مخبر هندسة الأنظمة المعقدة
LABORATOIRE D'INGÉNIERIE DES SYSTÈMES COMPLEXES

جامعة باجي مختار-عنابة
Université Badji Mokhtar-Annaba

Administration des bases de données avancées

Master I: Ingénierie des Logiciels Complexes (ILC)

Dr Kamilia MENGHOUR
Laboratoire d'Ingénierie des Systèmes Complexes
Université Badji Mokhtar-Annaba
K_menghour@yahoo.fr

Année Universitaire : 2024- 2025

Chapitre 3

Langages de création et d'interrogation des bases de données

Plan du cours

Chapitre 3 : *Langages de création et d'interrogation des bases de données (20%)*

- LDD,
- SQL,
- PHP,
- XML

XML

Extensible Markup Language

XML: Extensible Markup Language

- **XML:** C'est un langage permettant de représenter et structurer des informations à l'aide de balises que chacun peut définir et employer comme il le veut.
- Un ***document XML*** sert alors de vecteur à l'information : c'est une manière *universelle* de représenter des données et leur sens dans un cadre précis.
- Le **XML** permet de structurer les données par des balises (tags).
- Les balises ne sont pas pré-définies comme pour le HTML.
- Séparation de la présentation et du contenu
- **Caractéristique**:
  - La structure d'un document XML est définissable et validable par un schéma.

XML : Exemple

- ***Bibliographie***
  - G. Gardarin, XML : des bases de données aux services web, Dunod, 2003
  - S. Abiteboul, N. Polyzotis, The Data Ring, CIDR, 2007
- ***HTML***

```
<h1>Bibliographie</h1>
<ul>
    <li>G. Gardarin, <i>XML : Des Bases de Données aux Services Web </i>, Dunod, 2003
    <li>S. Abiteboul, N. Polyzotis, <i>The Data Ring</i>, CIDR, 2007
</ul>
```

- ***Base de données relationnelle***

Auteur	Titre	Éditeur	Conférence	Année
G. Gardarin	XML : des bases de données aux services web	Dunod	NULL	2003
S. Abiteboul	The Data Ring	NULL	CIDR	2007

- ***XML***

```
<bibliographie>
    <ouvrage année="2003">
        <auteur>G. Gardarin</auteur>
        <titre>XML : Des Bases de Données aux Services Web</titre>
        <éditeur>Dunod</éditeur>
    </ouvrage>
    <ouvrage année="2007">
        <auteur>S. Abiteboul</auteur>
        <auteur>N. Polyzotis</auteur>
        <titre>The Data Ring</titre>
        <conférence>CIDR</conférence>
    </ouvrage>
</bibliographie>
```

Applications du XML

Le format **XML** est au cœur de nombreux processus actuels :

- Format d'enregistrement de nombreuses applications,
- Échange de données entre serveurs et clients,
- Outils et langages de programmation,
- Bases de données **XML** natives.

- **XML** définit la syntaxe des documents, mais les applications définissent les balises, leur signification et ce qu'elles doivent contenir.
- Des **API** existent dans tous les langages pour lire et écrire des documents **XML**.

XML et les Bases de données

- ***XML*** permet aussi de représenter des données complexes.

- ***Web sémantique*** : c'est un projet qui vise à faire en sorte que toutes les connaissances présentes plus ou moins explicitement dans les pages web puissent devenir accessibles par des mécanismes de recherche unifiés.

  L'un des mécanismes du *Web sémantique* est une base de données appelée ***RDF***. Elle peut être interrogée à l'aide d'un langage de requêtes appelé *SparQL*.

- ***Bases de données XML native*** : les données sont au format ***XML*** et les requêtes sont dans un langage (*XQuery*) permettant de réaliser l'équivalent de *SQL*.

Pourquoi utiliser XML?

- ***XML*** peut stocker et organiser n'importe quel type d'informations sous une forme adaptée à vos besoins.
- En tant que *norme standard* d'échange de données, ***XML*** n'est pas lié à un logiciel en particulier.
- Avec le jeu de caractères standard *Unicode*, ***XML*** prend en charge un nombre impressionnant de systèmes d'écriture (scripts) et de symboles.
- Avec sa syntaxe claire et simple et sa structure sans ambiguïté, ***XML*** est facile à lire et à analyser par les humains et les programmes. Il nécessite une faible courbe d'apprentissage.
- ***XML*** offre de nombreuses façons pour vérifier la qualité d'un document, avec des règles pour la syntaxe, la vérification de liens internes, la comparaison aux modèles de document et le typage des données.
- ***XML*** se combine facilement avec des feuilles de style pour créer des documents formatés dans le style de votre choix.
- On peut utiliser ce langage pour échanger des données entre presque tous les systèmes actuellement utilisés.

XML orienté données et orienté texte

- Lorsque les données sont élaborées par des êtres humains, on dit que les fichiers ***XML*** produits sont ***orientés document***. La réalisation de tels documents impose de se focaliser sur le contenu et non sur la syntaxe du format de document.
- *Un fichier XML orienté document* peut être, par exemple, un livre, un article, un message…
- Lorsque les données sont construites automatiquement par des programmes, on dit que les fichiers ***XML*** sont ***orientés données***. Dans ce type de format, il n'y a pas de représentation facilement utilisable pour l'être humain.
- *Un fichier XML orienté donnée* est, par exemple, un sous-ensemble d'une base de données.
- Il faut noter que l'élaboration des fichiers ***XML*** nécessite des moyens de contrôle et d'édition plus ou moins sophistiqués.

Historique

- ***SGML*** a été introduit en 1986 par *C. Goldfarb*.
- Le langage ***HTML*** a été défini en 1991 par *t. Berners-lee* pour le WEB.
- Ce langage est une version simplifiée de ***SGML***, destinée à une utilisation très ciblée.
- ***XML*** est, en quelque sorte, l'intermédiaire entre ***SGML*** et ***HTML***. Il évite les aspects les plus complexes de **SGML**.
- La version 1.0 de ***XML*** a été publiée en 1998 par le *W3C* (*world wide web*).
- Une redéfinition ***XHTML*** de ***HTML*** 4.0 à travers ***XML*** a été donnée en 1999.
- La version 1.1 de ***XML*** a été publiée en 2004, c'est une mise à jour pour les caractères spéciaux en lien avec unicode.

Langages et technologies à base de XML

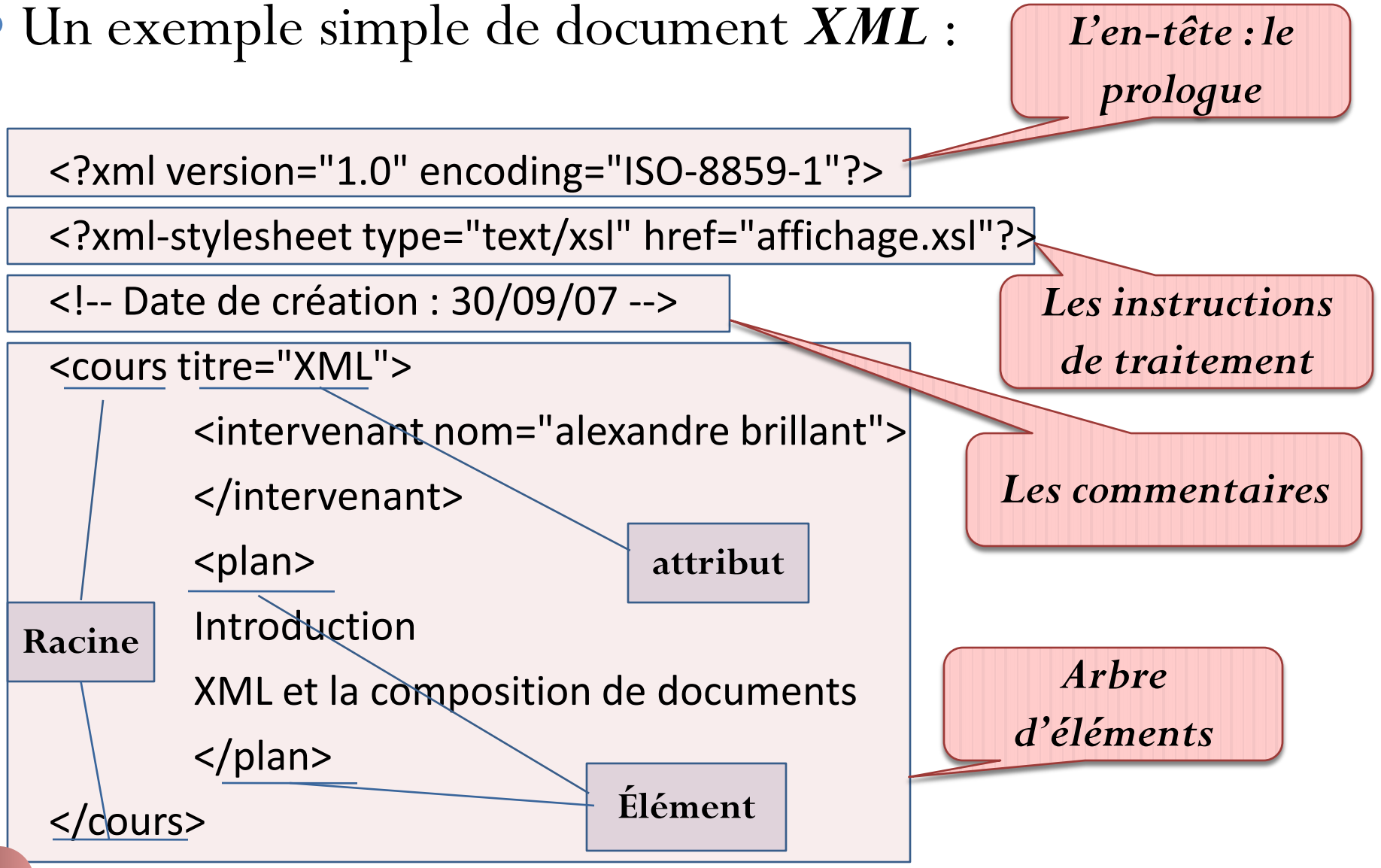
- Les principaux langages qui font partie de l'environnement XML sont:
  - **XLink** et **XPointer** :liens entre documents.
  - **XPath** : un langage d'expressions permettant de sélectionner des éléments dans un document XML.
  - **Xquery**: un langage permettant d'extraire des informations à partir d'un ou plusieurs documents XML et de synthétiser de nouvelles informations à partir de celles extraites. C'est un langage d'interrogation de bases de données et il joue le rôle de SQL pour les documents XML.
  - **Schémas XML**: remplacent les DTD héritées de SGML pour décrire des modèles de documents. Ils sont beaucoup plus souples et beaucoup plus puissants que les DTD.
  - **XSLT**: un langage permettant d'exprimer facilement des transformations complexes entre documents XML.

XML

Structure d'un document XML

Structure d'un document XML

- Un exemple simple de document ***XML*** :



Structure d'un document XML

- Le document ***XML*** se compose de:

➢ ***Le prologue(déclaration :*** Il s'agit de la première ligne d'un document XML servant à donner les caractéristiques globales du document, c'est-à-dire : La version XML et le jeu de caractères employé (*encoding)*.

➢ ***Les instructions de traitement*** (processing instruction ou PI): n'ont pas de rôle lié aux données ou à la structuration du document. Elles servent à donner à l'application qui utilise le document XML des informations. Un cas typique est l'utilisation avec les navigateurs pour effectuer la transformation d'un document XML en document XHTML affichable avec l'instruction : <?xml-stylesheet type="text/xsl" href="affichage.xsl"?>

➢ ***Les commentaires:*** Ce sont les mêmes qu'en HTML. Ils se positionnent n'importe où après le prologue et peuvent figurer sur plusieurs lignes.

Structure d'un document XML

- ***Arbre d'éléments :*** Les éléments gèrent la structuration des données d'un document XML,
  - Un **arbre d'éléments** contenant au moins un élément :l'élément racine.
  - L'**élément racine** contient l'intégralité du document.
  - Chaque ***élément*** possède une ***balise ouvrante*** et une ***balise fermante***
    - <element> : balise ouvrante.
    - </element> : balise fermante.
    - <element/> : balise ouverte et fermée que l'on nomme balise autofermée. C'est l'équivalent de <element></element>.Elle désigne donc un élément vide.
  - L'imbrication des ***balises*** doit être correcte.
  - Chaque élément peut contenir du texte simple, d'autres éléments, ou encore un mélange des deux.
  - Les ***balises ouvrantes*** peuvent contenir des ***attributs*** associés à des valeurs.
  - Un **attribut** est une information supplémentaire attachée à un élément, on parle de métadonnée. L'association de la valeur à l'attribut prend la forme *attribute*='*value*'.

Quelques règles de syntaxe

- Le nom d'un ***élément*** ne peut commencer par un chiffre.
- Si le nom d'un ***élément*** est composé d'un seul caractère il doit être dans la plage [a-zA-Z] (c'est-à-dire une lettre minuscule ou majuscule sans accent) ou _ ou :.
- Avec au moins 2 caractères, le nom d'un ***élément*** peut contenir _, -, . et : plus les caractères alphanumériques (attention, le caractère : est réservé à un usage avec les *espaces de nom*).
- Tous les ***éléments*** *ouverts* doivent être *fermés*.
- Un ***élément*** *père* est toujours fermé après la fermeture des éléments *enfants*.

Voici un *contre-exemple* où l'élément enfant ***b*** est incorrectement fermé après la fermeture de son élément père ***a*** :

<a>.

Les espaces de nom

- Les ***espaces de noms*** ont été introduits en ***XML*** afin de pouvoir mélanger plusieurs vocabulaires au sein d'un même document.
- Un ***espace de noms*** est identifié par un ***URI*** appelé *URI de l'espace de noms*.
- **Idée**: rajouter un préfixe afin de rendre "uniques" et identifiables les noms utilisés
  - On les introduit dans une balise par l'attribut *xmlns: namespace ="URL"*
  - Puis dans le document les balises correspondantes auront la forme *<namespace: nombalise>*

Exemple:

```
<racine xmlns:h="http://www.w3.org/TR/html4/"
        xmlns:f=" http://www.site.com/furniture">
    <h:table>
        <h:tr>
            <h:td>Pommes</h:td>
            <h:td>Bananes</h:td>
        </h:tr>
    </h:table>
    <f:table>
        <f:name>African Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
    </f:table>

</racine>
```

Document Bien Formé et Document Valide

- La validation permet de vérifier la structure et le contenu d'un document XML avant de commencer à le traiter.
- Il y a deux niveaux de correction pour un document XML :
  - ***Un document XML bien formé*** (well formed) respecte les règles syntaxiques d'écriture XML : écriture des balises, imbrication des éléments, entités, etc.
  - ***Un document valide*** respecte des règles supplémentaires sur les noms, attributs et organisation des éléments.
- Comme pour une BD, un document **XML** est souvent accompagné d'information de description, qui peut être spécifiée:
  - Soit par ***DTD***, ***Document Type Definition***, écrite dans un langage de définition de DTD.
  - Soit par ***XML schéma*** , décrit dans un langage appelé Schéma XML

DTD

Document Type Definition

DTD (Document Type Definition)

- Une ***Document Type Definition*** est une liste de règles définies au début d'un document ***XML*** pour permettre sa validation pendant sa lecture. Elle est déclarée par un élément spécial ***DOCTYPE*** juste après le prologue et avant la racine :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE itineraire ... >
<itineraire nom="essai">
    <etape distance="0km">départ</etape>
    <etape distance="1km">tourner à droite</etape>
</itineraire>
```

NB: les **DTD** sont issues de la norme ***SGML*** et n'ont pas la syntaxe XML.

DTD (Document Type Definition)

- Une ***DTD*** peut être :
  - ***Interne*****:** intégrée au document. C'est signalé par un couple [] :

```
<!DOCTYPE itineraire [
… déclaration des éléments
]>
```

Nom de la racine

  - ***Externe:*** dans un autre fichier, signalé par ***SYSTEM*** suivi de l'URL du fichier :

```
<!DOCTYPE itineraire SYSTEM "itineraire.dtd">
```

  - ***Mixte:*** il y a à la fois un fichier et des définitions locales :

```
<!DOCTYPE itineraire SYSTEM "itineraire.dtd" [
…
]>
```

Nom de la racine

Nom du fichier

- Le plus souvent, c'est une DTD externe car identique pour tous les documents à traiter.

DTD (Document Type Definition)

- Une DTD contient des règles comme celles-ci :

  <!ELEMENT itineraire (etape+)>

  <!ATTLIST itineraire nom (#PCDATA)>

  <!ELEMENT etape (#PCDATA)>

  <!ATTLIST etape distance (#PCDATA)>

- Ce sont des règles qui définissent :
  - Des éléments (ELEMENT) : leur nom et le contenu autorisé,
  - Des attributs (ATTLIST) : leur nom et options.

DTD (Document Type Definition)

- ***Exemple:***

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE itineraire [
    <!ELEMENT itineraire (boucle?, etape+, variante*)>
    <!ELEMENT boucle EMPTY>
    <!ELEMENT etape (#PCDATA)>
    <!ELEMENT variante ANY>
]>
<itineraire>
    <boucle/>
    <etape>départ</etape>
    <etape>tourner à droite</etape>
    <variante>
        <etape>départ</etape><etape>tourner à gauche</etape>
    </variante>
</itineraire>
```

DTD (Document Type Definition)

- Le nom présent après le mot-clé **DOCTYPE** indique la racine du document. C'est un élément qui est défini dans la DTD.
- La définition du contenu peut prendre différentes formes :
  - ***EMPTY*** : signifie que l'élément doit être vide,
  - ***ANY*** : signifie que l'élément peut contenir n'importe quels éléments (définis dans la DTD) et textes (leur ordre d'apparition et leur nombre ne seront pas testés),
  - ***(#PCDATA)*** : signifie que l'élément ne contient que des textes
- Les sous-éléments: C'est une liste ordonnée dans laquelle chaque sous-élément peut être suivi d'un joker parmi * + ?
  - ***Opérateur « + » 1 à n fois. exp: A+***
  - ***Opérateur « * » 0 à n fois. exp: A****
  - ***Opérateur « ? » 0 ou 1 fois.***
  - ***Opérateur « \| » Opérateur de choix.(Alternatifs)***

    exp: <!ELEMENT itineraire (boucle?, etape+, variante*)>
  - ***Opérateur « , » opérateur de suite (ou séquence)***

    exp: <!ELEMENT personne(prenom,nom)>
  - ***Opérateur « () » utilisées pour lever les ambiguïtés. exp: (A,B)+***

Les limites des DTD

- les DTD ne sont pas au format XML.
- les DTD ne supportent pas les «espaces de nom».
- le «typage» des données est extrêmement limité.

Schéma XML

Schéma XML

- Les Schémas XML sont une norme W3C pour spécifier le contenu d'un document XML. Ils sont écrits en XML et permettent d'indiquer les conditions de validité beaucoup plus finement.
- **Exemple**:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="reference" type="ElemReference" />
    <xsd:complexType name="ElemReference">
            <xsd:sequence>
                <xsd:element name="titre" type="xsd:string" />
                <xsd:element name="auteur" type="xsd:string" />
                <xsd:element name="ISBN" type="xsd:string" />
            </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Schéma XML

- Pour attribuer un schéma de validation local à un document XML, on peut ajouter un attribut situé dans un *namespace spécifique* :
- **Exemple**:

```
<?xml version="1.0"?>
<reference xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="reference.xsd">
    <titre>Comprendre XSLT</titre>
    <auteur>Bernd Amann et Philippe Rigaux</auteur>
    <ISBN>2-84177-148-2</ISBN>
</reference>
```

Structure générale d'un Schéma XML

- Un schéma est contenu dans un arbre XML de racine <xsd:schema>. Le contenu du schéma définit les éléments qu'on peut trouver dans le document.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="itineraire" type="ElemItineraire" />
    … définition du type ElemItineraire …
</xsd:schema>
```

référence un espace de nom

prologue

déclarations d'éléments, d'attributs et de types

Élément racine

- Il valide le document partiel suivant :

```
<?xml version="1.0"?>
<itineraire>
…
</itineraire>
```

Schéma XML: Définition d'éléments

- Un élément <*nom*>*contenu*</*nom*> du document est défini par un élément <*xsd:element name="nom" type="TypeContenu"*> dans le schéma.
- Dans l'exemple suivant, le type est *xsd:string*, c'est du texte quelconque (équivalent à #PCDATA dans une DTD) :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="message" type="xsd:string"/>
</xsd:schema>
```

- Ce schéma valide le document suivant :

```
<?xml version="1.0"?>
<message> Tout va bien ! </message>
```

Schéma XML: définition de type de données

- Il y a de nombreux types simples prédéfinis, dont :
- Chaîne : – **xsd:string** est le type le plus général
- Date et heure :
  - **xsd:date** correspond à une chaîne au format AAAA-MM-JJ
  - **xsd:time** correspond à HH:MM:SS.s
  - **xsd:datetime** valide AAAA-MM-JJTHH:MM:SS, on doit mettre un T entre la date et l'heure.
- Nombres :
  - **xsd:float, xsd:decimal** valident des nombres réels
  - **xsd:integer** valide des entiers
  - il y a de nombreuses variantes comme **xsd:nonNegativeInteger, xsd:positiveInteger. . .**
- Autres :
  - **xsd:ID** pour une chaîne identifiante**, xsd:IDREF** pour une référence à une telle chaîne
  - **xsd:boolean** permet de n'accepter que true, false, 1 et 0 comme valeurs dans le document.
  - **xsd:base64Binary et xsd:hexBinary** pour des données binaires**.**
  - **xsd:anyURI** pour valider des URI (URL ou URN).

Schéma XML: Restriction sur les types de données

- Lorsque les types ne sont pas suffisamment contraints et risquent de laisser passer des données fausses, on peut rajouter des contraintes. Elles sont appelées facettes (facets).
- ***Restriction d'un type simple:*** consiste à ajouter des contraintes à un type de base:

- ***Exemple***: Définition d'un type simple avec des restrictions

```
<xsd:element name="temperature" type="TypeTemperature" />
<xsd:simpleType name="TypeTemperature">
    <xsd:restriction base="xsd:decimal">
        <xsd:minInclusive value="-30"/>
        <xsd:maxInclusive value="+40.0"/>
    </xsd:restriction>
</xsd:simpleType>
```

Schéma XML: Restriction sur les types de données

- ***Définition de restrictions:***

La structure d'une restriction est :

```
<<xsd:restriction base="type de base">
    <xsd:CONTRAINTE value="PARAMETRE"/>
    ...
</xsd:restriction>
```

- ***Restrictions communes à tous les types***

1. Longueur de la donnée.
2. Expression régulière sur tous les types simples.
3. Bornes sur les entiers et les dates.
4. énumération de valeurs possibles.
5. ***xsd:whiteSpace*** indique ce qu'on doit faire avec les caractères espaces, tabulation et retour à la ligne éventuellement présents dans les données à vérifier :
   - value="preserve" : on les garde tels quels
   - value="replace" : on les remplace par des espaces
   - value="collapse" : on les supprime tous .

Schéma XML: Restriction sur les types de données

1- Longueur de la donnée :

- **xsd:length**,
- **xsd:maxlength**,
- **xsd:minlength.**

Ces contraintes vérifient que le texte présent dans le document a la bonne longueur.

2- Expression régulière étendue sur tous les types simples : C'est une contrainte très utile pour vérifier toutes sortes de données

- **xsd:pattern.**

Cette contrainte permet de n'accepter dans un espace de valeur que les valeurs désignées par des littéraux qui correspondent à une expression régulières spécifique.

```
<xsd:simpleType name="TypeTemperature">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[-+]?[1-9][0-9]?°C"/>
        <xsd:whiteSpace value="collapse"/>
    </xsd:restriction>
</xsd:simpleType>
```

Schéma XML: Restriction sur les types de données

3- Bornes sur les entiers et les dates: Les dates et nombres possèdent quelques contraintes sur la valeur exprimée :

- ***Bornes inférieure et supérieure :***

  – **xsd:minExclusive** et **xsd:minInclusive**
  – **xsd:maxExclusive** et **xsd:maxInclusive**

- En plus de ces facettes, les nombres permettent de vérifier le nombre de chiffres :

  – **xsd:totalDigits** : vérifie le nombre de chiffres total (partie entière et fractionnaire, sans compter le point décimal)
  – **xsd:fractionDigits** : vérifie le nombre de chiffres dans la partie fractionnaire.

Schéma XML: Restriction sur les types de données

4- Énumération de valeurs possibles.

```
<xsd:simpleType name='marques'>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Audi"/>
        <xsd:enumeration value="Golf"/>
        <xsd:enumeration value="BMW"/>
    </xsd:restriction>
</xsd:simpleType>
```

Schéma XML: Types à alternatives

- ***Question*** : Comment valider une donnée qui pourrait être de plusieurs types possibles?
- ***Exemple:*** valider les deux premiers éléments et refuser le troisième

```
<couleur>rouge</couleur>
<couleur>#7FFF00</couleur>
<couleur>02 96 46 93 00</couleur>
```

- ***Solution:*** on crée un « type à alternatives » qui est équivalent à plusieurs possibilités.

```
<xsd:simpleType name="TYPE_ALTERNATIF">
<xsd:union memberTypes="TYPE1 TYPE2 ..."/>
</xsd:simpleType>
```

- Les types possibles sont séparés par un espace.
- ***Exemple: le type couleur***

```
<xsd:simpleType name="TypeCouleurs">
    <xsd:union memberTypes="TypeCouleursNom TypeCouleursHex"/>
</xsd:simpleType>
<xsd:simpleType name="TypeCouleursNom">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[A-Z][a-z]+"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TypeCouleursHex">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="#[0-9A-F]{6}"/>
    </xsd:restriction
</xsd:simpleType>
```

Schéma XML: Données de Type liste

- ***Question*** :Comment contraindre un élément à contenir des données sous forme de liste (séparées par des espaces),
- ***Exemple***

```
<departements>22 29 35 44 56</departements>
```

Solution: à l'aide d'un « type liste » basé sur un type simple. C'est une construction en deux temps :
- il faut le type de base.
- on l'emploie dans une définition de type <xsd:list> :

```
<xsd:simpleType name="TYPE_LISTE">
    <xsd:list itemType="TYPE_BASE"/>
</xsd:simpleType>
```

- ***Exemple: le type liste de départements***

```
<xsd:element name="departements" type="TypeLstDepartements"/>
<xsd:simpleType name="TypeLstDepartements">
    <xsd:list itemType="TypeDepartement"/>
</xsd:simpleType>
<xsd:simpleType name="TypeDepartement">
    <xsd:restriction base="xsd:positiveInteger">
        <xsd:totalDigits value="2"/>
    </xsd:restriction>
</xsd:simpleType>
```

Schéma XML: Type complexe

- ***Question:*** Comment définir un élément dont le contenu peut être d'autres éléments, ainsi que des attributs ?

Exemple: Pour modéliser un élément <*personne*> ayant deux éléments enfants <*prénom*> et <*nom*>, il suffit d'écrire ceci

```
<xsd:element name="personne" type="ElemPersonne"/>
<xsd:complexType name="ElemPersonne">
    <xsd:all>
        <xsd:element name="prénom" type="xsd:string"/>
        <xsd:element name="nom" type="xsd:string"/>
    </xsd:all>
</xsd:complexType>
```

La structure <*xsd:all*> contient une liste d'éléments qui doivent se trouver dans le document à valider. Il y a d'autres structures.

Un <*xsd:complexType*> peut contenir trois sortes d'enfants :

1. <**xsd:sequence**> éléments… <**/xsd:sequence**> : ces éléments doivent arriver dans cet ordre
2. <**xsd:choice**> éléments. . . <**/xsd:choice**> : le document à valider doit contenir l'un des éléments
3. <**xsd:all**> éléments. . . <**/xsd:all**> : le document à valider doit contenir certains de ces éléments et dans l'ordre qu'on veut.

Schéma XML: Type complexe

Exemple de séquence : les éléments <personne>, <numero>, <rue>, <cpostal> et <ville> doivent se suivre dans cet ordre :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="adresse" type="ElemAdresse"/>
    <xsd:complexType name="ElemAdresse">
        <xsd:sequence>
            <xsd:element name="personne" type="ElemPersonne"/>
            <xsd:element name="numero" type="xsd:integer"/>
            <xsd:element name="rue" type="xsd:string"/>
            <xsd:element name="cpostal" type="xsd:integer"/>
            <xsd:element name="ville" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Schéma XML: Type complexe

Exemple de choix : Pour représenter une limite temporelle, par exemple la date de fin, soit on mettra un élément <date_fin> soit un élément <durée> :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="limite" type="ElemLimiteTemps"/>
    <xsd:complexType name="ElemLimiteTemps">
        <xsd:choice>
            <xsd:element name="date_fin" type="xsd:date"/>
            <xsd:element name="durée" type="xsd:positiveInteger"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:schema>
```

Schéma XML: Type complexe

Nombre de répétitions : Dans le cas de la structure <xsd:sequence>, il est possible de spécifier un nombre de répétition pour chaque sous-élément.

```
<xsd:complexType name="ElemPersonne">
    <xsd:sequence>
        <xsd:element name="prénom" type="xsd:string" minOccurs="1" maxOccurs="2"/>
        <xsd:element name="nom" type="xsd:string"  minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

- Par défaut, les nombres de répétitions min et max sont 1.
- Pour enlever une limite sur le nombre maximal, il faut écrire *maxOccurs*="*unbounded*".

Schéma XML: Définition d'attributs

Les attributs se déclarent dans un **<*xsd:complexType*>** :

```
<xsd:complexType name="ElemPersonne">
    ...
    <xsd:attribute name="NOM" type="TYPE" [OPTIONS]/>
</xsd:complexType>
```

- NOM: le nom de l'attribut
- TYPE : le type de l'attribut, ex: *xsd:string* pour un attribut quelconque
- OPTIONS :
    - mettre **use="required"** si l'attribut est obligatoire,
    - mettre **default="valeur"** s'il y a une valeur par défaut.

Schéma XML: contraintes d'intégrité

- ***DTD:*** intégrité référentielle (ID/IDREF): On peut définir des identifiants (attribut de type ID), et des références d'identifiant (de type IDREF).
- ***XML-Schema:*** valeurs uniques, clés et références: On se rapproche du modèle relationnel.
- ***Exemple :***

```
<cinéma>
    <film film_id="f23"> Gladiator </film>
    <film film_id="f12"> Avatar </film>
    ...
    <salle nom="St André des Arts">
        <seance ref_film="f12"> 14:30 </seance>
        <seance ref_film="f23"> 17:00 </seance>
        ...
    </salle>
    <salle nom="Kinorama">
        <seance ref_film="f12"> 15:30 </seance>
        ...
    </salle>
</cinéma>
```

Schéma XML: contraintes d'intégrité

Unicité: valeurs uniques

Exemple: l'attribut *film_id* d'un élément film doit être unique

– Déclaration dans l'élément cinéma du schéma qui contient les valeurs contraintes

```
<xsd:unique name="idfilm">
    <xsd:selector xpath='film' />
    <xsd:field xpath='@film_id' />
</xsd:unique>
```

– *selector*: chemin relatif par rapport à exemple

– *field*: chemin relatif par rapport au *selector*

• Signification: pour tout sous-élément de ***cinéma*** sélectionné par *selector*, la valeur de *field* doit être unique

– Il peut y avoir plusieurs *field* pour un *selector*

Schéma XML: contraintes d'intégrité

- **Clé d'un élément**: similaire à la déclaration d'unicité
  - Nommée, la valeur doit toujours exister et ne peut pas être nulle

```
<xsd:key name='filmclé'>
    <xsd:selector xpath='film' />
    <xsd:field xpath='@film_id' />
</xsd:key>
```

Référence: clé étrangère

- Fait référence à une clé définie
- Dans l'exemple: l'attribut *ref_film* de *seance* est une clé étrangère
  - Définie dans l'élément *salle*

```
<xsd:keyref name='filmref' refer='filmclé'>
    <xsd:selector xpath='seance' />
    <xsd:field xpath='@ref_film' />
</xsd:keyref>
```

Merci de votre attention

Des Questions

K_menghour@yahoo.fr