées, à savoir : la définition et la vérification des documents XML, les langages de requêtes, le contrôle d'accès aux documents XML,…

2 Documents et hyper documents multimédias

2.1 Les documents

2.1.1 Introduction

Il est important de comprendre comment les caractères et les documents sont représentés et modélisés pour pouvoir estimer correctement leurs besoins de stockage. Afin d'introduire la manipulation des documents, nous commençons par une considération épistémologique. Selon Umberto Eco, un **document** (mot venant du latin docere qui signifie enseigner) a 3 significations :

1. Ce que l'auteur veut exprimer (Intentio auctoris).
2. Le sens "propre" du document (Intentio operas).
3. Le sens compris par le consommateur (Intentio lectoris).

Un **document** est représenté comme un ensemble structuré d'informations. Ces informations peuvent être de natures différentes. Elles sont agencées sous forme d'une arborescence et définissent une certaine hiérarchie. L'architecture du document décrit les connexions entre les éléments individuels représentés sous forme de modèles (par exemple, modèle de présentation, modèle de manipulation).

2.1.2 Modélisation des documents spécifiques

Lors de la modélisation de documents, une approche consiste à séparer la **structure logique** de la **structure physique**. Plusieurs structures physiques peuvent être associées à une structure logique en fonction de la plate-forme et du périphérique de sortie. On parle de modèle logique et d'instances de document logique spécifiques. On parle aussi de modèle physique et d'instances de documents physiques spécifiques.

Si on pense à un contenu HTML ou peut-être mieux encore, à un contenu XML (langage de balisage extensible) et ses vues sur un écran haute résolution de 32 pouces par rapport à une vue sur un écran de smartphone. Le contenu est modélisé dans des structures logiques, les vues sont composées d'informations physiques.

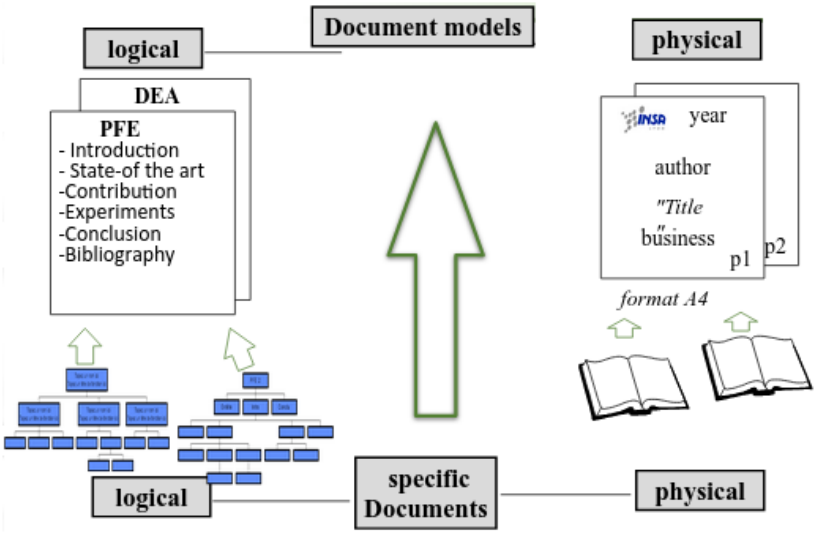


Figure 2 Modèles de documents logiques et physiques et leurs instances spécifiques.

Les modèles logiques et physiques sont composés d'objets qui peuvent être élémentaires ou composés. Les objets logiques sont transformés en objets physiques à l'aide de feuilles de style de formatage (stylesheets).

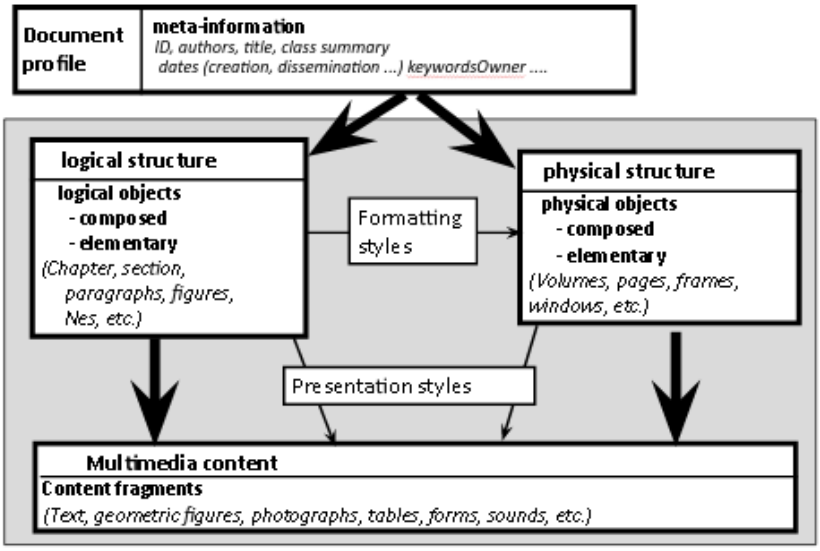


Figure 3 Modélisation de documents

Prenons une lettre comme exemple :

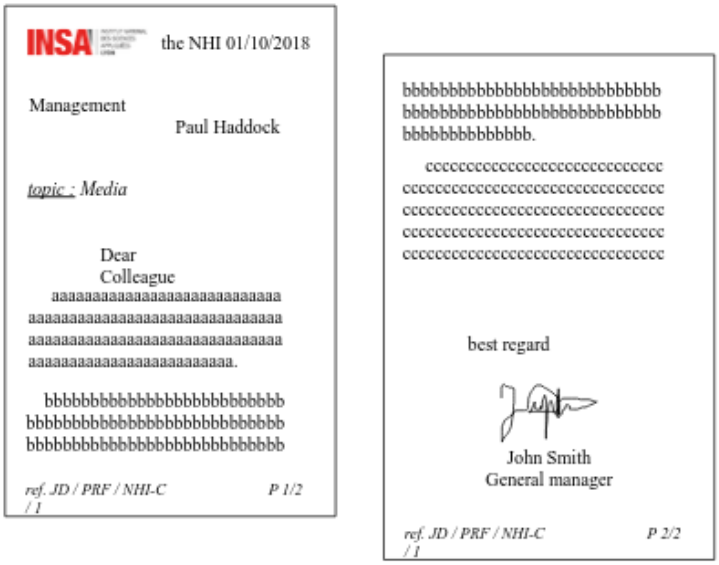


Figure 4 Une lettre type

La **structure logique** peut être composée d'un en-tête (header), le corps (body) et le pied de page (end). Chacun de ces objets est un objet composé. L'en-tête à son tour est composé d'une date, d'un expéditeur, d'un destinataire, d'un sujet et d'un titre. Le corps est une liste de paragraphes tandis que la fin est composée d'une formule, d'une signature, de noms et de titres.

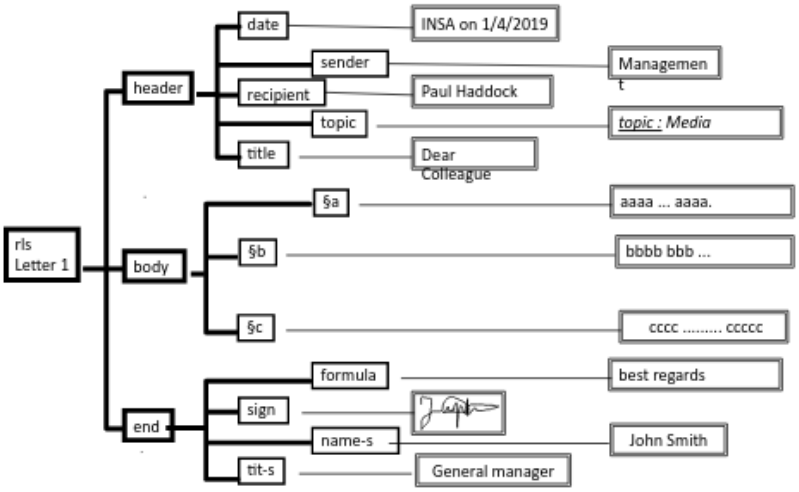


Figure 5 Structure logique d'une lettre type

Une **structure physique** peut être rendue sur des pages et être structurée comme suit : Sur la première page, il existe différents cadres pour l'en-tête, le corps et le pied de page. Le cadre d'en-tête contient d'autres cadres pour un logo, la date, l'expéditeur, le destinataire, le sujet et le titre. Les autres pages n'ont pas de cadres d'en-tête, mais seulement des cadres de corps et de pied de page.

Certains cadres peuvent être associés directement aux objets logiques correspondants, d'autres contiennent des informations spécifiques, comme le journal ou le numéro de page.

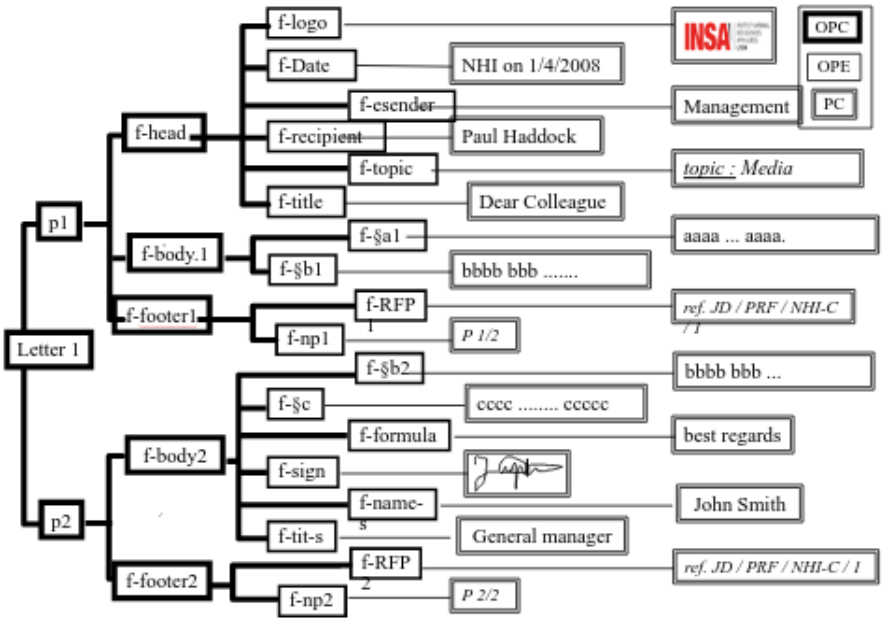


Figure 6 Document spécifique

Qu'il s'agisse d'une lettre de deux pages ou d'une lettre d'une page, elles ont toutes la même structure logique.

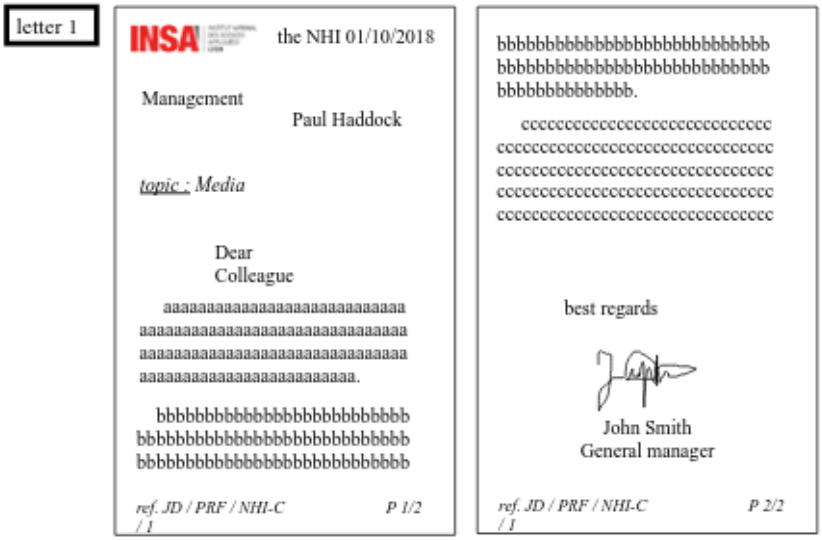


Figure 7 Exemple de lettre de deux pages

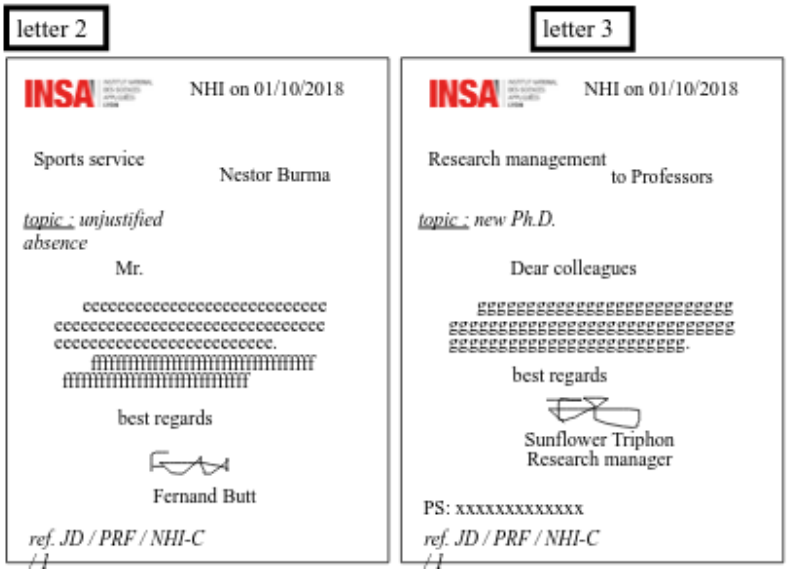


Figure 8 Exemples de lettres d'une page

Examinons de plus près une représentation plus formelle de la structure logique, en précisant les cardinalités et l'ordre des objets. Ici, nous avons l'en-tête qui est une séquence de Date, Expéditeur (Sender), Destinataire (Recipient), Sujet (Topic) et Titre du Destinataire (Recipient-title). En outre, le destinataire est facultatif et peut être répété, tandis que le sujet est facultatif.

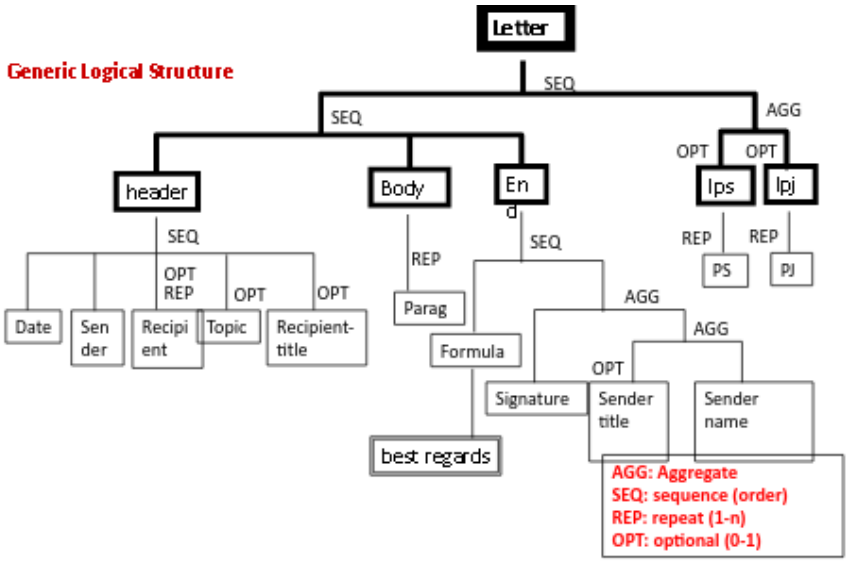


Figure 9 Représentation de la structure logique générique des lettres

2.1.3 Modélisation des classes de documents

Pour la structure physique basée sur les pages, nous définissons des **cadres** (frames) dans les pages.

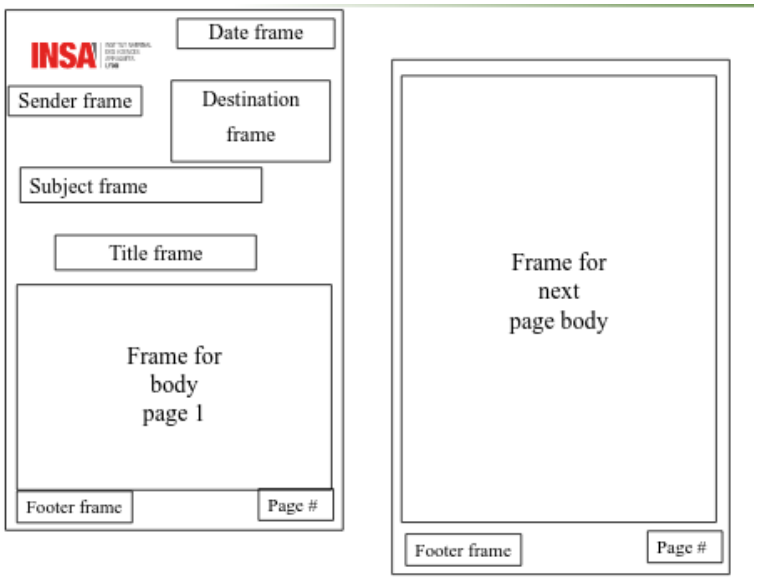


Figure 10 Exemple de définitions de cadre dans les pages de lettres

Cela induit à son tour le **modèle hiérarchique** suivant :

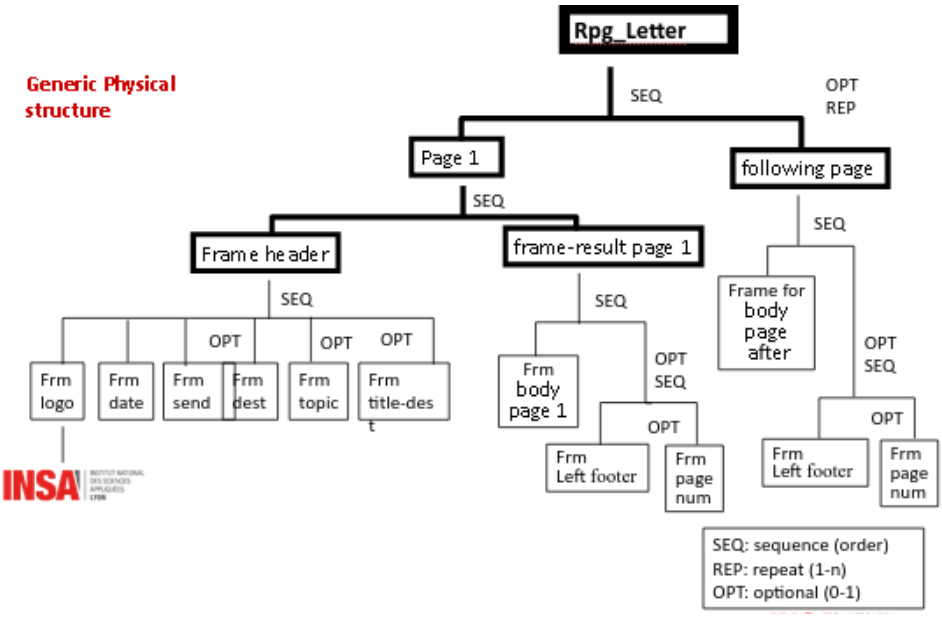


Figure 11 Structure physique générique de la lettre

Pour modéliser des documents, nous pouvons spécifier un **modèle logique** (généralement un) ou une structure et (généralement plusieurs correspondants) **des modèles ou structures physiques**.

2.2 Les hyper documents

Hypertextes et hypermédias peuvent être classés dans une catégorie plus large que Jean-Pierre Balpe a défini comme « **hyperdocuments** » : tout contenu informatif informatisé dont la caractéristique principale est de ne pas être assujetti à une lecture préalablement définie mais de permettre un ensemble plus ou moins complexe, plus ou moins divers, plus ou moins personnalisé de lectures [BAL 90]. Selon cette définition, le **domaine** des **hyperdocuments** comprend plusieurs produits le représentant, ceux-ci sont : les bases de données, les hypertextes, les hypermédias et les hypertextes fictionnels (scénarios interactifs).

Il est important de distinguer un système hypertexte qui est un logiciel d'un hyperdocument textuel qui est une information gérée par un système hypertexte. De même un système hypermédia qui est un logiciel qui permet l'utilisation de données multimédias forment un hyperdocument multimédia qui à son tour est une information gérée par un système hypermédia.

Les informations (textuelles ou autres) contenues dans un **hyperdocument** sont divisées en fragments, les nœuds, reliées entre eux par des liens permettant d'accéder à un nœud à partir d'un autre. Les nœuds et les liens sont deux composantes fondamentales et ils forment une réticulation qui permet de multiples parcours de lecture. A chaque nœud qu'il consulte, le lecteur doit donc choisir celui auquel il accèdera ensuite. Donc, des chemins pour guider le lecteur sont nécessaires.

Un **nœud** est une unité sémantique. Les nœuds sont associés à des contenus tels que : texte (avec ou sans options), graphiques (géométriques ou photographiques), et son ou vidéo. Les **liens** (typés) entre nœuds définissent la structures sont : hypergraphe, graphique et arbre (i.e. document).

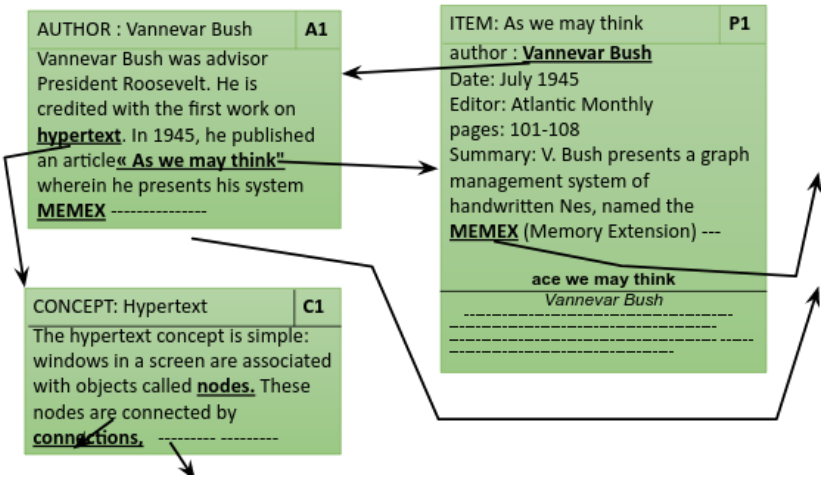


Figure 12 Un exemple d'hyperdocument

2.3 Les contenus multimédias

Le fichier multimédia peut être un produit numérique intégrant plusieurs médias sur un même support (textes, images, sons), un nouveau domaine de création, de diffusion et de consultation. Produire du multimédia, c'est créer stocker, organiser, annoter, lier et synchroniser des fichiers numériques. Les éléments de données multimédia et leurs différentes formes sont décrits ci-dessous :

- **Texte** : devenu plus important que jamais avec l'explosion récente d'Internet et du WWW. Le Web est un langage de balisage hypertexte conçu à l'origine pour afficher des documents texte simples sur des écrans d'ordinateur, avec des images graphiques occasionnelles ajoutées en tant qu'illustrations.
- **Audio** : Le son est peut-être l'élément le plus multimédia. Il peut procurer le plaisir d'écoute de la musique, l'accent saisissant d'effets spéciaux ou l'ambiance d'un fond d'ambiance.
- **Images** : Les images, qu'elles soient représentées de manière analogique ou numérique, jouent un rôle essentiel dans un multimédia. Il s'exprime sous la forme d'une image fixe, d'une peinture ou d'une photographie prise à l'aide d'un appareil photo numérique.
- **Animation** : C'est l'affichage rapide d'une séquence d'images d'illustrations 2D ou de positions de modèles. Il s'agit d'une illusion d'optique de mouvement due au phénomène de persistance de la vision, et peut être créée et démontrée de plusieurs façons.
- **Vidéo** : La vidéo numérique a supplanté la vidéo analogique comme méthode de choix pour faire de la vidéo à usage multimédia. La vidéo dans le multimédia est utilisée pour représenter des images animées en temps réel dans un projet multimédia.

3 Noyau XML

3.1 Introduction à XML

XML, pour eXtensible Markup Language (langage de balisage extensible) est un moyen de représenter des informations dans des fichiers texte. Son but est d'échanger et de stocker des informations. Il est conçu pour être à la fois lisible par l'homme et la machine et auto-descriptif. C'est un langage de balisage comme HTML. XML est présent derrière des technologies telles que : communication Ajax, fichiers de configuration (MavenMS Visual Studio, MS Office), description de l'interface graphique humaine, Flux RSS, web sémantique (RDF), services Web (WSDL), Graphiques Web (SVG). Il est également derrière : le Web (les pages Web sont souvent des instances XML), de nombreux CMS (systèmes de gestion de contenu), dans les supports de documentation industrielle (DocBook), dans l'édition numérique (TEI, ePub, docx), …

3.2 Structure XML de base

XML est exactement une syntaxe "squelette" pour créer des langages de balisage. Sa syntaxe est identique à celle de XHTML : `<element attribute="value">content</element>`

Dans l'exemple suivant, on peut voir les principaux composants d'un document XML : déclaration, balises : balises de début et de fin, attributs et valeurs d'attribut et entités XML. Tout ce qui se trouve entre une balise de début et une balise de fin est un élément XML.

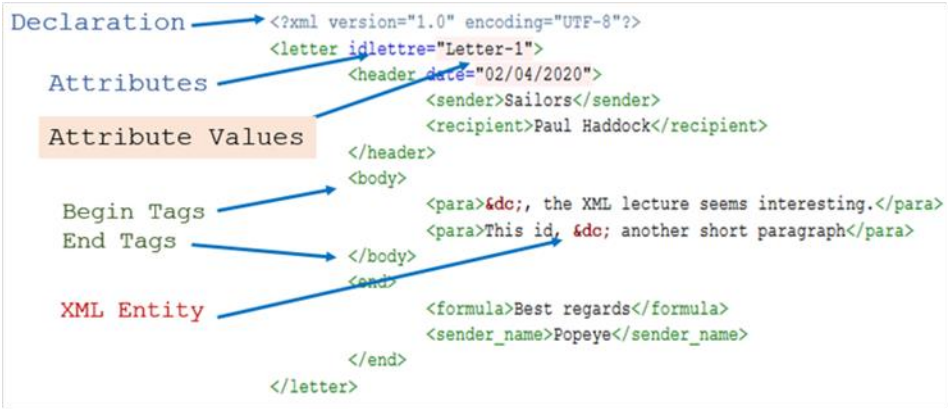


Figure 13 Document XML simple et ses composants

Document XML

Les premiers concepts principaux des documents XML sont la déclaration, les balises, les éléments, les attributs, les commentaires, les entités de caractères, les sections CDATA, les espaces blancs, le traitement, l'encodage et la validation XML. Un document XML est une unité de base d'informations XML composée d'éléments et d'autres balises dans un ensemble ordonné.

Voici un document XML simple avec ses parties :

Figure 14 Les différentes parties d'un document XML

Le **prologue** du document se trouve en haut du document, avant l'élément racine. Cette section contient une déclaration XML et une déclaration du type de document.

Les **éléments** de document sont les blocs de construction du XML. Ils divisent le document en une hiérarchie de sections, chacune servant un objectif spécifique. Un document peut être séparé en plusieurs sections afin qu'elles puissent être rendues différemment, ou utilisées par un moteur de recherche. Les éléments peuvent être des conteneurs, avec une combinaison de texte et d'autres éléments et peuvent également contenir des attributs.

Les documents XML forment une structure **arborescente** qui commence à "la racine" et se ramifie vers "les feuilles". Donc, dans l'exemple suivant, notre exemple de lettre est représenté sous forme d'arbre. La racine est l'élément « letter » et il possède trois "branches" : header, body et end. La branche de "header" (enfant de l'élément letter) a à son tour 2 "feuilles" ou 2 enfants : "sender" et "recipient".

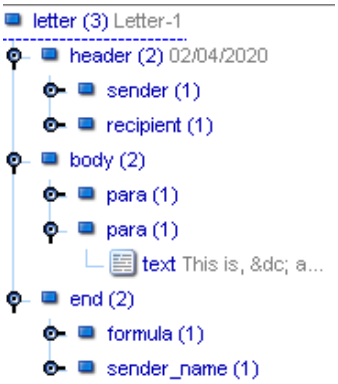


Figure 15 Document XML sous forme d'arborescence

Tous les éléments peuvent avoir des sous-éléments (éléments enfants). Les termes parent, enfant et frère sont utilisés pour décrire les relations entre les éléments. Les parents ont des enfants. Les enfants ont des parents. Les frères et sœurs sont des enfants au même niveau, avec le même parent. Tous les éléments peuvent avoir un contenu textuel (Marins, Paul Haddock, …) et des attributs (date="02/04/2020").

La déclaration XML (parfois appelée le prologue XML) est facultative, mais si nécessaire, elle doit figurer sur la première ligne du document. Pour éviter les erreurs, une bonne pratique consiste à préciser l'encodage, couramment et par défaut cet encodage est UTF-8. Une déclaration XML simple est donnée par l'exemple suivant : `<?xml version="1.0" encoding="UTF-8"?>`

Balises et éléments XML

Un document XML est composé d'un ou plusieurs éléments XML délimitée soit par des **balises** de début et de fin, soit pour les éléments vides, par une balise d'élément vide. Les **éléments** XML

peuvent être définis comme des blocs de construction d'un document XML. Les éléments peuvent se comporter comme des conteneurs pour contenir du texte, d'autres éléments, des attributs, des objets multimédias ou éléments vides.

L'élément XML suivant est entouré par la balise header. Les balises XML sont entourées des deux caractères < et >.

```
<header date="02/04/2020">
               <sender name= "Sailors"/>
               <recipient>Paul Haddock </recipient>
</header>
```

La balise de début <nom_balise> est souvent appelée balise d'ouverture et la balise de fin </ nom_balise> est souvent appelée balise de fermeture. **Les noms doivent être identiques**.

Chaque élément XML doit être fermé avant que son élément parent ne soit fermé. Chaque balise ouvrante à l'intérieur d'un élément XML est considérée comme étant la balise ouvrante d'un de ses éléments enfants. Un document XML doit être convertible en arborescence, où chaque élément n'a qu'un seul parent. Les balises XML sont sensibles à la casse.

Les attributs XML sont placés à l'intérieur de la balise d'ouverture : <sender name="Sailors"/>. Ici, name="Sailors" est un attribut XML.

Un élément vide (élément sans contenu) a la syntaxe suivante : `<element_name attribute1 attribute2 ... />`. Dans les exemples précédents, <sender name= "Sailors"/> est un élément XML vide.

Attributs XML

Les **attributs** font partie des éléments XML. Un élément peut avoir plusieurs attributs uniques. Les noms d'attribut doivent être uniques pour un élément XML donné. Un attribut XML est toujours une paire nom-valeur. Ils définissent généralement les propriétés des éléments XML.

Les attributs sont utilisés pour distinguer les éléments du même nom, lorsque vous ne souhaitez pas créer un nouvel élément pour chaque situation.

Dans l'exemple suivant, date="02/04/2020" et name="Sailors" sont des attributs.

```
        <header date="02/04/2020">
               <sender name= "Sailors"/>
               <recipient>Paul Haddock </recipient>
        </header>
```

Il existe trois types d'attributs décrits ci-dessous :

- Attribut de type chaînes : ce type d'attribut prend n'importe quel littéral de chaîne comme valeur. **CDATA** sont des données de caractères de type chaînes.
- Attribut de type énuméré : distribué en deux types :
    - Type de notation : qu'un élément sera référencé à une **NOTATION** déclarée ailleurs dans le document XML.
    - Énumération : cet attribut est utilisé pour spécifier une liste particulière de valeurs qui correspondent aux valeurs d'attribut.
- Attribut de type tokenisé : cet attribut est en outre distribué dans de nombreux types :
    - **ID** : utilisé pour identifier l'élément.
    - **IDREF** : utilisé pour référencer un ID qui a déjà été nommé pour un autre élément.
    - **IDREFS** : utilisé pour référencer tous les ID d'un élément.
    - **ENTITY** : utilisé pour indiquer l'attribut qui représentera une entité externe utilisée dans le document.
    - **ENTITIES** : utilisé pour indiquer l'attribut qui représentera les entités externes utilisées dans le document.

Commentaires XML

Un commentaire en xml commence par <!-- et se termine par -->. Vous pouvez ajouter des notes textuelles sous forme de commentaires entre les caractères. Les commentaires ne peuvent pas apparaître avant la déclaration XML et ne doivent pas apparaître dans les valeurs d'attribut. En plus, un commentaire ne doit pas être imbriqué dans un autre.

Entités XML

Les entités XML sont un moyen de représenter un élément de données dans un document XML, au lieu d'utiliser les données elles-mêmes. En général, les entités XML sont des espaces réservés. Ils permettre l'utilisation de caractères prédéfinis dans les données XML et de caractères qui ne sont pas présents sur un clavier donné. Ils permettre également de définir des raccourcis, des abréviations de chaînes de caractères souvent utilisées. Nous les appelons entités personnalisées. Ces entités sont définies dans la DTD (Document Type Definition).

Il existe trois types d'entités de caractères : les entités de caractères prédéfinies, de caractères numérotées et de caractères nommés.

Diverses entités prédéfinies sont construites en fonction des spécifications du langage XML. Par exemple, les entités &lt, &gt, &amp, &quot et &apos représentent les caractères <, >, &, " et '. Ces caractères sont des métacaractères utilisés pour désigner les balises XML. Les entités sont utilisées pour définir des raccourcis vers des caractères spéciaux ou des textes plus longs.

Les entités peuvent être déclarées internes, dans le document XML ou externe (dans un DTD). Il existe également des entités de paramètres, utilisées pour abréger les fragments DTD.

XML fournit une nouvelle construction exclusivement pour une utilisation dans DTDs, l'entité de paramètre, qui est mentionné par une référence d'entité de paramètre. Les entités de paramètres se comportent et sont déclarées presque exactement comme une entité générale. Cependant, ils utilisent un % au lieu d'un &, et ils ne peuvent être utilisés que dans un DTD tandis que les entités générales ne peuvent être utilisées que dans le contenu du document.

Considérez un document immobilier, fournissant des informations sur les maisons, les appartements,… les éléments de leur document peuvent ressembler :

```
<!ELEMENT apartment (address, surface, rooms, baths, availability_date)>
<!ELEMENT bungallow (address, surface, rooms, baths, availability_date)>
<!ELEMENT cottage   (address , surface, rooms, baths, price)>
<!ELEMENT pavilion  (address, surface, rooms, baths, price)>
<!ELEMENT house     (address, surface, rooms, baths, price)>
```

Une référence d'entité de paramètre est déclarée comme une référence d'entité générale. Toutefois, un signe de pourcentage supplémentaire est placé entre le mot-clé <!**ENTITY** et le nom de l'entité.

```
<!ENTITY % residential_part "address, footage, rooms, baths">
<!ENTITY % rental_part       "availability_date">
<!ENTITY % purchase_part     "price ">
```

Les entités de type paramètre sont référencées de la même manière que les entités générales, mais avec le signe pourcent au lieu de & :

```
<!ELEMENT apartment (%residential_part;, %rental_part;)>

<!ELEMENT bungallow (%residential_part; , %rental_part;)>
<!ELEMENT cottage   (%residential_part; , %purchase_part;)>
<!ELEMENT pavilion   (%residential_part;,%purchase_part; )>
<!ELEMENT house     (%residential_part; , %purchase_part;)>
```

3.3 Domaines nominaux

Un espace de nommage (namespace) définit une famille de noms afin d'éviter les confusions entre des éléments qui auraient le même nom mais pas le même sens. Cela arrive quand le document XML modélise les informations de plusieurs domaines.

Les espaces nominaux permettent de qualifier de manière unique tous les objets XML (éléments, attributs, ...). Ils permettent la coexistence dans le même document d'objets avec le même nom mais avec une connotation différente: comme:

```
Kitchen table -> kitchen:table
Table (SQL) -> sql:table
```

Ou :

ex1: <fn> tag in the 10"> example of footnote on page </fn>
ex2: <fn> tag in the DTD "mathml.dtd" refers to a "function"

```
<fn>
<apply>
       <int/>
       <bvar> <ci> x </ci> </bvar>
       <lowlimit> <cn> 0 </cn> </lowlimit>
       <uplimit> <cn> 1 </cn> </uplimit>
</apply>
</fn>
```

Pour éviter toute confusion, nous pouvons déclarer des espaces de nommages :

- Attribut **xmlns**
- Syntaxe de la déclaration : xmlns, prefix = URI
- *URI* = **U***niforme* **R***esource* **I***dentifier*
- Pour associer un *prefix* à une URI : une DTD, un fragment DTD, un schéma XML, un fragment de schéma, etc.
- utilisation de la syntaxe : *prefix* ":" tag

L'instance XML ressemblerait à ceci :

```
<report>
xmlns math = "http://www.w3.org/...../REC-MathML.dtd"
xmlns bt = "http://foo.bar.org/xml/...../basic-text.dtd" >
<!-- further  -->
     < bt:fn id = "Note 10">
                        example footnote
                  </bt:fn>
<!--  further  -->
   < math:fn>
     <math:apply>
       <math: int/>
        <math:bvar> < math:this > X </ math:this > </ math:bvar >
                  <math:lowlimit> <math:cn> 0 </math:cn> </math:lowlimit>
                  <math:uplimit> <math:cn> 1 </math:cn></math:uplimit>
    </math:apply>
   </math:fn>
<!-- further  -->
</report>
```

3.4 Schémas XML

La validation permet de vérifier la structure et le contenu d'un document XML avant de commencer à le traiter : les bons éléments avec les bons attributs présents aux bons endroits. Il existe deux techniques : La plus simple basée sur les **Document Type Definitions (DTD)** venant de la norme SGML, La plus complète basée sur des **XML Schema**.

3.4.1 XML – DTD

La déclaration de type de document XML, communément appelée DTD, est un moyen de décrire précisément le langage XML. Les DTD vérifient le vocabulaire et la validité de la structure des documents XML par rapport aux règles grammaticales du langage XML approprié.

Une DTD XML peut soit être spécifiée à l'intérieur du document, soit être conservée dans un document distinct et être ensuite appréciée séparément.

La syntaxe de base d'une DTD est la suivante :

```
<!DOCTYPE element DTD identifier
[
   declaration1
```

```
   declaration2
   ........
]>
```

Dans la syntaxe ci-dessus,

- La DTD commence par le délimiteur <!DOCTYPE.
- Un élément indique à l'analyseur d'analyser le document à partir de l'élément racine spécifié.
- L'identifiant DTD (DTD identifier) est un identifiant pour la définition du type de document, qui peut être le chemin d'accès à un fichier sur le système ou l'URL d'un fichier sur Internet. Si la DTD pointe vers un chemin externe, elle est appelée sous-ensemble externe.
- Les crochets [ ] entourent une liste facultative de déclarations d'entité appelée sous-ensemble interne.

Une DTD est appelée **DTD interne** si des éléments sont déclarés dans les fichiers XML. Pour le référencer en tant que DTD interne, l'attribut *standalone* dans la déclaration XML doit être défini sur "yes". Cela signifie que la déclaration fonctionne indépendamment d'une source externe.

Voici la syntaxe de la DTD interne où root-element est le nom de l'élément racine et element-declarations est l'endroit où vous déclarez les éléments : <!DOCTYPE root-element [element-declarations]>

Voici un exemple simple de DTD interne où la déclaration DOCTYPE doit apparaître au début du document (précédée uniquement de l'en-tête XML) et suivie du corps de la DTD, où des éléments, des attributs, des entités et des notations sont déclarés. Le nom dans la déclaration de type de document doit correspondre au type d'élément de l'élément racine. <!ELEMENT name (#PCDATA)> définit le nom de l'élément comme étant de type "#PCDATA". Ici, #PCDATA signifie des données textuelles analysables.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Dans la **DTD externe**, les éléments sont déclarés en dehors du fichier XML. Ils sont accessibles en spécifiant les attributs système qui peuvent être soit le fichier .dtd légal, soit une URL valide. Pour le référencer en tant que DTD externe, l'attribut *standalone* dans la déclaration XML doit être défini sur "no". Cela signifie que la déclaration inclut des informations provenant de la source externe.

Voici la syntaxe pour DTD externe : <!DOCTYPE root-element SYSTEM "file-name"> où nom-fichier est le fichier avec l'extension .dtd. L'exemple suivant montre l'utilisation d'une DTD externe :

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Le contenu du fichier DTD **address.dtd** est comme indiqué :

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

3.4.2 Schéma XML

Le schéma XML est un langage utilisé pour exprimer des contraintes sur les documents XML. Il existe de nombreux langages de schéma, par exemple XSD (XML schema definition). Un schéma XML est utilisé pour définir la structure d'un document XML. C'est comme DTD mais offre plus de contrôle sur la structure XML.

Un document XML est dit "**bien formé**" s'il contient la syntaxe correcte. Un document XML **valide** et bien formé est un document qui a été validé par rapport au schéma. Il existe des sites web comme http://www.xmlvalidation.com ou des applications comme par exemple XMLValidator pour valider le fichier XML par rapport au schéma ou à la DTD.

Ci-dessous, un exemple de schéma XML *employee.xsd.*

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.univ-usto.dz"
xmlns="http:// www.univ-usto.dz "
elementFormDefault="qualified">
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Voyons le fichier xml *employee.xml* en utilisant le schéma XML ou le fichier XSD.

```
<?xml version="1.0"?>
<employee
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com employee.xsd">
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@univ-usto.dz</email>
</employee>
```

Description du schéma XML

- <**xs:element name="employee"**> : Il définit le nom de l'élément "employee".
- <**xs:complexType**> : Il définit que l'élément "employee" est de type complexe.
- <**xs:sequence**> : Il définit que le type complexe est une séquence d'éléments.
- <**xs:element name="firstname" type="xs:string"/**> : Il définit que l'élément "firstname" est de type string/text. La même chose pour l'élément "lastname" et "email".

Il existe trois **types de types de données** dans le schéma :

L'élément de **type simple** est utilisé uniquement dans le contexte du texte. Certains des types simples prédéfinis sont : xs:integer, xs:boolean, xs:string, xs:date. Par exemple :

<xs:element name = "phone_number" type = "xs:int" />

Un **type complexe** est un conteneur pour d'autres définitions d'éléments. Cela permet de spécifier les éléments enfants qu'un élément peut contenir et de fournir une certaine structure dans des documents XML. Dans l'exemple ci-dessus, l'élément Adress est composé d'éléments enfants. Il s'agit d'un conteneur pour d'autres définitions <**xs:element**>, qui permet de construire une hiérarchie simple d'éléments dans le document XML.

```
<xs:element name = "Address">
   <xs:complexType>
      <xs:sequence>
```

```
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
         <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Avec le **Type Global**, un seul type peut être défini dans un document, qui peut être utilisé par toutes les autres références. Par exemple, supposons que vous souhaitiez généraliser la personne et la société pour différentes adresses de la société. Dans ce cas, on peut définir un type général comme suit :

```
<xs:element name = "AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Utilisons maintenant ce type dans notre exemple comme suit :

```
<xs:element name = "Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone1" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name = "Address2">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone2" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Au lieu de devoir définir le nom et la société deux fois (une fois pour Address1 et une fois pour Address2), nous avons maintenant une seule définition. Cela simplifie la maintenance, c'est-à-dire que si vous décidez d'ajouter des éléments "Code postal" à l'adresse, vous devez les ajouter à un seul endroit.

Les attributs

Les attributs dans XSD fournissent des informations supplémentaires dans un élément. Les attributs ont un nom et une propriété de type comme indiqué ci-dessous :

```
<xs:attribute name = "x" type = "y"/>
```

3.5 Exercices

Exercice 1 : soit le code XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue>
      <cours code="INFO-ISIL-600">
         <titre>Technologies XML
         <professeur email="youcef.benmohamed@univ-usto.dz>Youcef.
Benmohamed</prof>
         <assistant email=m.naili@univ-usto.dz>Meriem Naili</assistant>
         <credit >3 <unit type="ECTS" /></credit>
      </cours>
</catalogue>
<catalogue>
       <cours code="INFO- ISIL-601" <!-- Ce cours ne sera plus donné cette
année-->>
          <titre>  web sémantique </titre>
          <professeur email="a.hansali@univ-usto.dz">Abdelkader Hansali
          <assistant email="a.mekhaldi@univ-usto.dz">Aya Mekhaldi
</assistant>
          < assistant email="m.haddad@univ-usto.dz">Mohamed
Haddad</assistant>
           <credit>3 <unit type="ECTS"></credit>
       </Cours>
</catalogue>
```

1. Enregistrer le document fournit sous le nom « catalogues.xml ». Ce document est-il **bien formé** (i.e. respecte-t-il la syntaxe XML) ? S'il ne l'est pas, corrigez les erreurs.
2. Compléter le document pour inclure le cours de Bases de données, INFO-ISIL-602 avec 6 ECTS. Ce cours est enseigné par Yamina Djafri comme professeur, et Yacine Boudia comme assistant.
3. Compléter ce document en ajoutant plusieurs éléments personne, de sorte à reprendre les informations mentionnées dans le tableau ci-dessous.

Nom	**Laboratoire**	**email**	**Domaines de recherche**
Lamis Mezaourou	EDIS	l.mezaourou@univ-usto.dz	Systèmes d'information Bases de données scientifiques Web sémantique
Mohamed Belkheir	EDIS	m.belkheir@univ-usto.dz	Systèmes d'information Entrepôts de données Web sémantique

Exercice 2 :

Proposez une modélisation XML avec les exemples de données immobilières suivants : L'élément racine immobilier contiendra une séquence de sous-éléments agences, propriétaires, propriétés et appartements, tous avec un contenu vide.

Assurez que le document est bien formé (i.e. il respecte les règles syntaxiques du XML).

Exercice 3 :

Créer une **DTD interne** pour le document XML précédent. Assurez sa **validité**.

Exercice 4 :

Déplacez la DTD précédente vers un fichier externe et validez à nouveau le document XML.

Exercice 5 :

1. Étendre la DTD précédente de telle sorte que les éléments agences, propriétaires, propriétés et appartements peuvent désormais contenir respectivement des éléments particuliers d'agence, de propriétaire, de propriété et d'appartement.
   - Agence = nom, e-mail, téléphone.
   - Propriétaire = nom.

- Propriété = nom, description, nombre d'appartements, caractéristiques, référence du propriétaire. La description des propriétés peut contenir un texte avec des références aux appartements (par exemple : ...<refAppartement nomAppartement ="A1001"> ...</ refAppartement>...).
- Appartement = nom, description, caractéristiques, référence de la propriété. La description des appartements contient une description structurée avec un fragment XML arbitraire, mais toujours bien formé.

2. Utilisez des attributs pour les références, sinon des éléments.
3. Utilisez uniquement CDATA et #PCDATA pour le contenu et les valeurs.
4. Etendez le document XML de sorte qu'il contienne 1 exemple d'agence, 1 propriétaire, 1 propriété et 2 appartements et assurez que le document est bien formé et valide.

Exercice 6 :

Créez un document **XSD** pour le document XML immobilier de l'exercice 1. Modifier le document XML de sorte que sa validité puisse être vérifiée par rapport à ce XSD.

Exercice 7 :

1. Étendez le **Schéma XSD** précédent pour le document XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<immobilier
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Immobilier_xsd.xsd">
        <agences>
         <agence>
             <nom>Premier immobilier</nom>
             <email>info@agence-immob.dz</email>
             <téléphone>+213 777777777</téléphone>
         </agence>
        </agences>
    <propriétaires>
         <propriétaire idPropriétaire="o1">
             <nom>Amine Nemmich</nom>
         </propriétaire>
        </propriétaires>
    <propriétés>
         <propriété idPropriété="p1" refPropriétaire="o1">
             <nom>Residence Maldives</nom>
             <nombreAppartements>20</nombreAppartements>
         </propriété>
        </propriétés>
    <appartements>
         <appartement idAppartement="a1" refPropriété="p1"
niveauxComfort="E" dateAffichage="2021-07-01">
             <nom>K0101</nom>
             <description>Petit appartement sans fenêtres</description>
             <tarif>123</tarif>
         </appartement>
         <appartement idAppartement="a2" refPropriété="p1"
niveauxComfort="C">
             <nom>K0501</nom>
             <description>Appartement ordinaire avec de nombreuses
fenêtres</description>
             <tarif>1234</tarif>
         </appartement>
         <appartement idAppartement="a3" refPropriété="p1"
niveauxComfort="A" dateAffichage="2022-09-01">
             <nom>K1501</nom>
             <description>Appartement de luxe avec une immense
terrasse</description>
             <tarif>12345</tarif>
```

```
        </appartement>
       </appartements>
</immobilier>
```

2. Les éléments agences, propriétaires, propriétés et appartements peuvent désormais contenir respectivement des éléments particuliers d'agence, de propriétaire, de propriété et d'appartement.
   - Agence = nom, e-mail, téléphone.
   - Propriétaire = nom.
   - Propriété = nom, nombre d'appartements.
   - Appartement = nom, description, tarif.
3. Définissez des types complexes globaux pour les 4 entités principales.
4. Ajoutez les attributs obligatoires suivants :
   - Identificateurs des propriétaires, propriétés et appartements.
   - Référence des appartements aux propriétés auxquelles ils appartiennent.
   - Niveaux de confort des appartements.
5. Ajoutez les attributs facultatifs suivants :
   - Référence des propriétés aux propriétaires auxquels elles appartiennent.
   - Dates d'affichage des appartements.
6. Choisissez des types simples prédéfinis appropriés pour toutes les valeurs de texte et d'attribut et/ou introduisez vos propres types simples dérivés globaux/anonymes.
   - Utilisez des identificateurs basés sur la DTD pour les propriétaires, les propriétés et les appartements, ainsi que pour leurs relations mutuelles.
   - N'autorisez que les valeurs A, B, C, D, E, F pour les niveaux de confort des appartements.
   - Les tarifs des appartements sont des entiers positifs ou 0.
   - Les identifiants des appartements doivent correspondre à l'expression régulière [a-A][0-9]{1,5}.
   - Le nombre d'appartements d'une propriété doit être égal à 1500 au maximum.
   - Les noms des appartements doivent comporter au moins 5 et au plus 100 caractères.

3.6 Corrigés des exercices

Solution exercice 1 :

Pour tester si un document XML est bien formé, il respecte les règles syntaxiques du XML : écriture des balises, imbrication des éléments, entités, etc. Il suffit de lui donner l'extension « .xml » et de l'ouvrir dans navigateur, par exemple ou un éditeur XML comme XML Copy Editor.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogues>
  <catalogue>
        <cours code="INFO- ISIL-600">
           <titre>Technologies XML </titre>
           <professeur email="youcef.benmohamed@univ-usto.dz">Youcef.
Benmohamed</professeur>
           <assistant email="m.naili@univ-usto.dz">Meriem Naili</assistant>
           <credit >3 <unit type="ECTS" /></credit>
        </cours>
  </catalogue>
  <catalogue>
         <cours code="INFO- ISIL-601" >
            <titre>  web sémantique </titre>
            <professeur email="a.hansali@univ-usto.dz">Abdelkader Hansali
</professeur>
            <assistant email="a.mekhaldi@univ-usto.dz">Aya Mekhaldi
</assistant>
            <assistant email="m.haddad@univ-usto.dz">Mohamed
Haddad</assistant>
              <credit>3 <unit type="ECTS"/></credit>
         </cours>
  </catalogue>
   <catalogue>
```

```
     <cours code="INFO- ISIL-602" >
       <titre> base de donnée </titre>
       <professeur email=""> Yamina Djafri </professeur>
       <assistant email=""> Yacine Boudia  </assistant>
       <credit> 6 <unit type="ECTS"/></credit>
     </cours>
   </catalogue>
   <personne email="l.mezaourou@univ-usto.dz" laboratoire="EDIS">
       <nom>  Lamis Mezaourou   </nom>
       <DomaineRecherche> Systèmes d'information </DomaineRecherche>
       <DomaineRecherche> Bases de données scientifiques
       </DomaineRecherche>
       <DomaineRecherche> Web sémantique</DomaineRecherche>
   </personne>
   <personne email="l.belkheir@univ-usto.dz" laboratoire="EDIS">
      <nom>  Larbi Belkheir </nom>
      <DomaineRecherche> Systèmes d'information </DomaineRecherche>
      <DomaineRecherche> Entrepôts de données </DomaineRecherche>
   </personne>
</catalogues>
```

Solution exercice 2 :

Immobilier.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<immobilier>
    <agences></agences>
    <propriétaires></propriétaires>
    <propriétés></propriétés>
    <appartements></appartements>
</immobilier>
```

Solution exercice 3 :

Les DTD permettent de spécifier ce que doit contenir un document XML : balises, structuration, attributs et types. Les DTD sont simples, valables pour une première validation.

Pour vérifier la validité de document XML, il suffit d'ouvrir le fichier Immobilier.xml dans XML Copy Editor et appuyer sur la touche F5 ou le menu XML, Valider, DTD/XML Schema. Le message en bas signale que le document est valide.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE immobilier
[
    <!ELEMENT immobilier (agences, propriétaires, propriétés,
appartements)>
    <!ELEMENT agences EMPTY>
    <!ELEMENT propriétaires EMPTY>
    <!ELEMENT propriétés EMPTY>
    <!ELEMENT appartements EMPTY>

]>
<immobilier>
    <agences></agences>
    <propriétaires></propriétaires>
    <propriétés></propriétés>
    <appartements></appartements>
</immobilier>
```

Solution exercice 4 :

Ouvrir les deux documents dans XML Copy Editor et répéter la procédure de l'exercice précédent. La DTD a été mentionnée à l'intérieur du document par l'instruction <!DOCTYPE immobilier SYSTEM "Immobilier_dtd.dtd">.

Immobilier.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE immobilier SYSTEM "Immobilier_dtd.dtd">
<immobilier>
    <agences></agences>
    <propriétaires></propriétaires>
    <propriétés></propriétés>
    <appartements></appartements>
</immobilier>
```

Immobilier_dtd.dtd :

```
<!ELEMENT immobilier (agences, propriétaires, propriétés, appartements)>
<!ELEMENT agences EMPTY>
<!ELEMENT propriétaires EMPTY>
<!ELEMENT propriétés EMPTY>
<!ELEMENT appartements EMPTY>
```

Solution exercice 5 :

Document XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE immobilier SYSTEM "Immobilier_dtd.dtd">
<immobilier>

    <agences>
        <agence>
            <nom>Les meilleures propriétés à Oran</nom>
            <email>info@best-oran.com</email>
            <téléphone>+213 6.66.66.66.66</téléphone>
        </agence>
    </agences>

    <propriétaires>
        <propriétaire>
            <nom>Immobilier à Oran</nom>
        </propriétaire>
    </propriétaires>

    <propriétés>
        <propriété propriétaire="Immobilier à Oran">
            <nom>Gratte-ciel de luxe</nom>
            <description>
                Un total de 154 appartements sur 26 étages.
                Il y a 2 appartements de luxe sur le toit, avec jardin
d'été.
                Un gardien est présent 24h/24 et 7j/7 à la réception.
                C'est à proximité d'une station de tramway, vue mer.
                La police au coin de la rue. <refAppartement.
nomAppartement="A2601">A2601</refAppartement.>,
                qui est le meilleur que nous proposons actuellement..
            </description>
            <nombreAppartements>154</nombreAppartements>
            <caractéristiques>
                <caractéristique>ascenseur</caractéristique>
                <caractéristique>parking</caractéristique>
                <caractéristique>securité</caractéristique>
                <caractéristique>vue sur la mer</caractéristique>
                <caractéristique>tramway</caractéristique>
                <caractéristique>parking</caractéristique>
            </caractéristiques>
```

```
        </propriété>
    </propriétés>

    <appartements>
        <appartement propriété="Gratte-ciel de luxe">
            <nom>A1001</nom>
            <information>Un petit appartement sans fenêtres. Punaises de
lit incluses.</information>
            <caractéristiques/>
        </appartement>
        <appartement propriété="Gratte-ciel de luxe">
            <nom>A2601</nom>
            <information>
                                <description>Grande suite avec jardin sur le
toit.</description>
                                vue sur la mer, 200 mètres carres.
                                <nom>Vue mer, 200 m</nom>
                        </information>
            <caractéristiques>
                <caractéristique>propre ascenseur</caractéristique>
                <caractéristique>jardin</caractéristique>
                <caractéristique>bain</caractéristique>
                <caractéristique>2 salles de bains</caractéristique>
            </caractéristiques>
        </appartement>
    </appartements>

</immobilier>
```

Document DTD :

```
<!ELEMENT immobilier (agences, propriétaires, propriétés, appartements)>

<!ELEMENT agences (agence*)>
<!ELEMENT propriétaires (propriétaire*)>
<!ELEMENT propriétés (propriété*)>
<!ELEMENT appartements (appartement*)>

<!ELEMENT agence (nom, email, téléphone)>

<!ELEMENT propriétaire (nom)>

<!ELEMENT propriété (nom, description, nombreAppartements,
caractéristiques)>
<!ATTLIST propriété
        propriétaire CDATA #REQUIRED>

<!ELEMENT appartement (nom, information, caractéristiques)>
<!ATTLIST appartement
        propriété CDATA #REQUIRED>

<!ELEMENT nom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT téléphone (#PCDATA)>
<!ELEMENT nombreAppartements (#PCDATA)>

<!ELEMENT caractéristiques (caractéristique*)>
<!ELEMENT caractéristique (#PCDATA)>

<!ELEMENT description (#PCDATA | refAppartement.)*>
<!ELEMENT refAppartement. (#PCDATA)>
<!ATTLIST refAppartement.
```

```
        nomAppartement CDATA #REQUIRED>

<!ELEMENT information ANY>
```

Solution exercice 6 :

Les Schemas XML permettent de spécifier ce que doit contenir un document XML : balises, structuration, attributs et types. Les schémas sont plus complets et plus précis par rapport aux DTD, avec une notion de types de données très forte, mais ils sont peu lisibles et nettement plus complexes.

Pour vérifier la validation, il faut Ouvrir les deux fichiers dans XML Copy Editor, puis appuyer sur F5 dans immobilier.xml.

Document XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<immobilier
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Immobilier_xsd.xsd">
    <agences></agences>
    <propriétaires></propriétaires>
    <propriétés></propriétés>
    <appartements></appartements>
</immobilier>
```

Document XSD :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="immobilier">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="agences"/>
                                <xs:element name="propriétaires"/>
                                <xs:element name="propriétés"/>
                                <xs:element name="appartements"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Solution exercice 7 :

Document XSD :

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="immobilier">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="agences">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="agence" type="TAgence" minOccurs="0" maxOccurs="unbounded"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="propriétaires">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="propriétaire" type="TPropriétaire" minOccurs="0" maxOccurs="unbounded"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
```

```
                                <xs:element name="propriétés">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="propriété" type="TPropriété" minOccurs="0" maxOccurs="unbounded"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="appartements">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="appartement" type="TAppartement" minOccurs="0" maxOccurs="unbounded"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:complexType name="TAgence">
                <xs:sequence>
                        <xs:element name="nom" type="xs:string"/>
                        <xs:element name="email" type="xs:string"/>
                        <xs:element name="téléphone" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="TPropriétaire">
                <xs:sequence>
                        <xs:element name="nom" type="xs:string"/>
                </xs:sequence>
                <xs:attribute name="idPropriétaire" type="xs:ID" use="required"/>
        </xs:complexType>

        <xs:complexType name="TPropriété">
                <xs:sequence>
                        <xs:element name="nom" type="xs:string"/>
                        <xs:element name="nombreAppartements">
                                <xs:simpleType>
                                        <xs:restriction
base="xs:positiveInteger">
                                                <xs:maxInclusive value="1500"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:element>
                </xs:sequence>
                <xs:attribute name="idPropriété" type="xs:ID" use="required"/>
                <xs:attribute name="refPropriétaire" type="xs:IDREF"/>
        </xs:complexType>
        <xs:complexType name="TAppartement">
                <xs:sequence>
                        <xs:element name="nom" type="TNomAppartement"/>
                        <xs:element name="description" type="xs:string"/>
                        <xs:element name="tarif" type="xs:nonNegativeInteger"/>
                </xs:sequence>
                <xs:attribute name="idAppartement" type="TAppartementID"
use="required"/>
                <xs:attribute name="refPropriété" type="xs:IDREF" use="required"/>
                <xs:attribute name="niveauxComfort"
type="TNiveauxComfortAppartement" use="required"/>
                <xs:attribute name="dateAffichage" type="xs:date" use="optional"/>
        </xs:complexType>
        <xs:simpleType name="TAppartementID">
        <xs:restriction base="xs:ID">
            <xs:pattern value="[aA][0-9]{1,5}"/>
        </xs:restriction>
    </xs:simpleType>
        <xs:simpleType name="TNomAppartement">
```

```
        <xs:restriction base="xs:string">
            <xs:minLength value="5"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>
        <xs:simpleType name="TNiveauxComfortAppartement">
        <xs:restriction base="xs:string">
            <xs:enumeration value="A"/>
            <xs:enumeration value="B"/>
            <xs:enumeration value="C"/>
            <xs:enumeration value="D"/>
            <xs:enumeration value="E"/>
            <xs:enumeration value="F"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

4 Galaxie XML

XML a aussi sa galaxie d'outils : langages, outils associés (xPath, XQuery, XML Schemas, Relax-NG, XSLT, ...), dialectes ou instances (RSS, SVG, XUL, MathML, WSDL, SOAP, OpenStreetMap, SAML, OpenDoument, TEI, DocBook, epub...)

4.1 Les chemins: XPATH

4.1.1 Principes

XPath est un langage non-XML (comme les DTD), qui permet d'identifier et localiser des parties d'un document XML, et de sélectionner des nœuds et des ensembles de nœuds (sous-arbres) dans une arborescence XML résolue. Son principe est assez simple : il décrit des chemins à prendre pour aller de la racine (ou d'un nœud «local») vers le nœud recherché en suivant les filiations. Ce langage simple est utilisé de manière indépendante pour faire des recherches dans un document XML ou comme outil dans d'autres dialectes XML (XPointer, XLink, XSLT, XQuery, XForms, etc.).

Les éléments de base de Xpath sont : les axes (qui expliquent dans quel sens on cherche), les nœuds qui restreignent la recherche sur un type d'élément, et les prédicats (optionnels) qui permettent d'exprimer des restrictions sur les nœuds.

4.1.2 Les axes

Chaque étape XPath est composée de deux composants obligatoires : **axe** et filtre (**filter)**, et d'un élément facultatif : le prédicat (**predicate)**. Pour spécifier la direction à suivre pour atteindre les nœuds suivants à partir d'un nœud de contexte. Ça peut être par hiérarchie directe ou indirecte entre les nœuds ou par position relative des nœuds entre eux.

Un axe a un type qui peut être un attribut, un élément ou un espace de nommage (namespace). Les axes XPath sont :

- **attribute**: (Notation abrégée : @) Sélectionne tous les attributs du nœud actuel.
- **child**: Sélectionne tous les enfants du nœud actuel
- descendant: Sélectionne tous les descendants (enfants, petits-enfants, etc.) du nœud courant
- **descendant-or-self**: Sélectionne tous les descendants (enfants, petits-enfants, etc.) du nœud actuel et le nœud actuel lui-même
- **following**: Sélectionne tout dans le document après la balise de fermeture du nœud actuel
- **following-sibling**: Sélectionne tous les frères après le nœud actuel
- **namespace**: Sélectionne tous les nœuds d'espace de nommage du nœud actuel
- **parent**: (Notation abrégée : **..**) Sélectionne le parent du nœud actuel
- **preceding**: Sélectionne tous les nœuds qui apparaissent avant le nœud actuel dans le document, à l'exception des ancêtres, des nœuds d'attribut et des nœuds d'espace de nommage
- **preceding-sibling**: Sélectionne tous les frères et sœurs avant le nœud actuel
- **self** : (Notation abrégée : **.**) Sélectionne le nœud actuel.

Considérons le document XML bookstore.xml suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

Exemples d'axe XPath :

- **child::book** : sélectionne tous les nœuds de book qui sont des enfants du nœud actuel
- **attribute::lang** : sélectionne l'attribut lang du nœud actuel
- **child::*** : Sélectionne tous les éléments enfants du nœud actuel
- **attribute::*** : sélectionne tous les attributs du nœud actuel
- **child::text()** : sélectionne tous les nœuds de texte enfants du nœud actuel
- **child::node()** : sélectionne tous les enfants du nœud actuel
- **descendant::book** : sélectionne tous les livres (nœud book) descendants du nœud actuel
- **ancestor::book** : sélectionne tous les ancêtres de book du nœud actuel
- **ancestor-or-self::book** : sélectionne tous les ancêtres book du nœud courant - et le courant aussi s'il s'agit d'un nœud book
- **child::*/child::price** : Sélectionne tous les petits-enfants de prix (price) du nœud actuel

4.1.3 Les filtres

Dans XPath, il existe deux types de filtres : les filtres de **nom** et les filtres de **type**.

Les **filtres de nom** filtrent les éléments qui ont un nom : élément, instructions de traitement, les attributs et nom des nœuds.

Exemples :

- /A/B/D sélectionne les éléments D, enfants des éléments B eux-mêmes enfants de l'élément A.
- /A/* sélectionne chaque élément enfant de l'élément A : [B,B,C].
- /A/B/D sélectionne les trois noeuds D [D,D,D].

Les **filtres de type** permettent de filtrer les nœuds selon leur type :

- **text()** : type de nœud Texte. Par exemple : /book/title//text()
- **comment()** : type de nœud Commentaire. Par exemple : /comment()
- **processing-instruction()** : type de nœud Instruction de traitement. Par exemple : processing-instruction('XSL')
- **node()** : tous les types de nœuds.

Exemples :

- /A/B//text(): sélectionne les nœuds de texte
- //@att1 : sélectionne les attributs "att1" des deux nœuds.
- //* : sélectionne tous les attributs.

4.1.4 Les prédicats

Les prédicats XPath sont des expressions booléennes construites à partir d'expressions de chemin et de fonctions prédéfinies. Ils sont utilisés pour trouver un nœud spécifique ou un nœud qui contient une valeur spécifique. Les prédicats sont toujours intégrés entre crochets.

Ci-dessous, certains exemples d'expressions de chemin avec des prédicats et le résultat des expressions :

- /bookstore/book[1] : sélectionne le premier élément book qui est l'enfant de l'élément bookstore.
- /bookstore/book[last()] : sélectionne le dernier élément book qui est l'enfant de l'élément bookstore.
- /bookstore/book[position()<3] : sélectionne les deux premiers éléments book qui sont des enfants de l'élément bookstore.
- //title[@lang] : sélectionne tous les éléments de titre qui ont un attribut nommé lang.
- //title[@lang='en'] : sélectionne tous les éléments de titre qui ont un attribut "lang" avec une valeur de "en".
- /bookstore/book[price>35.00] : sélectionne tous les éléments book de l'élément bookstore qui ont un élément price avec une valeur supérieure à 35.00.
- /bookstore/book[price>35.00]/title : sélectionne tous les éléments title des éléments book de l'élément bookstore qui ont un élément price avec une valeur supérieure à 35.00.

4.2 Les feuilles de style et traitements XSL

Dans les documents HTML, les balises sont prédéfinies, mais dans les documents XML, les balises ne sont pas prédéfinies. Le W3C a développé **XSL (signifie EXtensible Stylesheet Language)** pour comprendre et styliser un document XML, qui peut agir comme langage de feuille de style basé sur XML, tout comme CSS (Cascading Style Sheets) est un langage de style pour HTML. Un document XSL spécifie comment un navigateur doit restituer un document XML.

Principales parties du document XSL :

- **XSLT** : C'est un langage pour transformer des documents XML en divers autres types de documents (comme transformer XML en HTML).
- **XPath** : C'est un langage de navigation dans les documents XML.
- **XQuery** : C'est un langage pour interroger des documents XML.
- **XSL-FO** : C'est un langage de formatage de documents XML.

Comment fonctionne XSLT (Modèle de traitement XSL) :

La feuille de style XSLT est écrite au format XML. Le processeur XSLT prend la feuille de style XSLT et applique les règles de transformation sur le document XML cible, puis il génère un document formaté au format XML, HTML ou texte. À la fin, il est utilisé par le formateur XSLT pour générer la sortie réelle et affiché sur l'utilisateur final.

La structure de base d'une feuille de style XSL est le modèle (Template). Un modèle s'applique dans le nœud de contexte d'un arbre et produit un fragment de résultat de la transformation. Le résultat global (arbre) est construit par une hiérarchie de modèles. Ce résultat peut contenir les éléments de l'arborescence des sources XML (éléments non filtrés et éléments organisés dans un ordre différent), et de nouveaux éléments créés par les modèles.

Les modèles (Templates) comportent deux parties : Un modèle à rechercher dans l'arbre source (dépend du contexte structurel de l'arbre source et il est exprimé en XPath) et un modèle utilisé pour former l'arbre de résultat.

Une feuille de style XML a la structure suivante :

Figure 16 Structure d'une feuille de style XML

Les modèles sont tous des enfants directs du nœud **xsl:stylesheet**. Ils ne peuvent pas être imbriqués. Leur ordre n'a aucune importance.

Le **processus de transformation** XSL passe par les étapes suivantes : Il commence par l'analyse des nœuds du document source (séquentiellement), c'est à dire, la recherche d'un modèle correspondant défini dans la feuille de style et l'application des instructions contenues dans ce modèle et le remplacement du nœud du document source courant par le résultat du modèle. Dès qu'un modèle est trouvé, il est appliqué. Si plusieurs modèles trouvés : Si les templates concurrents proviennent de 2 feuilles de style (import external sheet), donc la priorité d'importation peut être explicitement définie à l'aide de l'élément <xsl:import>. Si aucune priorité explicite n'est définie, la feuille d'importation a une priorité plus élevée que celle importée. Sinon si les modèles concurrents proviennent de la même feuille, dans ce cas la priorité explicite la plus élevée ou la plus précise est choisie (attribut de priorité du modèle, modèle de sélectivité : règle la plus précise = priorité la plus

élevée). Si aucun modèle correspondant n'est trouvé, le modèle XSL par défaut est appliqué en fonction du type de nœud.

Quelques instructions de base XSL :

À l'intérieur d'un xsl:template :

- <xsl:value-of select = "nom de l'élément" /> : copier le contenu sans balises.
- <xsl:copy-of select = "nom de l'élément" /> : copier le contenu avec des balises.
- <xsl:apply-templates select = "XPATH vers les éléments à traiter" /> : appeler le traitement d'autres nœuds.

Autres instructions XSL :

- xsl:sort : structure de tri
- xsl:if, xsl:choose, xsl:otherwise, ... : traitement conditionnel

Si nous prenons l'exemple suivant :

```
<table>
    <description>4th floor staff</description>
    <person>
      <name>Bond</name>
      <office>U1</office>
    </person>
    <person>
      <name>Lupine</name>
      <office>U2</office>
    </person>
    <person>
      <name>Templar</name>
      <office>U3</office>
    </person>
</table>
```

Le modèle XSL suivant :

```
<xsl:template match ="person">
        <li>
               <xsl:value-of select ="name"/> -
               <xsl:value-of select ="office"/>
        </li>
</xsl:template>
```

Produira la liste :

```
<li>Bond - U1</li>
<li>Lupine - U2</li>
<li>Templar - U3</li>
```

A noter que les balises "<li>" et le caractère "-" sont recopiés à l'identique sur le résultat, tandis que les balises <xsl:value-of …/> sont exécutées car elles sont interprétées comme des instructions XSL.

4.3 Applications XML : RDF, SVG, ...

Il existe de nombreux dialectes basés sur XML créés par des communautés de différents domaines. Alors que beaucoup ne sont utilisés que par de petites communautés, certains d'entre eux sont devenus des standards de facto et sont supportés nativement par des navigateurs Web tels que RDF, SVG, MathML, etc…

4.3.1 RDF

RDF (Resource Description Framework) est un langage de description des données Meta pour les documents. Les données Meta peuvent jouer différents rôles: par exemple représenter des saisies pour les robots de moteurs de recherche ou bien livrer les données nécessaires à l'enregistrement dans le système de catalogue d'une certaine bibliothèque électronique. Un exemple simple :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.0/">
  <rdf:Description rdf:about="http://localhost/doc"
      dc:creator="Frédéric Figaret"
      dc:title="Pêche d'aujourd'hui"
      dc:description="Une documentation sur les méthodes de pêche moderne"
      dc:date="2001-03-30" />
</rdf:RDF>
```

L'exemple montre une instance d'espace de nommage RDF, introduit par <rdf:RDF... et clôturé par </rdf:RDF>. Pour l'instance d'espace de nommage deux sources sont toutefois mentionnées: une fois la définition de syntaxe de données Meta d'après le schéma RDF du consortium W3, et une fois celle d'après le schéma RDF de Dublin Core (dc). Il est ainsi possible d'utiliser à la suite le schéma du W3 avec le repère rdf: et le schéma Dublin-Core avec le repère dc: .

4.3.2 SVG

SVG (Scalable Vector Graphics) est un langage de marquage défini avec XML pour la description de graphiques vectoriels en 2 dimensions. Grâce à ce langage des données graphiques peuvent être décrites telles qu'elles peuvent être créées par des programmes comme Corel Draw par exemple. Un exemple simple :

```
<svg width="4in" height="3in">
<g>
  <rect x="50", y="80", width="200", height="100" style="fill: #FFFFCC"/>
</g>
</svg>
```

L'exemple montre comment en SVG un "canevas" dans une surface d'image est créé à l'aide d'un élément central svg. À l'aide de l'élément de groupage g les éléments peuvent être reliés en unités complexes. Dans l'exemple, un simple rectangle a été défini dans le seul groupe défini. Pour cela, l'élément rect a été placé.

4.3.3 MathML

MathML (Mathematical Markup Language) a la mission de représenter des contenus techniques et scientifiques qui ont besoin pour l'être du langage de formulation mathématique. Un exemple simple :

```
<msup>
  <mfenced>
    <mrow>
       <mi>a</mi>
       <mo>+</mo>
       <mi>b</mi>
     </mrow>
   </mfenced>
</mn>2</mn>
</msup>
```

L'exemple donne comme sortie: $(a+b)^2$. L'avantage de la description en MarhML est que chacun des différents éléments d'une expression reçoit son propre marquage logique. Ce n'est qu'ainsi qu'il est possible de décrire des expressions complexes à souhait.

Avec <mfenced>...</mfenced> les parenthèses sont définies, avec <mrow>...</mrow> une expression notée horizontalement et formant un tout, avec <mi>...</mi> un identificateur (Identifier), avec <mo>...</mo> un opérateur, avec <mn>...</mn> une valeur numérique et avec <msup>...</msup> ce nombre est défini comme exposant (Superscript).

4.4 Traitement : DOM et SAX

Les documents XML peuvent être manipulés : analysés à partir de fichiers existants, créés à partir de zéro, modifiés via des langages de programmation. Les deux principales interfaces pour y parvenir sont **DOM** et **SAX**.

4.4.1 DOM

Le **DOM (Document Object Model)** est une recommandation officielle du W3C. Il conserve l'intégralité du document XML en mémoire et fournit des interfaces pour l'analyser, y accéder, le créer, le modifier et le sérialiser. Par exemple, les navigateurs Web fournissent une interface DOM en javascript pour gérer les pages Web chargées (il s'agit essentiellement de documents XML).

Un analyseur DOM doit être utilisé lorsque la structure d'un document est bien connue ou des parties d'un document XML doivent être déplacées (trier certains éléments, par exemple) ou il faut utiliser plusieurs fois les informations d'un document XML.

Lorsqu'un document XML est analysé avec un analyseur DOM, une **arborescence** qui contient tous les éléments de document est récupérée. Le DOM fournit une variété de fonctions pour examiner le contenu et la structure du document.

Interfaces DOM :

Le DOM définit plusieurs interfaces Java. Voici les interfaces les plus courantes :

- **Node** − Le type de données de base du DOM.
- **Element** − La grande majorité des objets utilisés sont des éléments.
- **Attr** − Représente un attribut d'un élément.
- **Text** − Le contenu réel d'un élément ou d'un attribut.
- **Document** − Représente l'ensemble du document XML. Un objet Document est souvent appelé un arbre DOM.

Méthodes DOM courantes :

Il y a plusieurs méthodes utilisées souvent :

- **Document.getDocumentElement()** − Renvoie l'élément racine du document.
- **Node.getFirstChild()** − Renvoie le premier enfant d'un nœud donné.
- **Node.getLastChild()** − Retourne le dernier enfant d'un nœud donné.
- **Node.getNextSibling()** − Ces méthodes renvoient le frère suivant d'un nœud donné.
- **Node.getPreviousSibling()** − Ces méthodes renvoient le frère précédent d'un nœud donné.
- **Node.getAttribute(attrName)** − Pour un nœud donné, il renvoie l'attribut avec le nom demandé.

4.4.2 SAX

SAX est l'acronyme de **Simple API for XML**. Cette API a été développée par David Megginson (http://www.saxproject.org/). **L'API SAX** permet également l'analyse de documents XML. Ce type d'analyseur utilise des **événements** pour piloter le traitement d'un fichier XML et permet aux méthodes de rappel implémentées par un objet appelé handler en anglais de gérer des fragments de document XML, tels que des éléments, des attributs,… sans avoir le document entier en mémoire. Par conséquent, SAX a une gestion de la mémoire efficace, contrairement à DOM.

Un analyseur syntaxique SAX doit être utilisé quand le document XML peut être traité de manière linéaire, de haut en bas ou il n'est pas profondément imbriqué ou il s'agit d'un très gros document XML dont l'arbre DOM consommerait trop de mémoire ou le problème à résoudre ne concerne qu'une partie du document XML.

L'inconvénient de SAX est que nous n'avons pas d'accès aléatoire à un document XML puisqu'il est traité de manière prospective. En plus, si nous devons garder la trace des données que l'analyseur syntaxique a vues ou modifier l'ordre des éléments, nous devons écrire le code et stocker les données vous-même.

Interface ContentHandler :

Cette interface spécifie les méthodes de rappel que l'analyseur SAX utilise pour notifier à un programme d'application les composants du document XML qu'il a vus.

- **void startDocument()** − Appelé au début d'un document.
- **void endDocument()** − Appelé à la fin d'un document.
- **void startElement(String uri, String localName, String qName, Attributes atts)** − Appelé au début d'un élément.
- **void endElement(String uri, String localName,String qName)** − Appelé à la fin d'un élément.

- **void characters(char[] ch, int start, int length)** − Appelé lorsque des données de caractères sont rencontrées.
- **void ignorableWhitespace(char[] ch, int start, int length)** − Appelé lorsqu'une DTD est présente et qu'un espace blanc ignorable est rencontré.
- **void processingInstruction(String target, String data)** − Appelé lorsqu'une instruction de traitement est reconnue.
- **void setDocumentLocator(Locator locator))** − Fournit un localisateur qui peut être utilisé pour identifier des positions dans le document.
- **void skippedEntity(String name)** − Appelé lorsqu'une entité non résolue est rencontrée.
- **void startPrefixMapping(String prefix, String uri)** − Appelé lorsqu'un nouveau mappage d'espace de nommage est défini.
- **void endPrefixMapping(String prefix)** − Appelé lorsqu'une définition d'espace de nommage termine sa portée.

Interface Attributs :

Cette interface spécifie les méthodes de traitement des attributs liés à un élément.

- **int getLength()** − Renvoie le nombre d'attributs.
- **String getQName(int index)**
- **String getValue(int index)**
- **String getValue(String qname)**

4.5 Les liens: XLINK

XLink est une recommandation du W3C, utilisé pour créer des hyperliens dans des documents XML. Tout élément dans un document XML peut se comporter comme un lien. Avec XLink, les liens peuvent être définis en dehors des fichiers liés.

En HTML, l'élément <a> définit un lien hypertexte. Cependant, ce n'est pas ainsi que cela fonctionne en XML. Dans les documents XML, nous pouvons utiliser les noms d'éléments de notre choix. Il est donc impossible pour les navigateurs de prédire quels éléments de lien seront appelés dans les documents XML.

Voici un exemple simple de la façon d'utiliser XLink pour créer des liens dans un document XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="http://www.w3ii.com">Visit
w3ii</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit
W3C</homepage>
</homepages>
```

Pour avoir accès aux fonctionnalités XLink, nous devons déclarer l'espace de noms XLink.

- L'espace de noms XLink est: "http://www.w3.org/1999/xlink".
- Le xlink: Type et xlink:href attributs dans le <homepage> éléments proviennent de l'espace de noms XLink.
- Le xlink: type = "simple" crée simple "HTML-like" lien (signifie "cliquez ici pour y aller").
- Le xlink:href attribut spécifie l'URL à lier.

4.6 Les pointeurs: XPOINTER

Pour permettre aux liens de pointer vers des parties spécifiques d'un document XML, le XPointer peut être utilisé. Il s'agit d'une recommandation du W3C et utilise des expressions XPath pour naviguer dans le document XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<flowers>
<flower breed="Rose" id="Rose">
  <picture url="images/rose.gif" />
  <description>A rose is.......</description>
```

```
  <colors>Pink-, yellow-, red- or....</colors>
</flower>
<flower breed="Lily" id="Lily">
  <picture url="images/lily.gif" />
  <description> Lilies are a group of ..... </description>
  <colors>White, pink, red, yellow, orange, and ....</colors>
</flower>
</flowers>
```

Dans l'exemple ci-dessus, XPointer est utilisé en conjonction avec XLink pour pointer vers une partie spécifique d'un autre document. Pour cela, nous examinons le document XML cible, c'est-à-dire le document vers lequel nous établissons un lien. Ici, le document XML utilise des attributs id sur chaque élément. Ainsi, en utilisant XPointer, nous pouvons créer un lien vers des parties spécifiques du document, au lieu de créer un lien vers le document entier (comme avec XLink). Nous devons ajouter un signe numérique (#) et une expression XPointer après l'URL dans l'attribut xlink:href pour créer un lien vers une partie spécifique d'une page : xlink:href="images/flowers.xml#xpointer(id(Rose))". L'expression ci-dessus fait donc référence à l'élément du document cible dont la valeur id est "Rose".

4.7 Exercices

Exercice 1 :

À partir du fichier XML Immobilier.xml, ci-dessous, exprimez les **requêtes XPath** suivantes :

1. Sélectionnez les codes postaux de tous les propriétaires de l'Algérie.
    A. Sélectionnez tous les propriétaires (avec leurs sous arbres entiers).
    B. Sélectionnez les noms de toutes les agences (uniquement le contenu textuel).
    C. Sélectionnez les pays propriétaires sans valeurs en double.
    D. Sélectionnez le nom du propriétaire avec l'id o2.
2. Sélectionnez les codes postaux de tous les propriétaires de l'Algérie.
3. Sélectionnez les valeurs tarifaires des appartements avec un niveau de confort C parmi toutes les propriétés autres que p2.
4. Sélectionnez la troisième caractéristique de la première propriété.
5. Sélectionnez des noms de propriétés sans propriétaires et présentant au moins deux caractéristiques.
6. Sélectionnez les propriétaires sans aucune propriété.
7. Sélectionnez les appartements avec des tarifs inférieurs à la moyenne.
8. Sélectionnez les noms des propriétaires qui possèdent un appartement avec une salle de bain.
9. Sélectionnez la somme globale des tarifs des appartements de niveau de confort C ou D parmi les propriétés du propriétaire o1.
10. Sélectionnez les noms du tout premier et du tout dernier appartement de tout le document, y compris les appartements imbriqués récursivement (c'est-à-dire par rapport à l'ordre du document).
11. Étant donné l'identifiant d'un appartement particulier, sélectionnez l'appartement de niveau supérieur qui le contient (récursivement). Par exemple. sélectionnez appartement a4 lorsque a4a1, a4a ou a4 est interrogé.
12. Sélectionnez les pays distincts des propriétaires d'appartements avec un niveau de confort A ou un loyer supérieur à 20000.

Document Immobilier.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE immobilier SYSTEM "Immobilier_dtd.dtd">
<immobilier language="FR">

    <!--Liste des agences -->
    <agences>
        <agence>
            <nom>Meilleures propriétés à Oran</nom>
            <email>info@best-oran.com</email>
            <téléphone>+213 555555555</téléphone>
            <adresse>
                <rue>934 5eme avenue</rue>
                <codePostal>31008</codePostal>
                <province>Oran</province>
                <pays>Algérie</pays>
            </adresse>
            <employé>
                                <prénom>Mohamed</prénom>
                                <nomFamille>Abdelhadi</nomFamille>
            </employé>
            <employé>
                                <prénom>Aymen</prénom>
                                <nomFamille>Amir</nomFamille>
            </employé>
            <employé>
                                <prénom>Amel</prénom>
                                <nomFamille>Abdelhadi</nomFamille>
            </employé>
        </agence>
    </agences>

    <!-- Liste des propriétaires -->
```

```
    <propriétaires>
        <propriétaire idPropriétaire="o1">
            <nom>Immobilier à Oran</nom>
            <adresse>
                <rue>984 6th avenue</rue>
                <codePostal>31009</codePostal>
                <province>Oran</province>
                <pays>Algérie</pays>
            </adresse>
        </propriétaire>
        <propriétaire idPropriétaire="o2">
            <nom>Premier immobilier</nom>
            <adresse>
                <rue>Moscou 234</rue>
                <codePostal>Paris 75005</codePostal>
                <province>Île-de-France</province>
                <pays>France</pays>
            </adresse>
        </propriétaire>
        <propriétaire idPropriétaire="o3">
            <nom>Ibrahim Naim</nom>
            <adresse>
                <rue>La Macta 25</rue>
                <codePostal>22001</codePostal>
                <province>Sidi Bel Abbes</province>
                <pays>Algérie</pays>
            </adresse>
        </propriétaire>
        <propriétaire idPropriétaire="o2a">
            <nom>Premier immobilier</nom>
            <adresse>
                <rue>Moscou 235</rue>
                <codePostal>Paris 75005</codePostal>
                <province>Île-de-France</province>
                <pays>France</pays>
            </adresse>
        </propriétaire>
    </propriétaires>

    <!-- Liste des propriétés -->
    <propriétés>

        <propriété idPropriété="p1" propriétaire="o1">
            <nom>Gratte-ciel de luxe</nom>
            <adresse>
                <rue>14 26e avenue</rue>
                <province>Oran</province>
                <codePostal>31003</codePostal>
            </adresse>
            <nombreAppartements>154</nombreAppartements>
            <caractéristiques>
                <caractéristique>ascenseur</caractéristique>
                <caractéristique>parking</caractéristique>
                <caractéristique>securité</caractéristique>
                <caractéristique>vue sur la mer</caractéristique>
                <caractéristique>tramway</caractéristique>
            </caractéristiques>
            <employé>
                                    <prénom>Mohamed</prénom>
                                    <nomFamille>Abdelhadi</nomFamille>
            </employé>
        </propriété>

        <propriété idPropriété="p3">
            <nom>Vieux gratte-ciel</nom>
            <adresse>
                <rue>16 26eme avenue</rue>
                <province>Oran</province>
```

```
                <codePostal>31004</codePostal>
            </adresse>
            <nombreAppartements>1</nombreAppartements>
            <caractéristiques>
                <caractéristique>vue sur la rue</caractéristique>
                <caractéristique>tramway</caractéristique>
            </caractéristiques>
        </propriété>

    </propriétés>

    <!-- Liste des appartements-->
    <appartements>

        <appartement idAppartement="a1" propriété="p1" comfort="F">
            <nom>A1001</nom>
            <tarif>1234</tarif>
            <information>Un petit appartement sans fenêtres. Punaises de lit
incluses</information>
            <enregistrement>
                <enregistrement>
                    <date>2015-07-11</date><texte>Punaises de lit observées</texte>
                </enregistrement>
                <enregistrement>
                    <date>2015-07-12</date><texte>Nettoyage effectué</texte>
                </enregistrement>
            </enregistrement>
            <enregistrement>
                <date><?php echo date("Y-m-d"); ?></date>
                <texte>Punaises de lit observées une fois de plus</texte>
            </enregistrement>
        </appartement>

        <appartement idAppartement="a2" propriété="p1" comfort="A">
            <nom>A2601</nom>
            <tarif>46001</tarif>
            <information>
                <description>Grande suite avec jardin sur le toit.</description>
                vue sur la mer, 200 mètres.
                <nom>Une suite splendide</nom>
                <description>46001</description>
                &lt;span class=&quot;btn&quot;&gt;Button&lt;/span&gt;
                <![CDATA[
                    <span class="btn">Button</span>
                ]]>
            </information>
            <caractéristiques>
                <caractéristique>propre ascenseur</caractéristique>
                <caractéristique>jardin</caractéristique>
                <caractéristique>bain</caractéristique>
                <caractéristique>2 salles de bains</caractéristique>
            </caractéristiques>
        </appartement>

        <appartement idAppartement="a3" propriété="p1" comfort="B">
            <nom>A654</nom>
            <tarif>12345</tarif>
            <information>
                <description>Belle vue, pas d'insectes.</description>
            </information>
            <caractéristiques>
                <caractéristique>tapis</caractéristique>
                <caractéristique>cuisine</caractéristique>
                <caractéristique>2 salles de bains</caractéristique>
            </caractéristiques>
        </appartement>

        <appartement idAppartement="a4" propriété="p1" comfort="C">
```

```
            <nom>A0001</nom>
            <tarif>456</tarif>
            <information>
                <description>Appartement au sous-sol, essentiellement destiné à
être utilisé par des groupes ou quelqu'un de bruyant.</description>
            </information>
            <caractéristiques>
                <caractéristique>insonoriser</caractéristique>
                <caractéristique>sombre</caractéristique>
                <caractéristique>petit</caractéristique>
            </caractéristiques>
            <appartement idAppartement="a4a" propriété="p1" comfort="C">
                <nom>A0001-A</nom>
                <tarif>45</tarif>
                <information>
                    <description>Salle A. Possède une cabine d'étude
séparée.</description>
                </information>
                <caractéristiques>
                    <caractéristique>insonoriser</caractéristique>
                    <caractéristique>sombre</caractéristique>
                    <caractéristique>petit</caractéristique>
                </caractéristiques>
                <appartement idAppartement="a4a1" propriété="p1" comfort="C">
                    <nom>A0001-A1</nom>
                    <tarif>26</tarif>
                    <information>
                        <description>Cabine d'étude.</description>
                    </information>
                    <caractéristiques>
                        <caractéristique>insonoriser</caractéristique>
                        <caractéristique>sombre</caractéristique>
                        <caractéristique>petit</caractéristique>
                        <caractéristique>chambre</caractéristique>
                    </caractéristiques>
                </appartement>
            </appartement>
            <appartement idAppartement="a4b" propriété="p1" comfort="C">
                <nom>A0001-B</nom>
                <tarif>46</tarif>
                <information>
                    <description>Pas de fenêtres.</description>
                </information>
                <caractéristiques>
                    <caractéristique>insonoriser</caractéristique>
                    <caractéristique>sombre</caractéristique>
                    <caractéristique>petit</caractéristique>
                    <caractéristique>pas de lumière du jour.</caractéristique>
                </caractéristiques>
            </appartement>
        </appartement>

    </appartements>

</immobilier>
```

Exercice 2 :

1. Créez un script **XSLT** vide pour le document XML fourni (Immobilier.xml de l'exercice précédent). Utilisez HTML comme format de sortie.

Étendez le script XSLT précédent :

2. Générez la structure de base de la page Web de sortie.

```
<html>
    <head>
        <title>Rapport immobilier</title>
    </head>
```

```
        <body>
        …
        </body>
    </html>
```

3. Créez une table de propriétés et ajoutez-la dans la sortie.

```
    <h2>Tableaux de Propriétés </h2>
    <table>
    <tr>
    <th>ID propriété</th>
    <th>Nom propriété</th>
    </tr>
    …
    <tr><td>ID</td><td>Nom</td><td>Propriétaire</td></tr>
    …
</table>
```

4. Générer des lignes de ce tableau (une pour chaque propriété)
5. Créez un seul nouveau modèle sans nom.
    - utilisez xsl:apply-templates, n'utilisez pas xsl:value-of
6. Créez une liste ordonnée d'appartements.

```
    <h2>Liste des appartements</h2>
    <ol>
    <li id="idAppartement">idAppartement: Nom</li>
    …
    </ol>
```

    - Remplacez ID et Nom par les valeurs correspondantes.
7. Créez un modèle sans nom pour les éléments appartement.
8. Utilisez xsl:value-of et les constructeurs calculés uniquement
9. C'est à dire. xsl :element, xsl :attribute, xsl :text
10. Ajoutez une nouvelle colonne dans la table des propriétés.
11. Pour chaque propriété, générez une liste non ordonnée de références à ses appartements.

```
    …
    <td>
    <ul>
    <li><a href="#idAppartement">Nom</a></li>
    …
    </ul>
    </td>
    …
```

12. Utilisez des modèles sans nom avec des modes.
13. Intégrez le CSS suivant dans l'élément head.

```
    <style>
        th { background-color: #AAAAAA; }
        tr.even td { background-color: #DDDDDD; }
    </style>
```

14. Modifier le tableau des propriétés.
    - Ajoutez une nouvelle colonne avec les numéros d'ordre des documents des propriétés.
      `<tr><td>1</td>…</tr>`
      `<tr><td>2</td>…</tr>`
      …
    - Set the background color of all even rows.
      `<tr class="even">…</tr>`
    - Utilisez les instructions conditionnelles xsl:if
15. Modifier le tableau des propriétés.
    - Ajoutez une nouvelle colonne avec le nom de propriétaire correspondant à chaque propriété.

- Lorsque cette référence de propriétaire ou ce propriétaire ou leur nom est manquant, écrivez Inconnu à la place.

16. Créez et appelez un modèle nommé à cet effet.
    - Utilisez un paramètre pour spécifier l'identifiant du propriétaire de l'entrée.
    - Lorsqu'il est appelé sans ce paramètre, récupère néanmoins l'identifiant du propriétaire à partir des données de propriété données.
17. Ajoutez une nouvelle colonne avec une liste de caractéristiques de propriété séparées par des virgules dans le tableau des propriétés.

```
<tr>
…
<td> alarme, portier, vue sur la rue, métro </td>
</tr>
```

    - Classez cette liste par ordre alphabétique.
    - Imprimer toutes les caractéristiques appartenant à et uniquement à l'emplacement de la catégorie en bleu,
    - Imprimer toutes les autres caractéristiques de la catégorie sécurité en rouge, et
    - conserver la couleur par défaut sinon.
    - Utilisez l'instruction xsl:choose

Exercice 3 :

Donnez le **graphe RDF** décrivant les éléments suivants :

« Amine et Amel habitent dans une ville nommée Arzew située dans la province d'Oran ».

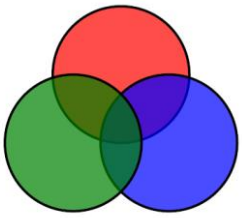
Exercice 4 :

Traduisez le graphe RDF de l'exercice précédent en **document RDF/XML**. Dans ce TP, il est conseillé de s'aider de http://www.w3.org/RDF/Validator/.

Exercice 5 :

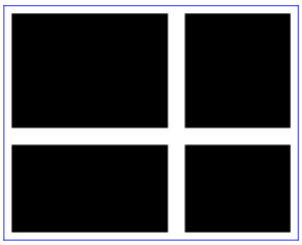
Le **SVG**, une recommandation du W3C, est "un langage pour décrire des graphiques vectoriels bidimensionnels et mixtes vecteur/ traceur en XML". En d'autres termes, il est possible de dessiner des images avec des éléments XML. Tous les navigateurs prennent en charge les fichiers SVG, à l'exception d'Internet Explorer, qui nécessite un plug-in.

Ecrire un SVG qui permet de dessiner trois cercles. Une fois rendue, cette image SVG ressemble à ceci :



Exercice 6 :

Ecrire un **SVG** qui permet de dessiner quatre rectangles entourés d'un autre rectangle tel que montré sur la figure suivante :



Exercice 7 :

Écrivez un programme en Java qui utilise **l'API DOM** et modifie le fichier XML initial Immobilier.xml en transformant les éléments de taux des appartements en attributs.

Exercice 8 :

Étant donné le document XML initial Immobilier.xml, faire un programme Java qui permet de supprimer les appartements dont le taux est >= 20000 en utilisant le **Parser DOM**.

Exercice 9 :

En utilisant le **Parser DOM**, écrire un programme Java qui permet de créer un nouvel enregistrement de propriétaire, à partir de document XML Immobilier.xml, avec toutes les données respectives :

Votre programme doit fonctionner même si…

- Propriétaires d'élément est vide.
- Les propriétaires d'éléments n'existent pas du tout.

Exercice 10 :

Ecrire un programme Java utilisant **l'API SAX** qui, à partir d'un fichier d'entrée Immobilier.xml, implémente la validation de documents XML selon les indications suivantes :

- Les identifiants doivent commencer par les lettres suivantes : propriétaires : o, agences : a, immeubles : p, appartements : a.
- Chaque propriété doit avoir au moins 2 et au plus 5 caractéristiques particulières lorsqu'une liste de caractéristiques est fournie.
- La valeur de l'élément nombreAppartements doit être non négative.
- Imprimer les messages d'erreur dans un nouvel élément erreurs à la fin du document XML de sortie.
- Pour chaque erreur, indiquez sa position (ligne, colonne) et sa description textuelle.

Exercice 11 :

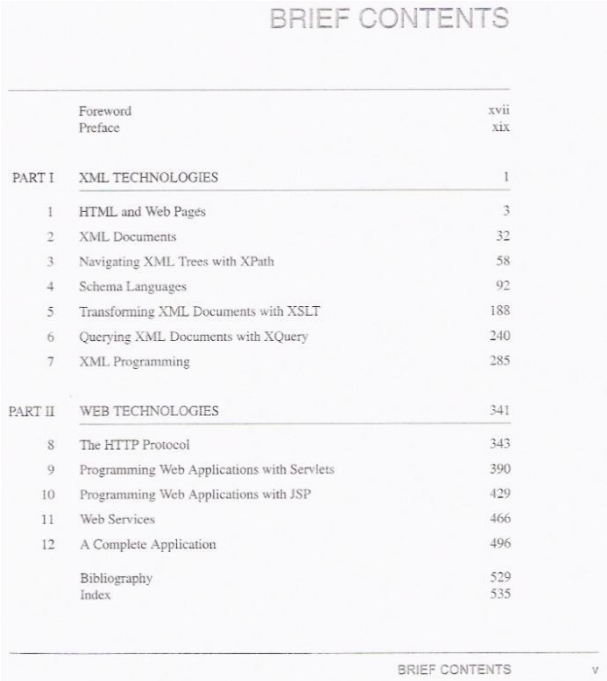
Soit le document xml suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<racesChiens>
<chien race="Rottweiler" id="Rottweiler">
  <image url="https://chien.com/rottweiler.gif" />
  <histoire>Les ancêtres du Rottweiler étaient probablement des chiens de bouviers
romains.....</histoire>
  <temperament>Confiant, audacieux, alerte et imposant, le Rottweiler est un choix
populaire pour sa capacité à protéger.... </temperament>
</chien>
<chien race="FCRetriever" id="FCRetriever">
  <image url="https://chien.com/fcretriever.gif" />
  <histoire>L'une des premières utilisations des chiens rapporteurs était d'aider
aider les pêcheurs à récupérer les poissons dans l'eau....</histoire>
  <temperament>Le Flat-coated retriever est un chien doux, exubérant, qui adore
jouer et récupérer....</temperament>
</chien>
</racesChiens>
```

En utilisant **XPointer** en conjonction avec **XLink**, proposer un document XML de telle sorte que le ce document contient des liens vers plus d'informations sur la race de chien.

Exercice 12 :

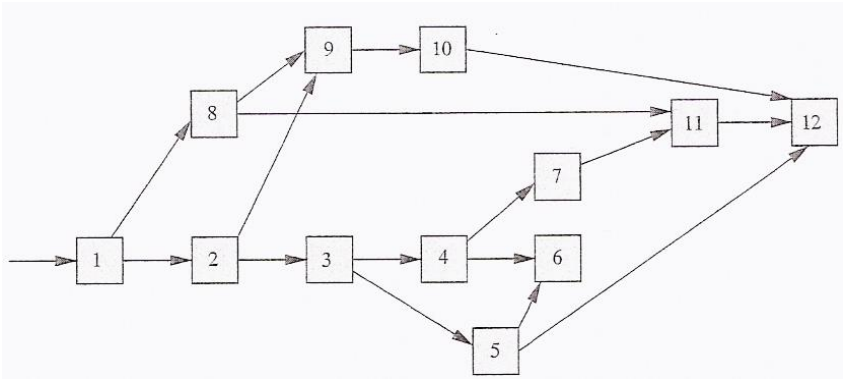
Soit la page du plan d'un livre :



BRIEF CONTENTS

BRIEF CONTENTS v

Dans ce livre, la lecture des différents chapitres suit le graphe suivant :



En utilisant **XLink**, créer un document XML qui permet de décrire ce graphe.

4.8 Corrigés des exercices

Solution exercice 1 :

Nous allons écrire des requêtes XPath sur différents fichiers. Pour l'exécution d'une requête, nous avons le choix, soit de travailler avec XML Copy Editor, soit avec des testeurs et évaluateur en ligne comme par exemple xpather.com. Avec XML Copy Editor : Chargez le fichier. La touche F9 permet de saisir une expression XPath. Les résultats sont affichés dans un nouvel onglet. Fermez cet onglet avant de saisir une nouvelle expression, sinon elle s'appliquera à cet onglet.

1A

```
/immobilier/propriétaires/ propriétaire
/child::immobilier/child::propriétaires/child::propriétaire
//propriétaire
//child::propriétaire
/descendant-or-self::node()/propriétaire
/descendant-or-self::node()/child::propriétaire
```

1B

```
/immobilier/agences/agence/name/text()
/immobilier/agences/agence/name/child::text()
//agence/nom/text()
```

```
Incorrecte:
//nom/text()
```

1C

```
distinct-values(/immobilier/propriétaires/propriétaire/adresse/pays)
distinct-values(/immobilier/propriétaires/propriétaire/adresse/pays/text())
distinct-values(//propriétaire/adresse/pays)
distinct-values(//propriétaire//pays)
Incorrecte:
distinct-values(//pays)
```

1D

```
/immobilier/propriétaires/propriétaire[@idPropriétaire = "o2"]/nom/text()
//propriétaire[@idPropriétaire = "o2"]/nom/text()
//propriétaire[@idPropriétaire = 'o2']/nom/text()
//propriétaire[./@idPropriétaire = "o2"]/nom/text()
//propriétaire[self::node()/@idPropriétaire = "o2"]/nom/text()
//propriétaire[attribute:: idPropriétaire = "o2"]/nom/text()
//propriétaire[./attribute:: idPropriétaire = "o2"]/nom/text()
```

2

```
//propriétaire/adresse[pays/text() = 'Algérie']/codePostal
//propriétaire/adresse[pays = 'Algérie']/codePostal
//propriétaire/adresse[child::pays = 'Algérie']/codePostal
//propriétaire/adresse[./pays = 'Algérie']/codePostal
//propriétaire/adresse[.//pays = 'Algérie']/codePostal

//propriétaire[adresse/pays = 'Algérie']/adresse/codePostal
//propriétaire[.//pays = 'Algérie']//codePostal

//propriétaire//pays[text() = 'Algérie']/../codePostal
//propriétaire//pays[. = 'Algérie']/preceding-sibling::codePostal
```

3

```
//appartement[@comfort = "C" and @propriété != "p2"]/tarif/text()
//appartement[(@comfort = "C") and (@propriété != "p2")]/tarif/text()
//appartement[@comfort = "C"][@propriété != "p2"]/tarif/text()
//tarif[parent::appartement/@comfort = "C" and
parent::appartement/@propriété != "p2"]/text()
//tarif[parent::appartement[@comfort = "C"] and
parent::appartement[@propriété != "p2"]]/text()
//tarif[../@comfort = "C" and ../@propriété != "p2"]/text()

//appartement[@comfort = "C"][not(@propriété = "p2")]/tarif/text()
```

4

```
/immobilier/properties/propriété[1]/caractéristiques/caractéristique[3]
/immobilier/properties/child::propriété[1]/caractéristiques/caractéristique
[3]
//propriété[1]/caractéristiques/caractéristique[3]
//propriété[1]//caractéristique[3]
//propriété[position() = 1]//caractéristique[position() = 3]
```

5

```
//propriété[not(@propriétaire) and count(caractéristiques/caractéristique)
>= 2]/nom
//propriété[not(@propriétaire) and count(.//caractéristique) >= 2]/nom
//propriété[not(@propriétaire) and count(self::node()//caractéristique) >=
2]/nom
```

```
//propriété[count(@propriétaire) = 0 and count(.//caractéristique) >=
2]/nom

//propriété[not(@propriétaire) and caractéristiques/caractéristique[2]]/nom

Incorrecte:
//propriété[@propriétaire = "" and count(caractéristiques/caractéristique)
>= 2]/nom
//propriété[@propriétaire = null and
count(caractéristiques/caractéristique) >= 2]/nom
//propriété[not(@propriétaire) and count(//caractéristique) >= 2]/nom
```

6

```
//propriétaire[not(@idPropriétaire = //propriété/@propriétaire)]

Incorrecte:
//propriétaire[@idPropriétaire != //propriété/@propriétaire]
//propriétaire[not(//propriété/[@propriétaire = @idPropriétaire])]
//propriétaire[count(@idPropriétaire = //propriété/@propriétaire) = 0]
//propriétaire[count(@idPropriétaire = //propriété/@propriétaire) = 1]
//propriétaire[count(@idPropriétaire != //propriété/@propriétaire) = 0]
//propriétaire[count(@idPropriétaire != //propriété/@propriétaire) = 1]
```

7

```
//appartement[tarif/text() < avg(//appartement/tarif/text())]
//appartement[tarif < avg(//appartement/tarif)]
//appartement[tarif < avg(//tarif)]
//appartement[tarif < avg(../appartement/tarif)]
//appartement[tarif < avg(..//tarif)]
//appartement[tarif < avg(data(//tarif))]
```

8

```
//propriétaire[
  @idPropriétaire = //propriété[
    @idPropriété = //appartement[caractéristiques/caractéristique =
'bain']/@propriété
  ]/@propriétaire
]/nom
```

9

```
sum(
  //appartement[@comfort = "C" or @comfort = "D"][
    @propriété = //propriété[@propriétaire = 'o1']/@idPropriété
  ]/tarif
)

… //appartement[@comfort = ("C", "D")] …
```

10

```
/descendant::appartement[position() = 1 or position() = last()]/nom
/descendant-or-self::appartement[position() = 1 or position() = last()]/nom
//appartement[
  (not(preceding::appartement) and not(ancestor::appartement))
  or
  (not(following::appartement) and not(descendant::appartement))
]
//appartement[
  (not(preceding::appartement or ancestor::appartement))
  or
```

```
  (not(following::appartement or descendant::appartement))
]
//appartement[
  (count(preceding::appartement) + count(ancestor::appartement) = 0)
  or
  (count(following::appartement) + count(descendant::appartement) = 0)
]
(//appartement)[position() = 1 or position() = last()]/nom

Incorrecte:
/descendant::appartement[1 or last()]/nom
//appartement[position() = 1 or position() = last()]/nom
/descendant-or-self::node()/appartement[position() = 1 or position() =
last()]/nom
```

11

```
/immobilier/appartements/appartement[descendant-or-
self::appartement[@idAppartement = 'a4a1']]
/immobilier/appartements/appartement[descendant-or-
self::appartement/@idAppartement = 'a4a1']
/immobilier/appartements/appartement[.//@idAppartement = 'a4a1']

//appartement[@idAppartement = 'a4a1']/ancestor-or-
self::appartement[last()]
//appartement[@idAppartement = 'a4a1']/ancestor-or-
self::appartement[not(ancestor::appartement)]

Incorrecte:
/immobilier/appartements/appartement[descendant::appartement/@idAppartement
= 'a4a1']
/immobilier/appartements/appartement[.//appartement/@idAppartement =
'a4a1']
/immobilier/appartements/appartement[//appartement/@idAppartement = 'a4a1']
//appartement[@idAppartement = 'a4a1']/ancestor-or-
self::appartement[not(ancestor-or-self::appartement)]
```

12

```
distinct-values(
  //propriétaire[
    @idPropriétaire = //propriété[
      @idPropriété = //appartement[@comfort = 'A' or tarif >
20000]/@propriété
    ]/@propriétaire
  ]/adresse/pays
)
//propriétaire[
  @idPropriétaire = //propriété[
    @idPropriété = //appartement[@comfort = 'A' or tarif >
20000]/@propriété
  ]/@propriétaire
]/adresse/pays[not(. = preceding::propriétaire/adresse/pays)]
```

Solution exercice 2 :

Le but du TP est d'écrire une feuille XSLT pour transformer le document XML de différentes manières. XSL donne du style au XML, tout comme CSS donne du style au HTML. Il suffira de changer le nom de la feuille dans Immoblier.xml dans la ligne suivante :

<?xml-stylesheet type="text/xsl" href="Immobilier_xsl.xsl"?>. Ensuite, pour exécuter la feuille et voir le résultat, nous avons plusieurs possibilités. Dans certains navigateurs, selon sa version et configuration, il vous suffit d'ouvrir le document XML avec Firefox. Dans XML Copy Editor, il faut ouvrir

le document XML puis utiliser le menu XML, transformation XSL ... (raccourci F8) pour appliquer la feuille sur le document. Le résultat est affiché dans un nouvel onglet.

Document Immobilier_xsl.xsl :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output
            method="html"
            version="4.0"
            encoding="UTF-8"
            indent="yes"
            doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
            doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
    />

    <xsl:template match="/">
        <html>
            <head>
                <title>Rapport immobilier</title>
                <style>
                                            th { background-color: #AAAAAA; }
                    tr.even td { background-color: #DDDDDD; }
                </style>
            </head>
            <body>

                <h2>Tableau des Propriétés</h2>
                <table>
                    <thead>
                        <tr>
                            <th>#</th>
                            <th>ID propriété </th>
                            <th>Nom propriété </th>
                            <th>Nom Propriétaire </th>
                            <th>References Appartement </th>
                            <th>Caractéristiques</th>
                        </tr>
                    </thead>
                    <tbody>
                        <xsl:apply-templates
select="immobilier/propriétés/propriété"/>
                    </tbody>
                </table>

                <h2>Liste des appartements</h2>
                <ol>
                    <xsl:apply-templates
select="immobilier/appartements/appartement"/>
                </ol>

            </body>
        </html>
    </xsl:template>

    <xsl:template match="propriétés/propriété">
                 <xsl:variable name="idPropriétaire" select="@idpropriété"/>
                 <xsl:variable name="numLigne" select="count(preceding-
sibling::propriété) + 1"/>
        <tr>
                        <xsl:if test="($numLigne mod 2) = 0">
                                <xsl:attribute name="class">even</xsl:attribute>
                        </xsl:if>
            <td>
                <xsl:value-of select="$numLigne"/>
            </td>
            <td>
```

```
                <xsl:apply-templates select="$idPropriétaire"/>
            </td>
            <td>
                <xsl:apply-templates select="nom"/>
            </td>
            <td>
                <xsl:call-template name="nomPropriétairePropriété">
                    <xsl:with-param name="idPropriétaire"
select="@refPropriétaire"/>
                </xsl:call-template>
            </td>
            <td>
                <ul>
                    <xsl:apply-templates select="//appartement[@propriétéRef =
$idPropriétaire]" mode="AppartementPropriété"/>
                </ul>
            </td>
            <td>
                <xsl:for-each select="./caractéristiques/caractéristique">
                    <xsl:sort select="text()" lang="cs" order="ascending"/>
                    <xsl:choose>
                                                <!--<xsl:when test="@catégories =
'location'">-->
                        <xsl:when test="concat(' ', normalize-space(@catégories), '
') = ' location '">
                            <span style="color: blue;"><xsl:value-of
select="."/></span>
                        </xsl:when>
                        <xsl:when test="contains(concat(' ', normalize-
space(@catégories), ' '), ' securité ')">
                            <span style="color: red;"><xsl:value-of
select="."/></span>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:value-of select="."/>
                        </xsl:otherwise>
                    </xsl:choose>
                    <xsl:if test="position() != last()">
                        <xsl:text>, </xsl:text>
                    </xsl:if>
                </xsl:for-each>
            </td>
        </tr>
    </xsl:template>

    <xsl:template name="nomPropriétairePropriété">
        <xsl:param name="idPropriétaire" select="@refPropriétaire"/>
        <xsl:variable name="nomPropriétaire" select="//propriétaire[@idPropriétaire
= $idPropriétaire]/nom"/>
        <xsl:if test="$nomPropriétaire">
            <xsl:value-of select="$nomPropriétaire"/>
        </xsl:if>
        <xsl:if test="not($nomPropriétaire)">
                        <i>Inconnue</i>
                </xsl:if>
    </xsl:template>

    <xsl:template match="appartements/appartement">
        <xsl:element name="li">
            <xsl:attribute name="id">
                <xsl:value-of select="concat('appartement-', @idAppartement)"/>
            </xsl:attribute>
            <xsl:text>appartement </xsl:text>
            <xsl:value-of select="@idAppartement"/>
            <xsl:text>: </xsl:text>
            <xsl:value-of select="nom"/>
        </xsl:element>
    </xsl:template>
```

```
    <xsl:template match="appartement" mode="AppartementPropriété">
        <li>
            <xsl:element name="a">
                <xsl:attribute name="href">#appartement-<xsl:value-of
select="@idAppartement"/></xsl:attribute>
                <xsl:value-of select="nom"/>
            </xsl:element>
        </li>
    </xsl:template>

</xsl:stylesheet>
```

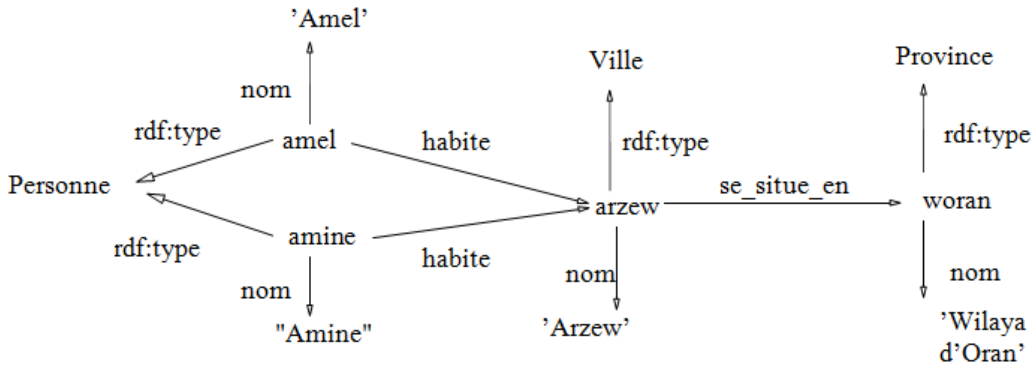
Le document HTML :

Le document HTML « Immobilier_html.html » produit est le suivant :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Real Estate Report</title>
      <style>
                th { background-color: #AAAAAA; }
                   tr.even td { background-color: #DDDDDD; }
               </style>
   </head>
   <body>
      <h2>Tableau des propriétés</h2>
      <table>
         <thead>
            <tr>
               <th>#</th>
               <th>ID propriété</th>
               <th>Nom propriété </th>
               <th>Nom Propriétaire</th>
               <th>References Appartement </th>
               <th>Caractéristiques</th>
            </tr>
         </thead>
         <tbody>
            <tr class="étrange">
               <td>1</td>
               <td>p1</td>
               <td>Résidence Cheminée</td>
               <td>Aymen Karam</td>
               <td>
                  <ul>
                     <li>
                        <a href="#appartement-a1">K0101</a>
                     </li>
                     <li>
                        <a href="#appartement-a2">K0501</a>
                     </li>
                     <li>
                        <a href="#appartement-a3">K1501</a>
                     </li>
                  </ul>
               </td>
               <td>
```

```
                    <span style="color: red;">alarm</span>, <span
style="color: red;">portier</span>, vue sur la rue, <span style="color:
blue;">métro</span></td>
            </tr>
            <tr class="encore">
                <td>2</td>
                <td>p2</td>
                <td>Nouveau gratte-ciel</td>
                <td>Aymen Karam</td>
                <td>
                    <ul></ul>
                </td>
                <td></td>
            </tr>
            <tr class="étrange">
                <td>3</td>
                <td>p3</td>
                <td>Chalet confortable</td>
                <td>
                    <i>Inconnu</i>
                </td>
                <td>
                    <ul></ul>
                </td>
                <td></td>
            </tr>
        </tbody>
    </table>
    <h2>Liste des appartements</h2>
    <ol>
      <li id="appartement-a1">Appartement a1: K0101</li>
      <li id="appartement-a2">Appartement a2: K0501</li>
      <li id="appartement-a3">Appartement a3: K1501</li>
  </ol>
  </body>
</html>
```

Solution exercice 3 :



Solution exercice 4 :

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:geography="http://www.example.org/geography#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >

<rdf:Description rdf:ID='amine'>
   <nom>Amine</nom>
   <rdf:type>
```

```
      <rdf:Description rdf:about='#Personne'/>
   </rdf:type>
   <habite>
      <rdf:Description rdf:ID='arzew'>
         <rdf:type>
            <rdf:Description rdf:about='#Ville'/>
         </rdf:type>
         <nom>arzew</nom>
         <se_situe_en>
            <rdf:Description rdf:ID='idf'>
               <rdf:type>
                  <rdf:Description rdf:about='Province'/>
               </rdf:type>
               <nom>Wilaya d'Oran</nom>
            </se_situe_en>
         </rdf:Description>
      </habite>
   </rdf:Description>
   <rdf:Description rdf:ID='amel'>
      <nom>Amel</nom>
      <rdf:type>
         <rdf:Description rdf:about='#Personne'/>
      </rdf:type>
      <habite>
         <rdf:Description rdf:about='arzew'>
      </habite>
</rdf:Description>
</rdf:RDF>
```

Solution exercice 5 :

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" width="12cm" height="12cm">
   <g style="fill-opacity:0.7; stroke:black; stroke-width:0.1cm;">
     <circle cx="6cm" cy="2cm" r="100" style="fill:red;"
                    transform="translate(0,50)" />
     <circle cx="6cm" cy="2cm" r="100" style="fill:blue;"
                    transform="translate(70,150)" />
     <circle cx="6cm" cy="2cm" r="100" style="fill:green;"
                    transform="translate(-70,150)"/>
   </g>
</svg>
```

La première ligne contient la déclaration XML. Le code SVG commence par l'élément <svg>, qui se compose de la balise d'ouverture <svg> et de la balise de fermeture </svg>. C'est l'élément racine. Les attributs width et height définissent la largeur et la hauteur du document SVG. L'attribut version définit la version SVG à utiliser et l'attribut xmlns définit l'espace de noms SVG. L'élément SVG <circle> est utilisé pour créer un cercle. Les attributs cx et cy définissent les coordonnées x et y du centre du cercle. Si cx et cy sont omis, le centre du cercle est défini sur (0, 0). L'attribut r définit le rayon du cercle. Les attributs stroke et stroke-width contrôlent l'apparence du contour d'une forme. L'attribut fill fait référence à la couleur à l'intérieur d'une forme. La balise de fermeture </svg> ferme l'élément SVG racine et le document.

Solution exercice 6 :

```
<?xml version="1.0" standalone="no"?>
<svg width="700" height="560" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <desc> Quatre rectangles séparés
  </desc>
  <rect x="20" y="20" width="370" height="272"/>
  <rect x="430" y="20" width="250" height="272"/>
  <rect x="20" y="332" width="370" height="208"/>
```

```
  <rect x="430" y="332" width="250" height="208"/>

  <!-- Affichez le contour de la toile en utilisant l'élément "rect"-->
  <rect x="1" y="1" width="698" height="558"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```

Solution exercice 7 :

Il faut lancer Eclipse et créer un nouveau projet dans le Workspace Java. Vérifier qu'il sépare bien les sources src des binaires bin. Ensuite il faut télécharger le fichier « simple-xml-2.7.1.jar ». Ce framework facilite le développement de systèmes XML avec un minimum d'effort et des erreurs réduites. Il offre une sérialisation et une dé-sérialisation complètes des objets, en conservant chaque référence rencontrée. Il faut ajouter ce fichier au projet Eclipse. Il faut créer un dossier appelé lib au même niveau que src, y placer ce fichier jar.

Le code complet pour réponde à la question est ci-dessous. Ce document est un code source, donc à placer dans le dossier src. Il contient des méthodes pour ouvrir, lire, écrire des fichiers XML et d'autres méthodes avec des commentaires pour répondre aux questions de l'exercice. Ensuite, dans Eclipse, taper F5 pour rafraîchir le projet, puis déplier le dossier lib et cliquer droit sur le jar pour le menu contextuel et choisir Build Path/Add to Build Path. Une autre façon est d'aller dans Build Path/Configure Build Path, onglet libraries ajouter le fichier jar avec Add External JARs.

Le code complet :

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;

public class DOMSolution{
    private static final String INPUT_FILE = "data.xml";
    private static final String OUTPUT_FILE = "data.out.xml";

    public static void main(String[] args) {

        try {

            // 1. Création d'une nouvelle instance de DOM parser factory.
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            // 2. Configuration de factory : le fichier source ne doit pas être
validé.
            dbf.setValidating(false);
            // 3. Création d'une nouvelle instance de DOM parser.
            DocumentBuilder builder = dbf.newDocumentBuilder();
            // 4. Analyse du fichier d'entrée et création d'un arbre DOM d'objets.
            Document doc = builder.parse(data.xml);
            // 5. Traitement et transformation de l'arbre DOM.
            processTree(doc);

            // 6. Création d'une nouvelle instance de DOM serializer factory.
            TransformerFactory tf = TransformerFactory.newInstance();
            // 7. Création d'une nouvelle instance de DOM serializer.
            Transformer writer = tf.newTransformer();
            // 8. Configuration de la sortie : réglage de l'encodage des fichiers.
            writer.setOutputProperty(OutputKeys.ENCODING, "utf-8");
            // 9. Sérialisation de l'arbre DOM modifié dans un document XML de
sortie.
            writer.transform(
                new DOMSource(doc),
```

```
                new StreamResult(new File(data.out.xml))
            );

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    /**
     * Traite l'arborescence du document du point de vue de l'utilisateur
     * @param doc Document à analyser
     */
    //!ADAPTER au type de résultat et à la liste de paramètres
   // Transformation des éléments de taux des appartements en attributs
    private static void processTree(Document doc) {

        NodeList rateList = doc.getElementsByTagName("rate");

        for (int i = rateList.getLength() - 1; i >= 0; --i) {

            Element rate = (Element)rateList.item(i);
            Element flat = (Element)rate.getParentNode();

            flat.setAttribute("rate", rate.getTextContent());
            flat.removeChild(rate);

        }

    }

}
```

Solution exercice 8 :

Pour cet exercice, il faut adapter le contenu de la méthode « processTree » de l'exercice précédent.

```
  // Suppression des appartements dont le taux est >= 20000
    private static void processTree(Document doc) {

                NodeList appartements =
doc.getElementsByTagName("appartement");

        for (int i = appartements.getLength() - 1; i >= 0; --i) {

            Element appartement = (Element)appartements.item(i);
            Element rate =
(Element)appartement.getElementsByTagName("rate").item(0);

            int value = Integer.parseInt(rate.getTextContent().trim());
            if (value >= 20000) {
                appartement.getParentNode().removeChild(appartement);
            }
        }
}
```

Solution exercice 9 :

```
// Création d'un nouvel enregistrement de propriétaire avec toutes les
données respectives.
    private static void processTree(Document doc) {

        Element propriétaire = doc.createElement("propriétaire");

        propriétaire.setAttribute("idPropriétaire", "o2");
```

```
        propriétaire.appendChild(doc.createElement("nom")).
            setTextContent("Aymen Karam");

        Element adresse = doc.createElement("adresse");
        adresse.appendChild(doc.createElement("rue")).
            setTextContent("Soukenne namesti 121/1");
        adresse.appendChild(doc.createElement("codePostal")).
            setTextContent("46001");
        adresse.appendChild(doc.createElement("province")).
            setTextContent("Liberec");
        adresse.appendChild(doc.createElement("pays")).
            setTextContent("Algerie");
        propriétaire.appendChild(adresse);

        NodeList propriétairesList =
doc.getElementsByTagName("propriétaires");
        Element propriétaires;
        if (propriétairesList.getLength() == 0) {
            propriétaires = doc.createElement("propriétaires");
            NodeList propertiesList =
doc.getElementsByTagName("properties");
            Element properties = (Element)propertiesList.item(0);
            doc.getDocumentElement().insertBefore(propriétaires,
properties);
        } else {
            propriétaires = (Element)propriétairesList.item(0);
        }
        propriétaires.appendChild(propriétaire);
    }
```

Solution exercice 10 :

Il faut Lancer Eclipse et créez un projet dans le Workspace Java favori. Il faut Vérifier qu'il sépare bien les sources src des binaires bin. **L'API SAX** permet de parcourir un document XML très rapidement et sans occuper beaucoup de mémoire. Le programme complet est le suivant. Il faut ajouter ce fichier aux sources de votre projet. Il contient des méthodes pour faciliter la lecture et l'écriture de fichiers XML et d'autres méthodes (bien commentées) pour répondre aux questions de l'exercice.

Le code complet :

```
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.InputSource;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.LinkedList;
import java.util.List;

// Classe principale pour l'analyseur SAX
public class SAXSolution {

    // Chemin d'accès au fichier d'entrée
    private static final String INPUT_FILE = "data.xml";

    // Classe principale
    public static void main(String[] args) {

        OutputStreamWriter outputStreamWriter = new OutputStreamWriter(System.out);
```

```
        try {
            // Créer une instance d'analyseur
            XMLReader parser = XMLReaderFactory.createXMLReader();
            // Créer un flux d'entrée à partir du document XML source
            InputSource source = new InputSource(INPUT_FILE);
            // Définir notre gestionnaire de contenu personnalisé pour la gestion
des événements SAX
            parser.setContentHandler(
                new SAXSolution_Handler(outputStreamWriter)
            );
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                outputStreamWriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }

}
/**
 * Gestionnaire de contenu personnalisé pour la gestion des événements SAX
 * @voir http://www.saxproject.org/apidoc/org/xml/sax/ContentHandler.html
 */
class SAXSolution_Handler implements ContentHandler {
    // Référence au localisateur
    Locator locator;

    Integer indent;

    OutputStreamWriter outputStreamWriter;

    private final Integer TAB_SIZE = 4;

    private String currentElementName;

    private final List<String> validationMessages = new LinkedList<>();

    private boolean caracteristiquesDetecte;
    private int nombreCaracteristiques;
    private StringBuffer nombreAppartements;

    public SAXSolution_Handler(OutputStreamWriter outputStreamWriter) {
        this.outputStreamWriter = outputStreamWriter;
        indent = 0;
    }

      /**
      * Stocke la référence au localisateur
      * @param Locator locator Emplacement dans le fichier source
      */
    @Override
    public void setDocumentLocator(Locator locator) {
        this.locator = locator;
    }
    /**
      * Gestionnaire de début de document
      * @throws SAXException
      */
    @Override
    public void startDocument() throws SAXException {
        printIndented("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    }
/**
      * Gestionnaire de début d'élément
```

```
      * @param uri URI de l'espace de noms de l'élément (vide s'il n'y a pas
d'espace de noms)
      * @param localName Nom local de l'élément (jamais vide)
      * @param qName Nom qualifié
      * @param atts Attributs
      * @throws SAXException
      */
    @Override
    public void startElement(
        String uri, String localName, String qName, Attributes atts
    ) throws SAXException {

        printIndented(String.format("<%s>", qName));

        currentElementName = qName;

        printAttributes(atts);
        validateIdentifiers(atts);

        if (qName.equals("propriété")) {
            caracteristiquesDetecte = false;
            nombreCaracteristiques = 0;
        }
        if (qName.equals("caractéristiques")) {
            caracteristiquesDetecte = true;
        }
        if (qName.equals("caractéristique")) {
            ++nombreCaracteristiques;
        }

        if (qName.equals("nombreAppartements")) {
            nombreAppartements = new StringBuffer();
        }

        ++indent;

    }
    /**
      * Gestionnaire de fin d'élément
      */
    @Override
    public void endElement(
        String uri, String localName, String qName
    ) throws SAXException {

        if (localName.equals("immobilier")) {
            printErrors();
        }
      /**  Chaque propriété doit avoir au moins 2 et au plus 5 caractéristiques
particulières lorsqu'une liste
        *   de caractéristiques est fournie.
        */
        if (localName.equals("propriété")) {
            if (caracteristiquesDetecte && (nombreCaracteristiques < 2 ||
nombreCaracteristiques > 5)) {
                reportError("La propriété doit avoir au moins 2 et au plus 5
caractéristiques");
            }
        }
        // La valeur de l'élément nombreAppartements doit être non négative.
        if (qName.equals("nombreAppartements")) {
            String s = nombreAppartements.toString().trim();
            try {
                int n = Integer.parseInt(s);
                if (n < 0) {
                    reportError("Le nombre d'appartements doit être un nombre non
négatif");
                }
```

```
            } catch (NumberFormatException e) {
                reportError(
                    String.format("Mauvais format numérique du nombre
d'appartements : %s", s)
                );
            }
        }

        --indent;
        printIndented(String.format("</%s>", qName));

    }

    /**
      * Gestionnaire de données de caractères
      * - Le contenu textuel peut être diffusé dans le cadre de plusieurs
événements, et non d'un seul
      * @param chars Tableau avec des données de caractères
      * @param start Index de la position de départ dans le tableau
      * @param length Nombre de caractères à lire dans le tableau
      * @throws SAXException
      */
    @Override
    public void characters(
        char[] chars, int start, int length
    ) throws SAXException {

        String s = new String(chars, start, length).toUpperCase().trim();

        if (s.length() > 0) {

            printIndented(s);

            if (currentElementName.equals("nombreAppartements")) {
                nombreAppartements.append(s);
            }

        }

    }

    private void printIndented(String what) {
        // https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html
        try {
            if (indent > 0) {
                outputStreamWriter.write(
                    String.format("%1$" + (indent * TAB_SIZE) + "s", "")
                );
            }
            outputStreamWriter.write(what + "\r\n");
            outputStreamWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    private void printAttributes(Attributes atts) {
        if (atts.getLength() > 0) {
            ++indent;
            for (int i = 0; i < atts.getLength(); ++i) {
                String name = atts.getQName(i);
                printIndented(
                    String.format("<%s>%s</%s>", name, atts.getValue(i), name)
                );
            }
            --indent;
        }
```

```
    }
    /**
    *Les identifiants doivent commencer par les lettres suivantes : propriétaires :
o, agences : a, immeubles
    * : p, appartements : a.
    */
    private void validateIdentifiers(Attributes atts) {

        if (atts.getLength() > 0) {
            if (currentElementName.equals("propriétaire") &&
(atts.getValue("idPropriétaire") == null ||
!atts.getValue("idPropriétaire").startsWith("o"))) {
                reportError("L'élément de type Propriétaire doit avoir un
identifiant qui commence par 'o' ");
            }
            if (currentElementName.equals("Appartement") &&
(atts.getValue("idAppartement") == null ||
!atts.getValue("idAppartement").startsWith("a"))) {
                reportError("Un élément de type Appartement devrait avoir un
identifiant qui commence par 'a' ");
            }
            if (currentElementName.equals("propriété") &&
(atts.getValue("idPropriété") == null ||
!atts.getValue("idPropriété").startsWith("p"))) {
                reportError("L'élément de type Propriété doit avoir un identifiant
qui commence par 'p' ");
            }
            if (currentElementName.equals("agence") && (atts.getValue("idAgence")
== null || !atts.getValue("idAgence").startsWith("a"))) {
                reportError("L'élément de type Agence devrait avoir un identifiant
qui commence par 'a'.");
            }
        }

    }

    /**
    *Pour chaque erreur, indiquer sa position (ligne, colonne) et sa description
textuelle.
    */
    private void reportError(String cause) {
        validationMessages.add(
            String.format(
                "Erreur : %s sur la ligne %d colonne %d.",
                cause, locator.getLineNumber(), locator.getColumnNumber()
            )
        );
    }
   /**
    * Imprimer les messages d'erreur dans un nouvel élément erreurs à la fin du
document XML de sortie
    */
    private void printErrors() {
        if (!validationMessages.isEmpty()) {
            printIndented("<erreurs>");
            ++indent;
            for (String m : validationMessages) {
                printIndented(m);
            }
            --indent;
            printIndented("</erreurs>");
        }
    }
}
```

Solution exercice 11 :

XPointer nous permet de créer un lien vers des parties spécifiques du document XML, au lieu de créer un lien vers le document entier comme avec XLink. XPointer utilise des expressions XPath pour naviguer dans le document XML.

Pour créer un lien vers une partie spécifique d'une page, il faut ajouter le signe dièse (#) et une expression XPointer après l'URL dans l'attribut xlink:href, comme ceci : xlink:href="https://chien.com/racesChiens.xml #xpointer(id('Rottweiler'))". L'expression fait référence à l'élément dans le document cible, avec la valeur d'id "Rottweiler".

XPointer est également une méthode abrégée pour créer un lien vers un élément avec un identifiant. Nous pouvons utiliser directement la valeur de l'identifiant, comme ceci : xlink:href="https://chien.com/racesChiens.xml#Rottweiler".

Le document XML suivant contient des liens vers plus d'informations sur la race de chien pour chaque chien :

```
<?xml version="1.0" encoding="UTF-8"?>
<mesChiens xmlns:xlink="http://www.w3.org/1999/xlink">
<monChien>
  <description>
   Anton est mon chien préféré. Il a gagné beaucoup de .....
  </description>
  <fait xlink:type="simple" xlink:href="https://chien.com/raceChiens.xml#Rottweiler
">
 Les faits concernant le Rottweiler
  </fait>
</monChien>
<monChien>
  <description>
 Pluto est le chien le plus doux de la terre......
  </description>
  <fait xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#FCRetriever">
 Les faits sur le Flat Coated Retriever
  </fait>
</monChien>
</mesChiens>
```

Solution exercice 12 :

XLink est une spécification du W3C (appelé parfois XLL pour XLink Language). Il permet de créer des liens entre des fichiers XML ou des fragments de fichiers XML (grâce à XPointer). Dans la solution proposée, les chapitres sont considérés comme des ressources locales et le plan du livre comme des liens ("arc").

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE plan SYSTEM "plan.dtd"-->
<plan xlink:type="extended" xmlns:xlink="http://www.w3.org/1999/xlink">
  <livre>
    <partie no="I" titre="XML Technologies">
        <chapitre xlink:label="chap1" xlink:type="ressource">HTML and Web
Pages</chapitre>
        <chapitre xlink:label="chap2" xlink:type="ressource">XML
Documents</chapitre>
        <chapitre xlink:label="chap3" xlink:type="ressource">Navigating XML Trees
with XPath</chapitre>
        <!-- ... -->
    </partie>
    <partie no="II" titre="WEB Technologies">
        <chapitre xlink:label="chap8" xlink:type="ressource">The HTTP
Protocol</chapitre>
        <chapitre xlink:label="chap9" xlink:type="ressource">Programming Web
Applications
            with Servlets</chapitre>
```

```
        <chapitre xlink:label="chap10" xlink:type="ressource">Programming Web
Applications
            with JSP</chapitre>
        <!-- ... -->
    </partie>
  </livre>
    <organisation>
        <lien xlink:from="chap1" xlink:to="chap2" xlink:type="arc"/>
        <lien xlink:from="chap1" xlink:to="chap8" xlink:type="arc"/>
        <lien xlink:from="chap2" xlink:to="chap9" xlink:type="arc"/>
        <lien xlink:from="chap2" xlink:to="chap3" xlink:type="arc"/>
        <!-- ... -->
    </organisation>
</plan>
```

5 Bases de données XML et bases de données semi-structurées

5.1 Données semi-structurées et XML

Dans certaines applications, les données sont collectées de manière ad hoc avant que l'on sache comment elles seront stockées et gérées. Ces données peuvent avoir une certaine structure, mais toutes les informations collectées n'auront pas une structure identique. Ce type de données est connu sous le nom de données semi-structurées. Dans les données semi-structurées, les informations du schéma sont mélangées aux valeurs des données, car chaque objet de données peut avoir différents attributs qui ne sont pas connus à l'avance. C'est pourquoi ce type de données est parfois appelé "données autodescriptives".

Le modèle de données semi-structurées est un moyen flexible de décrire les données. Cette flexibilité en fait un bon modèle de données pour l'échange de données. Il est généralement facile de convertir un ensemble de données donné en une représentation semi-structurée. Il est souvent facile d'effectuer une transformation d'une représentation semi-structurée à une autre.

Les données semi-structurées peuvent être représentées sous la forme d'un **graphe dirigé** avec des **nœuds** et des **arcs** avec des **étiquettes** (ou balises) entre les nœuds.

- Les étiquettes ou balises sur les arêtes dirigées représentent les noms du schéma - les noms des attributs, des types d'objets (ou des types d'entités ou des classes) et des relations.
- Un nœud racine unique, sans arcs entrants, représente la totalité de la base de données.
- Les nœuds internes ont des arcs qui entrent et sortent, mais pas de données et représentent des objets individuels ou des attributs composites.
- Les nœuds feuilles, sans arcs sortants, représentent les valeurs réelles des données des attributs simples (atomiques).

Exemple :

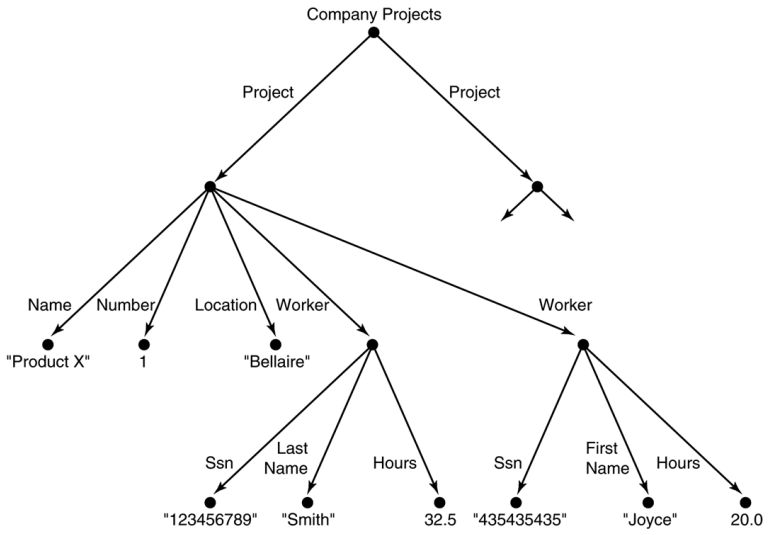


Figure 17 Représentation de données semi-structurées sous forme de graphique

Les données semi-structurées se présentent sous différents formats, en fonction de leur source d'origine. XML est devenu l'un des formats de données semi-structurées les plus populaires. Ce langage de balisage polyvalent et facile à utiliser permet aux utilisateurs de définir les balises et les attributs nécessaires au stockage des données sous une forme hiérarchique. Une représentation comme celle que propose XML permet de manipuler plus facilement des informations irrégulières, grâce à la possibilité de structurer et d'annoter des informations sans la définition d'un schéma fortement structuré et contraignant.

XML est principalement destiné aux données semi-structurées qui sont un arbre, c'est-à-dire où tous les nœuds (sauf la racine) ont exactement un arc entrant. Un arc dans l'arbre avec l'étiquette L par

exemple pointant vers un nœud n dans les données semi-structurées est représenté dans le document XML comme une paire de balises : <L>...</L>. Ici ... est la description XML de la partie des données semi-structurées. Les nœuds feuilles sont représentés par les données qu'ils contiennent. Donc, le langage XML représente les informations sous forme de documents textuels annotés et structurés par des balises. La structure générée par les balises correspond à une arborescence d'éléments. Donc, il existe deux principaux concepts de structuration qui sont utilisés pour construire un document XML : les éléments et les attributs. Les attributs fournissent des informations supplémentaires qui décrivent les éléments.

Par exemple, le document XML ci-dessous intègre les informations des deux projets de l'arbre. Les projets sont représentés sous forme d'éléments <project> qui contiennent les informations connues des deux projets.

```
<?xml version="1.0" standalone="yes"?>
<projects>

 <project>
  <Name>ProductX</Name>
  <Number>1</Number>
  <Location>Bellaire</Location>
  <DeptNo>5</DeptNo>
  <Worker>
   <SSN>123456789</SSN>
   <LastName>Smith</LastName>
   <hours>32.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <FirstName>Joyce</FirstName>
   <hours>20.0</hours>
  </Worker>
 </project>
 </project>
  <Name>ProductY</Name>
  <Number>2</Number>
  <Location>Sugarland</Location>
  <DeptNo >5</DeptNo >
  <Worker>
   <SSN>123456789</SSN>
   <hours>7.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <hours>20.0</hours>
  </Worker>
  <Worker>
   <SSN>333445555</SSN>
   <hours>10.0</hours>
  </Worker>
 </project>

...

</projects>
```

Comme en HTML, les éléments sont identifiés dans un document par leur balise de début et leur balise de fin. Les noms des balises sont enfermés entre des parenthèses angulaires <...>, et les balises de fin sont en outre identifiées par une barre oblique inverse </...>.

Les éléments **complexes** sont construits à partir d'autres éléments de manière hiérarchique, tandis que les éléments **simples** contiennent des valeurs de données.

Il est facile de voir la correspondance entre la représentation textuelle XML et la structure arborescente. Dans la représentation arborescente, les nœuds internes représentent des éléments complexes, tandis que les nœuds des feuilles représentent des éléments simples. C'est pourquoi le modèle XML est appelé un modèle arborescent ou un modèle **hiérarchique**.

Les arbres XML peuvent facilement être convertis en expressions de données semi-structurées et réciproquement. Par exemple :

XML :

```
<livre année="2002">
<titre> Prélude à
fondation</titre>
<auteur>
<nom> Asimov </nom>
<prénom> Isaac </prénom>
</auteur>
<éditeur> Pocket </éditeur>
<prix>6</prix>
</livre>
```

DSS :

```
{livre:
{année:2002,
titre:Prélude à fondation,
auteur:
{nom:Asimov,
prénom:Isaac},
éditeur:Pocket, prix:6}
}
```

5.2 Les langages de requêtes

Aujourd'hui, XML s'avère être le nouveau standard le plus important utilisé pour la représentation des données et l'échange sur le WWW. De nombreux nouveaux langages ont été projetés pour extraire et reformer le contenu de XML. Certains des langages existants sont des langages conventionnels (tels que SQL, OQL), et d'autres langages sont plus récents et très motivés par XML.

5.2.1 XML-QL

XML-QL est un langage de requête XML qui est utilisé pour le processus d'interrogation, de construction, de conversion et d'incorporation des données XML. Le langage affiche essentiellement XML sous forme de données semi-structurées comprenant une structure irrégulière ou se développant rapidement. Pour faire correspondre les données dans un document XML, des modèles sont utilisés par XML-QL.

Une extension de XML-QL connue sous le nom d'Elixir a été projetée pour assister les requêtes classées en fonction de la ressemblance textuelle. Dans le cas de XML-QL, le client est autorisé à interroger un document XML similaire à une BD et à illustrer une construction de sortie.

5.2.2 Lorel

Un des premiers langages de requête utilisé pour les données semi-structurées, le langage Lorel utilise l'OEM (Object Exchange Model) comme modèle de données pour les données semi-structurées. Lorel est utilisé pour développer OQL (Object Query Language) pour la procédure d'interrogation des éléments. Cela se fait en dépendant de la coercition à différents niveaux pour retenir le typage puissant d'OQL. Lorel est également utilisé pour étendre OQL avec des expressions de chemin. Ceci est fait afin que l'utilisateur puisse indiquer les modèles qui correspondent aux chemins réels dans les données référencées.

L'un des avantages du langage Lorel est sa syntaxe simple qui permet aux utilisateurs de le comprendre plus clairement. L'inconvénient du langage est qu'il dépend de l'analyseur OQL. Il comprend également des fonctionnalités limitées.

5.2.3 Quilt

Quilt est considéré comme un langage fonctionnel où l'on peut représenter une requête comme une expression. Les expressions de Quilt comprennent sept formes principales : expression de chemin, constructeurs d'éléments, expressions FLWR, expressions avec opérateur et fonctions, expressions conditionnelles, quantificateurs, et liaisons de variables.

Outre les opérations de jointure, Quilt inclut également des expressions imbriquées. Ainsi, il comprend principalement une sous-requête à l'intérieur d'une seule requête. Pour la progression de XQuery (discuté plus tard), des traits importants de Quilts ont été utilisés.

5.2.4 XQL

XQL est un autre langage d'interrogation XML qui utilise des expressions de chemin. Ainsi, les principales constructions de XQL correspondent directement aux principales structures de XML. En raison de cette nature, XQL est considéré comme très proche de XPath. Dans le cas de XQL, les nœuds comprennent l'identité. De même, l'identité, les séries de résultats de la requête et les associations de confinement sont maintenues par les nœuds. Les nœuds proviennent de différentes sources. Mais, le processus d'introduction des nœuds dans la requête n'est pas spécifié par XQL. Les jointures et certaines fonctions sont également supportées par XQL.

XQL est généralement utilisé pour rechercher et trier le texte et les éléments dans un document XML. XQL propose un dispositif utilisé pour rechercher et/ou sélectionner des éléments particuliers dans la collecte de données. Il s'appuie sur la syntaxe de modèle qui est utilisée dans le langage XSL et est projetée comme une extension de celui-ci. Une manière déclarative dans laquelle des éléments particuliers sont spécifiés pour le traitement est effectuée par le langage de modèle XSL. La notation de répertoire simple est utilisée par celui-ci. La possibilité d'utiliser la logique booléenne, de trier les éléments, d'indexer dans une collection d'éléments est ajoutée aux détails de ce répertoire par XQL. Au moyen de XQL, vous pouvez écrire un programme pour rechercher des référentiels de fichiers XML. Ceci est fait pour offrir des liens hypertextes vers des éléments particuliers, et pour d'autres applications.

La requête XQL comprend des expressions plus courtes faciles à utiliser. Cependant la sémantique peut ne pas être très instinctive.

5.2.5 XQuery

XQuery est un langage de requête XML produit par W3C et généré pour les sources de données XML. XQuery est considéré comme un langage de requête standard de l'industrie.

L'accès déclaratif aux données XML est fourni par XQuery de la même manière que SQL le fait pour les données relationnelles. XQuery utilise XPath comme base et utilise des expressions XPath. Ce langage sera détaillé dans une prochaine section.

5.3 Les bases de données XML

Indépendamment du fait que XML soit utilisé comme format de stockage ou d'échange pour les données de modèle centrées sur les données, ou utilisé pour créer des documents de modèle centrés sur les documents semi-structurés, tels que XHTML, il est parfois nécessaire de stocker le XML dans une sorte de référentiel ou de base de données qui permet un stockage et une récupération plus sophistiqués des données, en particulier si le XML doit être accessible à plusieurs utilisateurs.

Il existe conceptuellement deux manières différentes de stocker des documents XML dans une base de données (BD). La première consiste à mapper le modèle de données du document sur un modèle de BD et à convertir les données XML dans la BD, en fonction de ce mappage. La seconde consiste à mapper le modèle XML dans un ensemble fixe de structures persistantes (BD) pouvant stocker n'importe quel document XML. Les BDs qui prennent en charge la première méthode sont appelées **bases de données compatibles XML (XML-enabled databases)**. Les BDs qui prennent en charge la deuxième méthode sont appelées **bases de données XML natives (native XML databases)**. Les BDs compatibles XML ont leur propre modèle de données – relationnel, hiérarchique, orienté objet, et elles mappent des instances du modèle de données XML à des instances de leur modèle de données. Les BDs XML natives utilisent directement le modèle de données XML. Bien que le choix soit quelque peu arbitraire, il est généralement plus pratique et plus efficace de stocker et de manipuler des documents XML centrés sur les données à l'aide de BDs compatibles XML, et des documents XML centrés sur les documents à l'aide de BDs XML natives.

5.3.1 Bases de données compatibles XML

Une façon de mapper des documents XML dans une **BD compatible XML** consiste à créer des vues relationnelles sur des documents XML stockés dans des colonnes d'une BD relationnelle, qui peuvent ensuite être interrogées à l'aide de SQL. Dans les BDs compatibles XML, différents schémas de

document XML correspondent à différents schémas de BD. XML est externe à la BD et invisible à l'intérieur de celle-ci.

Voici un exemple de document étiqueté en XML (un extrait d'un menu de restaurant), contenu dans le fichier simple.xml :

```
<?xml  version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE  menu  (ViewSourceforfulldoctype...)  >
<menu  date='5.10.2006'>
    <food>
        <name>Homemade Bean  Soup  </name>
        <price>100.00  </price>
        <description>
            a bowl  of  white  bean  soup  with  paper  and  onion
        </description>
        <calories>650  </calories>
    </food>
    <food>
        <name>French  Toast  </name>
        <price>60.50din  </price>
        <description>
            thick  slices  made from  our  homemade  bread
        </description>
        <calories>300  </calories>
    </food>
    <food>
        <name>Homestyle  Breakfast  </name>
        <price>195.95din  </price>
        <description>
            two  eggs,  bacon  or  sausage,  toast
        </description>
        <calories>950  </calories>
    </food>
</menu>
```

Comme exemple d'une BD compatible XML, nous pouvons considérer le document XML "menu". Il peut être représenté (soit physiquement, soit sous forme de vue relationnelle) comme la relation suivante :

food	name	price	description	calories
	Homemade Bean Soup	100.00	a bowl of white bean soup with paper and onion	650
	French Toast	60.50	thick slices made from our homemade bread	300
	Homestyle Breakfast	195.95	two eggs, bacon or sausage, toast	950

5.3.2 Bases de données XML native

La BD XML native (NXD) est décrite comme une BD qui a un document XML (ou sa partie racine) comme unité fondamentale de stockage (logique) et qui définit un modèle (logique) pour un document XML, par opposition aux données de ce document (son contenu). Elle représente un modèle logique de document XML (et non un modèle de données de document XML), et stocke et manipule les documents en fonction de ce modèle. Ces BDs sont des sortes de bases NoSQL (not only SQL). La structuration des données et les méthodes d'interrogation sont d'un autre genre que les requêtes SQL sur une base relationnelle. Les caractéristiques de base d'une BD NXD sont les suivantes :

- une unité **logique** d'un NXD est un document XML ou sa partie racine, et elle correspond à une ligne dans une BD relationnelle,
- elle comprend au moins les composants suivants : éléments, attributs, données textuelles (PCDATA) et ordre des documents,

- le modèle **physique** (et le type de stockage NXD persistant) n'est pas spécifié. Cela implique que le stockage persistant des documents XML peut être arbitraire, tant qu'il stocke et manipule un document XML comme un tout (logique). Les différents types de stockage persistant peuvent être des fichiers sur disque, des champs CLOB dans les BDs relationnelles, des tables fixes de BDs relationnelles, des arbres DOM dans les BDs orientées objet, des tables de hachage, etc.

Dans une BD XML native, XML est visible à l'intérieur de la BD. Il existe une BD unique pour tous les schémas et documents XML. Les BDs XML natives sont particulièrement adaptées au stockage de données irrégulières, profondément hiérarchisées et récursives.

Par exemple, un document de menu XML peut être représenté dans une BD XML native stockée dans des tables de BD relationnelles fixes de la manière suivante :

Documents	Doc id	Doc name
	1	simple.xml
	. . .	. . .

Elements	Doc id	El id	Parent id	Name	OrdInParent
	1	1	NULL	menu	1
	1	2	1	food	1
	1	3	2	name	1
	1	4	2	price	2
	. . .	. . .	. . .	. . .	. . .

Attributes	Doc id	Atr id	Parent id	Name	Value
	1	1	1	date	5.10.2006
	. . .	. . .	. . .	. . .	. . .

Text	Doc id	Text id	Parent id	Value
	1	1	3	Homemade Bean Soup
	1	2	4	100.00
	. . .	. . .	. . .	. . .

Les avantages de l'utilisation de BDs XML natives par rapport à d'autres types de BDs sont nombreux. Elles évitent aux utilisateurs de devoir connaître le schéma du document à l'avance (avant de concevoir la BD), elles prennent en charge des modèles de données qui ne conviennent pas à d'autres BDs (par exemple, les BDs relationnelles), elles offrent une extensibilité, etc.

Les applications NXD courantes comprennent la gestion de documents, la prise en charge de données semi-structurées, la prise en charge de données éparses, les données de catalogue, les bases de données de pièces de fabrication, le stockage d'informations médicales, etc.

6 XQUERY et les bases de données

6.1 XML et données semi structurées

6.1.1 Bases de Données semi-structurées

L'utilisation de l'internet et le développement de la théorie des bases de données s'influencent mutuellement. Le contenu des sites web est généralement stocké dans des bases de données, tandis que les sites web et les références entre eux peuvent également être considérés comme une base de données qui n'a pas de schéma fixe au sens habituel du terme. Le contenu des sites et les références entre les sites sont décrits par les sites eux-mêmes, nous ne pouvons donc parler que de données semi-structurées, qui peuvent être mieux caractérisées par des graphes étiquetés dirigés. Dans le cas des données semi-structurées, les méthodes récursives sont utilisées plus souvent pour donner des structures de données et des requêtes que dans le cas des bases de données relationnelles classiques. Différents problèmes de bases de données, par exemple les restrictions, les dépendances, les requêtes, le stockage distribué, les autorités, le traitement de l'incertitude, doivent tous être généralisés en fonction de cela. La semi-structuration soulève également de nouvelles questions. Puisque les requêtes ne forment pas toujours un système fermé comme dans le cas des bases de données classiques, c'est-à-dire que l'applicabilité des requêtes l'une après l'autre dépend du type de résultat obtenu, le problème de la vérification des types devient plus important.

Le modèle semi-structuré est un modèle de base de données où il n'y a pas de séparation entre les données et le schéma, et où la quantité de structure utilisée dépend de l'objectif.

Les avantages de base de données semi-structurée sont les suivants :

- Elle peut représenter les informations de certaines sources de données qui ne peuvent pas être contraintes par un schéma.
- Elle fournit un format flexible pour l'échange de données entre différents types de bases de données.
- Elle peut être utile de voir les données structurées comme semi-structurées (à des fins de navigation).
- Le schéma peut être facilement modifié.
- Le format de transfert des données peut être portable.

La principale contrepartie de l'utilisation d'un modèle de base de données semi-structurée est que les requêtes ne peuvent pas être effectuées aussi efficacement que dans une structure plus contraignante, telle que le modèle relationnel. En général, les enregistrements d'une base de données semi-structurée sont stockés avec des identifiants uniques qui sont référencés avec des pointeurs vers leur emplacement sur le disque. Cela rend les requêtes basées sur la navigation ou le chemin d'accès assez efficaces, mais pour effectuer des recherches sur de nombreux enregistrements (comme c'est le cas en SQL), ce n'est pas aussi efficace car il faut chercher sur le disque en suivant les pointeurs.

6.2 XQUERY

XQuery est un langage fonctionnel basé sur des requêtes utilisé pour récupérer des informations stockées au format XML. XQuery peut être utilisé sur des documents XML, des bases de données relationnelles contenant des données au format XML ou des bases de données XML. XQuery est à XML ce que SQL est aux bases de données. Ce langage est basé sur XPath et utilise des expressions XPath pour naviguer dans les documents XML. Il est pris en charge par toutes les principales bases de données. XQuery est compatible avec plusieurs normes W3C, telles que XML, Namespaces, XSLT, XPath et XML Schema. XQuery 1.0 est devenu une recommandation du W3C en 2007.

6.2.1 Syntaxe XQuery

XQuery peut être considéré comme un surensemble de SQL avec des extensions de fonctionnalités de ce langage sur les tables (collections de tuples) pour supporter des opérations similaires sur les forêts (collections d'arbres). Ces extensions intègrent les projections d'arbres sur des sous-arbres,

des sélections d'arbres et de sous-arbres en utilisant des prédicats sur les valeurs des feuilles, utilisation de variables dans les requêtes pour mémoriser un arbre ou pour itérer sur des collections d'arbres, combinaison des arbres extraits de collections en utilisant des jointures d'arbres, réordonnancement des arbres, imbrication des requêtes, calculs d'agrégats et utilisation possible des fonctions utilisateur.

Une requête XQuery est composée d'expressions (simples et complexes) associées à des opérateurs et à des fonctions. Une expression a une valeur ou retourne une erreur. Une valeur est une séquence ordonnée d'items, qui peuvent être un nœud ou une valeur atomique (feuille) de l'arbre XML. Les nœuds et les valeurs sont typés.

Quelques règles de syntaxe de base :

- XQuery est sensible à la casse et les éléments, attributs et variables XQuery doivent être des noms XML valides.
- Une valeur de chaîne XQuery peut être entre guillemets simples ou doubles.
- Une variable XQuery est définie avec un $ suivi d'un nom, par ex. $bookstore.
- Les commentaires XQuery sont délimités par (: et :), par ex. (: Commentaire XQuery :).

Nous utiliserons le document "books.xml" dans les exemples ci-dessous :

```
<bookstore>
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="children">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="web">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<author>Per Bothner</author>
<author>Kurt Cagle</author>
<author>James Linn</author>
<author>Vaidyanathan Nagarajan</author>
<year>2003</year>
<price>49.99</price>
</book>
<book category="web" cover="paperback">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

Par exemple, les expressions "**If-Then-Else**" sont autorisées dans XQuery.

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="children")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

Remarques sur la syntaxe "**if-then-else**": les parenthèses autour de l'expression **if** sont obligatoires. **else** est obligatoire, mais il peut s'agir simplement **else ()**.

Le résultat de l'exemple ci-dessus sera :

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>XQuery Kick Start</adult>
<adult>Learning XML</adult>
```

Dans XQuery, il existe deux façons de comparer des valeurs :

- Comparaisons générales : =, !=, <, <=, >, >=
- Comparaisons de valeurs : eq, ne, lt, le, gt, ge

La différence entre les deux méthodes de comparaison est indiquée dans l'exemple ci-dessous.

L'expression suivante renvoie true si l'un des attributs q a une valeur supérieure à 10 : `$bookstore//book/@q > 10`

L'expression suivante renvoie true si un seul attribut q est renvoyé par l'expression et que sa valeur est supérieure à 10. Si plusieurs q sont renvoyés, une erreur se produit :
`$bookstore//book/@q gt 10`

6.2.1.1 Expressions XQuery

Une expression XQuery renvoie soit un ensemble de nœuds, une chaîne, un booléen ou un nombre. La version actuelle de XQuery répertorie plusieurs groupes d'expressions XQuery :

Les expressions primaires :

Les expressions primaires sont les primitives de base du langage XQuery. Elles comprennent les littéraux, les références de variables, les constructeurs et les appels de fonctions.

Un **littéral** est une représentation syntaxique directe d'une valeur atomique. XQuery prend en charge deux types de littéraux : les littéraux numériques et les littéraux de chaînes de caractères. Les principaux littéraux numériques de XQuery sont : les entiers (comme 12) de type xs:integer, les décimaux (comme 12.5) de type xs:decimal, et les doubles (125e2) de type xs:double. D'autres types atomiques populaires sont xs:date, xdt:dayTimeDuration.

Une **référence à une variable** dans XQuery est un QName précédé du signe $. Les QName correspondent aux noms d'éléments XML, aux noms d'attributs, etc. Deux références de variables sont équivalentes si leurs noms locaux sont les mêmes et si leurs préfixes d'espace de noms sont liés à la même adresse. Une référence de variable sans préfixe n'est pas dans un espace de noms. XQuery utilise le signe $ pour les variables afin de faire la distinction entre un QName et une variable. Par exemple, dans l'expression $b/titre, b est une variable définie précédemment (liée à certaines séquences), mais titre est un QName défini dans le document XML.

Dans XQuery, un **appel de fonction** se compose d'un QName suivi d'une liste entre parenthèses de zéro ou plusieurs expressions, appelées arguments. Si le QName n'a pas de préfixe d'espace de nom, on considère qu'il se trouve dans l'espace de nom de la fonction par défaut.

Expressions de chemin :

Dans XQuery, les expressions de chemin sont utilisées pour localiser les nœuds dans les données XML. Les expressions de chemin de XQuery sont dérivées de XPath 1.0. Une expression de chemin consiste en une série d'une ou plusieurs étapes, séparées par une barre oblique, /, ou une double barre oblique, //. Chaque étape est évaluée à une séquence de nœuds.

```
doc("bib.xml")/bib/book/author
```

ou

```
doc("bib.xml")//author
```

Expressions de séquence :

Les séquences XQuery sont utilisées pour spécifier une collection ordonnée d'éléments. Ces éléments peuvent être de types similaires ou différents. Une séquence XQuery est créée en utilisant des parenthèses avec des chaînes de caractères entre guillemets ou entre guillemets doubles ou des nombres. Les éléments XML peuvent également être utilisés comme éléments d'une séquence.

Il existe deux façons de créer des éléments de séquence itérés un par un, en utilisant l'index ou la valeur.

Exemple d'expression de séquence XQuery (Index) :

```
let $items := (1,2,3,4,5,6)
let $count := count($items)
return
   <result>
      <count>{$count}</count>
      <items>
      {
         for $item in $items[2]
         return <item>{$item}</item>
      }
      </items>
   </result>
```

Resultat:

```
<result>
   <count>6</count>
   <items>
      <item>2</item>
   </items>
</result>
```

Exemple Expression de séquence XQuery (valeur) :

```
let $items := (1,2,3,4,5,6)
let $count := count($items)
return
   <result>
      <count>{$count}</count>
      <items>
      {
         for $item in $items[. = (1,2,3)]
         return <item>{$item}</item>
      }
      </items>
   </result>
```

Resultat :

```
<result>
   <count>6</count>
   <items>
      <item>1</item>
      <item>2</item>
      <item>3</item>
   </items>
</result>
```

Les opérateurs arithmétiques, de comparaison et logiques :

XQuery prend en charge les **opérateurs arithmétiques** pour l'addition (+), la soustraction (-), la multiplication (*), la division (div et idiv) et le module (mod), dans leurs formes binaires et unaires habituelles. Les opérateurs unaires ont une priorité plus élevée que les opérateurs binaires et les parenthèses doivent être utilisées lorsque cela est nécessaire.

XQuery possède plusieurs ensembles **d'opérateurs de comparaison**, notamment les comparaisons de valeurs (eq, ne, lt, le, gt, ge), les comparaisons générales (=, !=, <, <=, >, >=) et les comparaisons de nœuds (is, <<, >>). Les comparaisons de valeurs et les comparaisons générales sont étroitement liées ; en fait, chaque opérateur de comparaison générale combine un quantificateur existentiel avec un opérateur de comparaison de valeurs correspondant.

Les seuls **opérateurs logiques** de XQuery sont "and" et "or". En outre, XQuery fournit une fonction not() qui prend une séquence générale comme paramètre et renvoie une valeur booléenne.

Les constructeurs d'éléments :

Ils vous permettent de créer de nouveaux nœuds d'éléments avec des noms, des listes d'attributs et des listes d'enfants spécifiés. Le constructeur d'élément XQuery se compose d'une balise de début et d'une balise de fin, entourant une liste facultative d'expressions qui fournissent le contenu de l'élément. Il existe deux types de constructeurs dans XQuery :

- Les constructeurs directs, qui ressemblent à du texte XML. Là, par exemple, le nom du noeud est explicitement donné. Par exemple, l'expression XQuery 1+1={1+1} est évaluée en 1+1=2.
- Les constructeurs calculés, qui ont une nouvelle syntaxe, et permettent de calculer, par exemple, le nom du nœud par une expression. Par exemple l'expression XQuery :

```
element STUDENT {
element SID { 101 }
element FIRST { "Ann" }
element LAST { "Smith" } }
}
```

donne :

```
<STUDENT>
<SID>101</SID>
<FIRST>Ann</FIRST>
<LAST>Smith</LAST>
</STUDENT
```

Les expressions FLWR :

Ces expressions permettent de créer des variables pour les résultats intermédiaires et de transformer des séquences d'éléments sur une base individuelle.

- La liste suivante indique ce qu'ils prennent en compte dans une expression FLWOR /
- F - For - Sélectionne une collection de tous les nœuds.
- L - Let - Place le résultat dans une variable XQuery.
- W - Where - Sélectionne les nœuds spécifiés par la condition.
- - Order by - Classe les nœuds spécifiés en fonction des critères.
- R - Return - Renvoie le résultat final.

Par exemple pour le document XML books.xml. Nous allons utiliser une expression FLWOR pour récupérer les titres des livres dont le prix est supérieur à 30. Le document Xquery suivant contient l'expression de requête à exécuter sur le document XML.

books.xqy :

```
let $books := (doc("books.xml")/books/book)
return <results>
{
   for $x in $books
   where $x/price>30
   order by $x/price
   return $x/title
}
</results>
```

Résultat :

```
<title lang="en">Learn XQuery in 24 hours</title>
<title lang="en">Learn .Net in 24 hours</title>
```

Les expressions de contrôle, telles que IF :

La fonction spéciale if() imite l'opérateur XQuery pour une utilisation dans XPath et XSLT. Les fonctions and() et or() sont également des expressions de contrôle car elles calculent les arguments dans un ordre strict de gauche à droite et peuvent omettre le calcul de certains résultats.

XQuery prend en charge l'instruction conditionnelle **if-then-else** suivante (dèjà abordée dans la section precedente) :

```
if (<expression1>)
then
 <expression2>
else
 <expression3>
```

Expressions quantifiées : SOME et EVERY :

Les fonctions XPath some() et every() sont implémentées pour rendre cette fonctionnalité disponible dans XPath et XSLT. XQuery prend en charge les expressions quantifiées sous la forme suivante :

```
( some | every ) <variable> in <Expression> (,…) satisfies <Expression>
```

Expressions qui testent ou modifient les types de données :

XQuery prend en charge tous les types de données de XML Schema, qu'il s'agisse de types primitifs ou complexes. Les valeurs constantes peuvent être écrites : comme des littéraux (comme string, integer, float), comme des fonctions de constructeur (true(), date("2001-06-07")), comme des castings explicites (cast as xsd:positiveInteger(47)).

Des documents XML Schema arbitraires peuvent être importés dans une XQuery. Un opérateur d'instance de permet la validation à l'exécution de toute valeur relative à un type de données ou à un schéma. Un opérateur de commutation de types permet de se brancher sur la base des types.

6.2.1.2 Fonctions et opérations

La même bibliothèque de fonctions est partagée par XQuery 1.0, XPath 2.0 et XSLT 2.0. Construits sur des expressions XPath, XQuery 1.0 et XPath 2.0 partagent le même modèle de données. Ils prennent également en charge les mêmes fonctions et opérateurs. Cela nous facilite également la définition de nos propres fonctions dans XQuery. Les mêmes types de données que XML Schema 1.0 (XSD) sont également partagés par XQuery.

Une expression XQuery renvoie soit un ensemble de nœuds, une chaîne, un booléen ou un nombre.

Une liste des opérateurs pouvant être utilisés dans les expressions XQuery :

Operateur	Description	Exemple
|	Calcule deux ensembles de nœuds	//book | //cd
+	Addition	6 + 4
-	Soustraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Egal	price=9.80
!=	Pas égal	price!=9.80
<	Inférieur à	price<9.80
<=	Inférieur ou égal à	price<=9.80
>	Supérieur ou égal	price>9.80
>=	Supérieur ou égal à	price>=9.80
or	Ou	price=9.80 or price=9.70
and	Et	price>9.00 and price<9.90
mod	Modulus (reste de la division)	5 mod 2

Une liste des différents types de fonctions XQuery : Fonctions d'accès, fonctions d'erreur et de trace, fonctions numériques, fonctions de chaîne, fonctions AnyURI, fonctions booléennes, fonctions durée/date/heure, fonctions QName, fonctions de nœud, fonctions de séquence, fonctions contextuelles. L'utilisateur peut également définir ses propres fonctions dans XQuery.

Les fonctions sont souvent appelées avec le préfixe fn:, tel que fn:string(). Cependant, puisque fn: est le préfixe par défaut de l'espace de noms, les noms de fonction n'ont pas besoin d'être préfixés lorsqu'ils sont appelés.

Fonctions d'accès :

Nom	Description
fn:node-name(node)	Il est utilisé pour retourner le nom du noeud de l'argument.
fn:nilled(node)	Il est utilisé pour retourner une valeur booléenne indiquant si le nœud argument est nul.
fn:data(item.item,...)	Il est utilisé pour prendre une séquence d'éléments et retourner une séquence de valeurs atomiques.
fn:base-uri() fn:base-uri(node)	Il renvoie la valeur de la propriété base-uri du noeud actuel ou spécifié.
fn:document-uri(node)	Il renvoie la valeur de la propriété document-uri du noeud spécifié.

Fonctions d'erreur et de trace :

Nom	Description
fn:error() fn:error(error) fn:error(error,description) fn:error(error,description,error-object)	Exemple : error(fn:qname('http://example.com/test', 'err:toohigh'), 'error : price is too high') résultat : renvoie http://example.com/test#toohigh et la chaîne "error : price is too high" à l'environnement de traitement externe.
fn:trace(value,label)	il est utilisé pour déboguer les requêtes.

Fonctions de valeurs numériques :

Nom	Description
fn:abs(num)	Elle renvoie la valeur absolue de l'argument Par exemple : abs(3.14) - Résultat : 3,14
fn:ceiling(num)	Elle renvoie le plus petit nombre entier qui est supérieur à l'argument numérique. Par exemple : ceiling(3.14) - Résultat : 4
fn:floor(num)	Elle renvoie le plus grand nombre entier qui n'est pas supérieur à l'argument numérique. Par exemple : floor(3.14) - Résultat : 3
fn:round(num)	Il est utilisé pour arrondir l'argument numérique au nombre entier le plus proche. Par exemple : round(3.14) - Résultat : 3
fn:round-half-to-even()	Exemple : round-half-to-even(0.5) - Résultat : 0 Exemple : round-half-to-even(1.5) - Résultat : 2 Exemple : arrondi demi-pair (2.5) - Résultat : 2

Fonctions de chaîne :

Il y a beaucoup de fonctions de chaîne utilisées dans XQuery mais ici nous presentons une liste des fonctions de manipulation de chaîne couramment utilisées de XQuery :

Nom	Description
string-length($string as xs:string) as xs:integer	Il retourne la longueur de la chaîne.
concat($input as xs:anyatomictype?) as xs:string	Il renvoie la chaîne concaténée en sortie.
string-join($sequence as xs:string*, $delimiter as xs:string) as xs:string	Elle renvoie la combinaison des éléments d'une séquence séparés par un délimiteur.

Fonctions de valeur booléenne :

Nom	Description
fn:boolean(arg)	Elle est utilisée pour retourner une valeur booléenne pour un nombre, une chaîne ou un ensemble de nœuds.
fn:not(arg)	Elle spécifie que l'argument est d'abord réduit à une valeur booléenne en appliquant la fonction Boolean(). Elle renvoie true si la valeur booléenne est false, et false si la valeur booléenne est true. Par exemple : not(true()) - Résultat : false
fn:true()	Elle renvoie la valeur booléenne true. Exemple : true() - Résultat : true
fn:false()	Elle renvoie la valeur booléenne false. Exemple : false() - Résultat : false

Fonction heure et date :

Nom	Description
current-date()	Il est utilisé pour retourner la date actuelle.
current-time()	Il est utilisé pour retourner l'heure actuelle.
current-datetime()	Il est utilisé pour renvoyer à la fois la date et l'heure actuelles.

Exemples d'appels de fonction :

Un appel à une fonction peut apparaître là où une expression peut apparaître. Regardez les exemples ci-dessous :

Exemple 1 : Dans un élément : `<name>{upper-case($booktitle)}</name>`

Exemple 2 : Dans le prédicat d'une expression de chemin :
`doc("books.xml")/bookstore/book[substring(title,1,5)='Harry']`

Exemple 3 : Dans une clause let : `let $name := (substring($booktitle,1,4))`

Fonctions définies par l'utilisateur XQuery :

Les fonctions définies par l'utilisateur peuvent être définies dans la requête ou dans une bibliothèque distincte. Il faut utiliser le mot-clé de la fonction declare. Le nom de la fonction doit être préfixé. Le type de données des paramètres est essentiellement le même que les types de données définis dans XML Schema et le corps de la fonction doit être entouré d'accolades.

Syntaxe :

```
declare function prefix:function_name($parameter as datatype)
as returnDatatype
{
 ...function code here...
};
```

Exemple de fonction définie par l'utilisateur déclarée dans la requête :

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

Voici un exemple d'appel de la fonction ci-dessus :

```
<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>
```

6.2.2 Usages et exemples de fonctions

À l'aide de XQuery, les données hiérarchiques et tabulaires peuvent être récupérées. Il peut être utilisé pour interroger des structures arborescentes et graphiques. XQuery peut être directement utilisé pour rechercher, créer ou interroger des documents Web ou transformer des documents XML. XQuery peut être utilisé pour extraire des informations à utiliser dans un service Web ou générer des rapports de synthèse.

XQuery est également idéal pour les bases de données basées sur XML et les bases de données basées sur des objets. Les bases de données d'objets sont beaucoup plus flexibles et puissantes que les bases de données purement tabulaires.

Cette section fournit des exemples généraux d'utilisation de XQuery.

Document exemple : biblio.xml

```
<bibliothèque>
<livre année="2002">
<titre> Prélude à fondation </titre>
<auteur><nom> Asimov </nom><prenom> Isaac </prénom></auteur> <éditeur>
Pocket </éditeur>
<prix>6</prix>
</livre>
<livre année"1995">
<titre> La marche des millénaires </titre>
auteur nom Asimov </nom><prenom> Isaac </prénom></auteur> auteur nom White
</nom><prénom> Frank </prénom></auteur> <éditeur> Flammarion </éditeur>
<prix>8.45</prix>
```

```
lire>
<livre année="2005"›
<titre> Avant Dune </titre>
auteur nom Anderson </nom>
<prénom> Kevin J. </prénom></auteur>
<auteur><nom> Herbert </nom><prenom> Brian prénom auteur <éditeur> Pocket
</éditeur>
<prix>12.3</prix>
</livre>
/bibliothèque>
```

Différence de séquences de nœuds :

Différences et des concaténations de nœuds. La différence est avec l'opérateur except : Exclusion de la recherche d'un type de noeud.

Exemple : Construction d'un noeud <livre> avec tous les éléments du premier noeud livre du document source sauf les noeuds auteur.

Requête :

```
<livre> Tous les sous-éléments sauf les auteurs:
{ document("biblio.xml")//livre[1]/(* except auteur)}
</livre>
```

Résultat :

```
<livre> Tous les sous-éléments sauf les auteurs:
<titre> Prélude à fondation </titre> <éditeur> Pocket </éditeur>
<prix>6</prix>
</livre>
```

Concaténation de séquences de nœuds :

Concaténation avec la syntaxe (nœud1, noeud2) : Renvoie les noeud1 puis les noeud2 du doc source.

Exemple: Concaténation dans le document résultat des éléments prix et auteur du document source. Different union de noeuds (ordre des éléments different dans les 2 documents).

Requête :

```
<livre> Le prix suivi des auteurs:
{document("biblio.xml")//livre[2]/(prix,auteur)}
</livre>
```

Résultat:

```
<livre> Le prix suivi des auteurs:
<prix>8.45</prix><auteur><nom> Asimov </nom>
<prénom> Isaac </prénom></auteur>
<auteur><nom> white </nom>
<prénom> Frank </prénom></auteur>
</livre>
```

Comparaison de valeurs atomiques :

Opérateurs eq, ne, lt, le, gt, ge.

Exemple : Extraction des nœuds auteur dont le fils nom vaut "White".

Requête:

```
Document("biblio.xml"//livre/auteur[nom eq "White"]
```

Résultat

```
<auteur><nom> White </nom><prénom> Frank
</prénom></auteur>
```

Fonction de Tri

```
Expr1 sort by Expr2 (ascending | descending)
```

Trie les éléments de la séquence retournée par l'expression Expr1 par les valeurs retournées par Expr2.

Exemple : Extraction du fils titre et l'attribut année de tous les noeuds livre du document ordonnée par rapport à la valeur de année.

Requête :

```
<livres> { for $b in document("biblio.xml")//livre
return <livre> { $b/titre, $b/@année } </livre>
sort by(@année) }
</livres>
```

Résultat :

```
<livres>
<livre année="2005"›<titre> Avant Dune </titre></livre>
<livre année="2002"><titre> Prélude à fondation </titre></livre>
<livre année="1995"><titre> La marche des millénaires </titre></livre>
</livres>
```

Exemples de fonctions:

- Transformation noeud -> valeur :xf:string()

Requête :

```
"Le titre du premier livre est",
document("biblio.xml")//livre[1]/xf:string(titre)
```

Résultat :

```
Le titre du premier livre est Prélude à fondation
```

- Index : index of ()

Requête :

```
let $bl := document{ "biblio.xml"}//livre     I
for $b in $bl
return {$b/title {attribute no {index-of($bl, $b)}}}
```

Résultat :

```
<titre no="1"> Prélude à fondation </titre>
<titre no="2"> La marche des millénaires </titre>
<titre no="3"> Avant Dune </titre>
```

6.3 Exercices

À partir du fichier XML data.xml, exprimez les requêtes **XQuery** suivantes :

Exercice 1 :

Sélectionner les propriétaires du pays Algérie. C'est-à-dire, trouver une séquence de tous les éléments propriétaires correspondants. Ordonner le résultat selon les noms des propriétaires.

Exercice 2 :

- Créer une séquence de toutes les propriétés transformées en respectant le modèle suivant :

```
< propriété  idPropriété =" PROPRIÉTÉ-ID ">
    < propriétaire >OWNER-NAME</ propriétaire >
    < adresse > ADRESSE-PROPRIÉTÉ-STRUCTURÉE </ adresse >
</ propriété >
…
```

- Utilisez des constructeurs directs et des requêtes imbriquées.

Exercice 3 :

- Créez une séquence de tous les employés de l'agence :

```
< employé >
    <nom> NOM-EMPLOYÉ-CONCATÉNÉ </nom>
    works at
    <agence id="AGENCE-ID">AGENCE-NOM</agence>
</ employé >
…
```

- Utilisez des constructeurs calculés.

Exercice 4 :

- Créez une séquence d'appartements avec les identifiants des propriétés, les noms des appartements et les niveaux de confort de tous les appartements disponibles :

```
< propriété  > PROPRIÉTÉ-ID </ propriété  >
    < nom > NOM-APPARTEMENT </ nom >
    <comfort> NIVEAU-CONFORT </comfort>
…
```

- Ordonner cette séquence en utilisant : les noms des propriétés dans un ordre décroissant, et puis en utilisant les noms des appartements dans un ordre croissant.

Exercice 5 :

Sélectionner les identifiants de tous les appartements dont le niveau de confort est B ou C, de telle sorte que leur valeur de tarif soit inférieure à la moyenne générale.

Exercice 6 :

- Regrouper les appartements selon leur niveau de confort.
- Trier les groupes de confort créés par ordre alphabétique.
- Inclure uniquement les groupes de confort avec 2 appartements ou plus.
- Ordonner les appartements à l'intérieur de ces groupes en utilisant leurs identifiants.

Exercice 7 :

- Sélectionnez les propriétés ayant le nombre maximum d'appartements annoncés.
- Ordonner le résultat en utilisant les noms de propriété.

6.4 Corrigés des exercices

Dans ce TP, nous utiliserons BaseX. Il y a également eXist. Tous deux sont gratuits et open source. BaseX a été développé initialement par l'Université de Constance en Allemagne et il est maintenant sur GitHub, licence BSD.

BaseX est un gestionnaire de bases de données XML. Il gère des données nativement représentées en XML. Le SGBD BaseX propose une interrogation à l'aide de XQuery (y compris version 3), des modifications à l'aide de XQuery Update, et une interface utilisateur complète. Premièrement, il faut prendre en charge le fichier data.xml à traiter par BaseX. On peut ensuite écrire des requêtes XQuery et lancer l'exécution.

Les requêtes sont exprimées comme suit :

Solution exercice 1 :

```
for $o in fn:doc('data.xml')//propriétaire
where $o/adresse/pays = "Algérie"
order by $o/nom
return $o

for $o in //propriétaire …

for $o in fn:doc('data.xml')//propriétaire[adresse/pays = "Algérie"]
order by $o/nom
return $o

for $o in fn:doc('data.xml')//propriétaire
let $c := $o/adresse/pays
where $c = "Algérie"
order by $o/nom
return $o
```

Solution exercice 2 :

```
for $p in fn:doc('data.xml')//propriété
return
  <propriété>
    { $p/@idPropriété }
    <propriétaire>
      {
         fn:doc('data.xml')//propriétaire[@idPropriétaire =
$p/@propriétaire]/nom/text()
      }
    </propriétaire>
    { $p/adresse }
  </propriété>
… return … <propriété idPropriété="{ $p/@idPropriété }"> …
```

Solution exercice 3 :

```
for $e in fn:doc('data.xml')//agence/employé
let $a := $e/parent::agence
return
  element employé {
    element nom {
      text { fn:concat($e/prénom, " ", $e/nomFamille) }
    },
    text { " travail à " },
    element agence {
      attribute id { $a/@idAgence },
      text { $a/nom }
    }
  }
```

```
… return
  element employé {
    element nom {
      fn:concat($e/prénom, " ", $e/nomFamille)
    },
    " travail à ",
    element agence {
      attribute id { fn:data($a/@idAgence) },
      $a/nom/text()
    }
  }

… return … element { "employé" } { … } …

for $a in fn:doc('data.xml')//agence
for $e in $a/employé
return …

for $a in fn:doc('data.xml')//agence, $e in $a/employé
return …

for $a in fn:doc('data.xml')//agence
return
  for $e in $a/employé
  return …
```

Solution exercice 4 :

```
for $f in fn:doc('data.xml')//appartement
let $pNom := //propriété[@idPropriété = $f/@propriété]/nom
order by $pNom descending, $f/nom
return
  (
    element propriété { data($f/@propriété) },
    $f/nom,
    <comfort>{ data($f/@comfort) }</comfort>
  )

… order by $pNom descending, $f/nom ascending …

… order by $pNom/text() …

… order by //propriété[@idPopriété = $f/refPopriété]/nom descending, $f/nom
…
```

Solution exercice 5 :

```
let $appartements := fn:doc('data.xml')//appartement
let $avg := fn:avg($appartements/tarif)
for $f in $appartements
let $c := $f/@comfort
where ($f/tarif < $avg) and ($c = "B" or $c = "C")
return fn:data($f/@idAppartement)

let $avg := fn:avg(//appartement/tarif)
for $f in //appartement[@comfort = "B" or @comfort = "C"][tarif < $avg]
return fn:data($f/@idAppartement)

let $appartements := …, $avg := … …

… where ($f/tarif < $avg) and ($c = ("B", "C")) …
```

Solution exercice 6 :

```
for $c in fn:distinct-values(fn:doc('data.xml')//appartement/@comfort)
let $appartements := fn:doc('data.xml')//appartement[@comfort = $c]
let $count := fn:count($appartements)
where $count >= 2
order by $c ascending
return
  <group comfort="{ $c }">
    {
      for $f in $appartements
      order by $f/@idAppartement
      return <appartement id="{ $f/@idAppartement }"/>
    }
  </group>

for $c in ("A", "B", "C", "D", "E", "F") …
```

Solution exercice 7 :

```
let $propriétés := fn:doc('data.xml')//propriété
let $values :=
  for $p in $propriétés
  return fn:count(fn:doc('data.xml')//appartement[@propriété =
$p/@idPropriété])
let $max := fn:max($values)
for $p in $propriétés

let $count := fn:count(fn:doc('data.xml')//appartement[@propriété =
$p/@idPropriété])
where $count = $max
order by $p/nom ascending
return
  <propriété id="{ $p/@idPropriété }" appartements="{ $count }"/>

let $propriétés := fn:doc('data.xml')//propriété
let $values :=
  for $p in $propriétés
  let $count := fn:count(fn:doc('data.xml')//appartement[@propriété =
$p/@idPropriété])
  return
    <item>
      <id>{ fn:data($p/@idPropriété) }</id>
      <nom>{ $p/nom/text() }</nom>
      <count>{ $count }</count>
    </item>
let $max := fn:max($values/count)
for $i in $values
where $i/count = $max
order by $i/nom ascending
return
  <propriété id="{ $i/id }" appartements="{ $i/count }"/>
```

Bibliographie

1. J. Friesen, Java XML and JSON : Document Processing for Java SE, Second Edition, Jef f Friesen, Dauphin, MB, Canada, 2019 https://doi.org/10.1007/978-1-4842-4330-5
2. M. Svoboda, P. Čontoš, NPRG036: XML Technologies, summer semester 2021/22, Department of Software Engineering Faculty of Mathematics and Physics, Charles University, consulté en 2022, https://www.ksi.mff.cuni.cz/~svoboda/courses/192-NPRG036/
3. P. Nerzic, Documents et outils XML, version février-mars 2022, IUT de Lannion - dept Informatique (France), consulté en 2022, https://perso.univ-rennes1.fr/pierre.nerzic/XML/
4. Alexandre Brillant, XML Cours et exercices : Modélisation - Schéma - Design patterns - XSLT - XPath - SOAP - XQuery - XSL-FO – SVG. ÉDITIONS EYROLLES (2007)
5. J. P., Balpe. Hperdocuments, hypertextes, hypermédias. Paris, Eyrolles, 1990.P 6.
6. R. Steinmetz, K. N. ahrstedt, Multimedia Applications, Springer-Verlag Berlin Heidelberg 2004.

E. Egyed-Zsigmond, M. Enjalbert, H. Lefleurier, Data for the Web (cours), INSA de Toulouse (France), consulté en 2022 https://open.insa-toulouse.fr/course/view.php?id=258

7. XQuery Tutorial, consulté en 2022, https://www.javatpoint.com/xquery-tutorial
8. XML Tutorial, consulté en 2022, https://www.w3schools.com/xml/default.asp
9. The largest web developer site on the net - Full Web Building Tutorials, consulté en 2022, http://www.w3schools.com/
10. Apprendre avec XMLFacile, consulté en 2022, http://www.xmlfacile.com/
11. S. Crozat, Introduction à XML : principes, syntaxe, schémas et manipulations, 19 février 2016, consulté en 2022, https://stph.scenari-community.org/lo17/xml/
12. Structurez vos données avec XML, consulté en 2022, https://tutoriel-xml.rolandl.fr/