LABD

Master Info M1 2014-2015

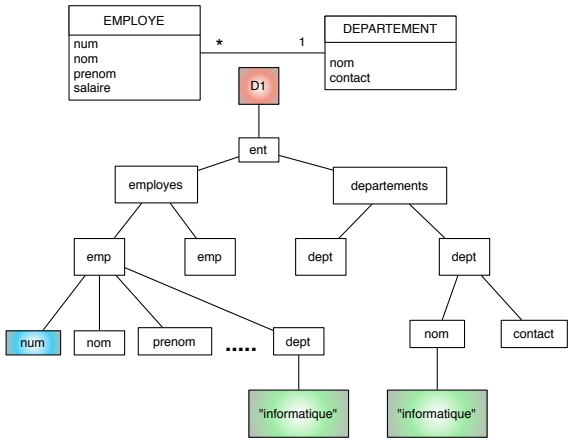
Cours 4 : Contraintes d'intégrité, espaces de noms

Contraintes d'intégrité en XML-Schema

- Modèle relationnel :
  - Contrainte d'unicité `UNIQUE`, de clef primaire (unicité et existence) `PRIMARY KEY`, de clef étrangère `FOREIGN KEY` ... `REFERENCES` ....
  - Une clef primaire est définie pour une relation, et elle composée d'un ou plusieurs attributs.
  - Une clef étrangère fait référence à des attributs d'une relation précise.
- Avec une DTD, on peut définir des identifiants (attribut de type `ID`), et des références d'identifiant (de type `IDREF`). L'existence ou non est définie en choisissant `IMPLIED` ou `REQUIRED`. Mais
  - les références ne sont pas typées (on ne sait pas à quel type de nœud on fait référence)
  - un identifiant est global au document
- XML-Schema : on se rapproche du modèle relationnel

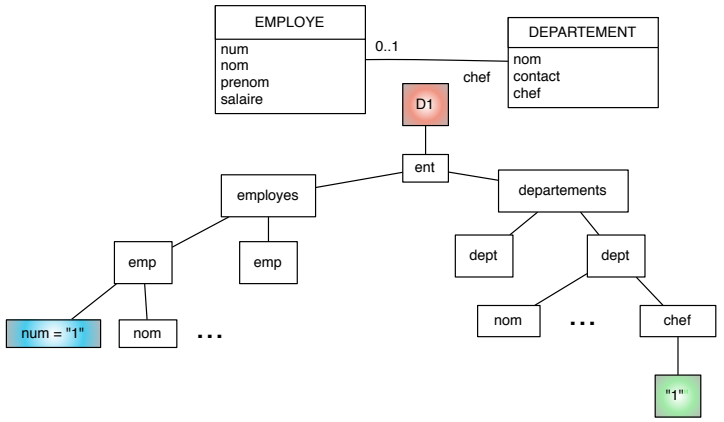
Exemple

Tout employé appartient à un département :



Exemple (suite)

Un département a au plus un chef qui est employé par l'entreprise



Exemple (suite)

On déclare un type pour un département, et pour une séquence de départements

```
<xsd:complexType name="TypeDept">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="contact" type="xsd:string"/>
    <xsd:element name="chef" type="xsd:int" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SeqDept">
  <xsd:sequence>
    <xsd:element name="dept" type="TypeDept" maxOccurs="unbounded"
        minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Exemple (suite)

Même chose pour les employés.

```
<xsd:complexType name="TypeEmploye">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="prenom" type="xsd:string"/>
    <xsd:element name="salaire" type="xsd:decimal"/>
  <xsd:element name="dept" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="num" type="xsd:int"/>
</xsd:complexType>
```

Exemple (suite)

```
<xsd:complexType name="SeqEmploye">
  <xsd:sequence>
    <xsd:element name="emp"
                 type="TypeEmploye"
                 maxOccurs="unbounded"
                 minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Exemple (suite)

Déclaration des éléments `employes` et `departements` avec les clefs primaires pour les éléments `emp` et `dept`.

```
<xsd:element name="employes" type="SeqEmploye">
  <xsd:key name="clefEmp">
    <xsd:selector xpath="emp"/>
    <xsd:field xpath="@num"/>
  </xsd:key>
</xsd:element>

<xsd:element name="departements" type="SeqDept">
  <xsd:key name="clefDept">
    <xsd:selector xpath="dept"/>
    <xsd:field xpath="nom"/>
  </xsd:key>
</xsd:element>
```

Exemple (suite)

L'élément racine du document avec les clefs étrangères.

```
<xsd:element name="ent">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="employes"/>
      <xsd:element ref="departements"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:keyref name="refDept" refer="clefDept">
    <xsd:selector xpath="employes/emp"/>
    <xsd:field xpath="dept"/>
  </xsd:keyref>
  <xsd:keyref name="refEmp" refer="clefEmp">
    <xsd:selector xpath="departements/dept"/>
    <xsd:field xpath="chef"/>
  </xsd:keyref>
</xsd:element>
```

Explications

- L'endroit où l'on déclare une contrainte détermine la zone où elle doit être vérifiée. En effet, la contrainte s'applique à l'intérieur de l'élément "contexte" de cette contrainte.
- Le `selector` détermine pour quel élément c'est une clef. C'est un chemin XPath qui désigne un ensemble de noeuds (appelés noeuds cibles) contenus dans l'élément contexte de la contrainte.
- Les `field` qui suivent donnent les composants (attributs ou éléments) de la clef. Chaque composant désigne un noeud unique (élément ou attribut), par rapport à 1 noeud cible désigné par le `selector`. De plus, chaque composant doit être de type simple.

Explications

- La syntaxe des chemins XPath est réduite, voir la norme pour plus de précision.
- Dans une clef étrangère, l'attribut `refer` indique à quelle clef primaire elle fait référence.
- Pour définir une contrainte d'unicité, remplacer `xsd:key` par `xsd:unique`.

Espaces de noms

Motivations

- mélanger différents vocabulaires et éviter les conflits de nom
- modularité, réutilisation
- exporter les définitions d'un schéma

➡ utilisation de noms qualifiés (= préfixés) pour les éléments et les attributs.

Utiliser des espaces de noms

Identification d'un espace de noms

Un espace de noms est identifié par une adresse IRI, (Internationalized Resource Identifier)

- IRI = extension des URI permettant l'utilisation de caractères internationaux (par ex. UTF-8) dans l'adresse elle-même.

```
http://www.exemple.org/clés
```

- URI = URN ou URL
  - URN : `um:isbn-0-395-36341-1`
  - URL : `ftp://ftpperso.free.fr/pxml`

Remarques

- Le W3C déconseille l'usage d'une IRI relative comme identifiant d'espace de nom
- L'analyse d'un identifiant d'espace de nom tient compte de la casse et ne prend pas en compte la résolution de l'IRI. Tous les exemples suivants représentent des identifiants différents :

```
http://www.Example.org/wine
http://www.example.org/Wine
http://www.example.org/rosé
http://www.example.org/ros%c3%a9
http://www.example.org/ros%c3%A9
```

Finalement en pratique

Identifiant d'espace de nom = URL absolue avec des caractères ASCII.

```
http://www.w3.org/XML/1998/namespace

http://www.w3.org/1999/xhtml

http://www.w3.org/2001/XMLSchema

http://www.w3.org/2001/XMLSchema-instance

http://www.w3.org/1999/XSL/Transform

http://xml.insee.fr/schema/

http://fil.univ-lille1.fr/miage-fa-fc
```

Déclaration d'un espace de noms

Le préfixe qui désigne un espace de noms doit avoir été déclaré, grâce à un pseudo attribut qui commence par `xmlns:`

- Les préfixes `xml` et `xmlns` sont réservés.
- La déclaration se fait dans la balise ouvrante d'un élément
- Lorsqu'on déclare un espace de noms, le préfixe est applicable dès la balise ouvrante où se fait la déclaration, et pour tout le contenu de cet élément, sauf si le même préfixe est utilisé plus bas pour un autre espace de noms.
- L'utilisation du préfixe pour un élément (ou attribut) indique que cet élément (ou attribut) appartient à l'espace de noms associé au préfixe. (nom qualifié)
- On peut déclarer plusieurs espaces de noms dans une même balise ouvrante.

Déclaration d'un espace de noms

- La déclaration se fait dans la balise ouvrante d'un élément
- Lorsqu'on déclare un espace de noms, le préfixe est applicable dès la balise ouvrante où se fait la déclaration, et pour tout le contenu de cet élément.

```
<x xmlns:edi="http://ecom.exple.org/schema">
  .....
</x>
```

Déclaration d'un espace de noms

- La déclaration se fait dans la balise ouvrante d'un élément
- Lorsqu'on déclare un espace de noms, le préfixe est applicable dès la balise ouvrante où se fait la déclaration, et pour tout le contenu de cet élément.

```
<edi:price xmlns:edi="http://ecom.exple.org/sch" units="Euro">
  32.18
</edi:price>
```

Déclaration d'un espace de noms

- La déclaration se fait dans la balise ouvrante d'un élément
- Lorsqu'on déclare un espace de noms, le préfixe est applicable dès la balise ouvrante où se fait la déclaration, et pour tout le contenu de cet élément.

```
<x xmlns:edi="http://ecom.exple.org/schema">
  <lineItem edi:taxClass="exempt">
    Baby food
  </lineItem>
</x>
```

Déclaration d'un espace de noms

- La déclaration se fait dans la balise ouvrante d'un élément
- Lorsqu'on déclare un espace de noms, le préfixe est applicable dès la balise ouvrante où se fait la déclaration, et pour tout le contenu de cet élément, sauf si le même préfixe est utilisé plus bas pour un autre espace de noms.

```
<x xmlns:edi="http://ecom.exple.org/sch1">
  <edi:a>
     <edi:b xmlns:edi="http://ecom.exple.org/sch2">
        <edi:a>
           <edi:c/>
        </edi:a>
     </edi:b>
     <edi:b/>
  </edi:a>
</x>
```

Déclaration d'un espace de noms

On peut déclarer plusieurs espaces de noms dans une même balise ouvrante.

```
<bk:book   xmlns:bk="urn:loc.gov:books"
           xmlns:isbn="urn:ISBN:0-395-36341-6">
  <bk:title>Cheaper by the Dozen</bk:title>
  <isbn:number>1568491379</isbn:number>
</bk:book>
```

Espace de noms par défaut

- Si on utilise l'attribut `xmlns` (sans `:`), on définit alors un espace de noms par défaut, pour lequel il n'existe pas de préfixe associé. L'espace de nom par défaut ne s'applique pas aux attributs.

```
<?xml version="1.1"?>
<!-- elements are in the HTML namespace, by default -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Frobnostication</title></head>
  <body>
    <p>
      Moved to <a href="http://frob.example.com">here</a>.
    </p>
  </body>
</html>
```

Espace de noms par défaut

- Si on utilise l'attribut `xmlns` (sans `:`), on définit alors un espace de noms par défaut, pour lequel il n'existe pas de préfixe associé.

```
<?xml version="1.1"?>
<!-- initially, the default namespace is "books" -->
<book xmlns="urn:loc.gov:books"
      xmlns:isbn="urn:ISBN:0-395-36341-6">
  <title>Cheaper by the Dozen </title>
  <isbn:number>1568491379</isbn:number>
  <notes>
  <!-- make HTML the default namespace for some commentary -->
    <p xmlns="http://www.w3.org/1999/xhtml">
      This is a <i>funny</i> book!
    </p>
  </notes>
</book>
```

Espace de noms par défaut

- Il faut, en général, réserver l'espace de noms par défaut à l'espace de noms le plus utilisé.
- Tant que l'espace de noms par défaut n'a pas été spécifié, les éléments dont le nom n'est pas qualifié ne font partie d'aucun espace de noms. Leur propriété espace de noms n'a pas de valeur.
- Il est possible de revenir à l'espace de noms par défaut non spécifié en affectant la chaîne vide à l'attribut `xmlns`.
- Les attributs peuvent également avoir des noms qualifiés formés d'un préfixe et d'un nom local. Ils font alors partie de l'espace de noms auquel est associé le préfixe.
- Les attributs dont le nom n'est pas qualifié ne font jamais partie de l'espace de noms par défaut. Cette règle s'applique que l'espace de noms par défaut soit spécifié ou non.

Exercice

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

<exercice xmlns:pre="http://A">
  <pre:niveau xmlns:pre="http://D" xmlns="http://A">
     <out:garage xmlns:out="http://C">
        <pre:alcove/>
        <entree xmlns:pre="http://E">
           <pre:cuisine/>
        </entree>
     </out:garage>
  </pre:niveau>
  <def xmlns="http://B">
     <pre:figue>
        <levain/>
     </pre:figue>
  </def>
</exercice>
```

exercice ->
niveau ->
garage ->
alcove ->
entree ->
cuisine ->
def ->
figue ->
levain ->

Créer des espaces de noms

Exporter un espace de noms

attribut `targetNameSpace`

- Pour exporter (créer) un espace de nom dans un schéma, on utilise l'attribut `targetNameSpace` de la balise xsd:schema

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
      "http://www.w3.org/2001/XMLSchema
          http://www.w3.org/2001/XMLSchema.xsd"
  targetNamespace="http://fil.univ-lille1.fr/miage-fa-fc">
   <!-- description de la miage fa fc -->
   ...
```

Exporter un espace de noms

```
<?xml version="1.0" encoding="utf-8"?>
<miage-fa-fc
  xmlns="http://fil.univ-lille1.fr/miage-fa-fc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://fil.univ-lille1.fr/miage-fa-fc http://
www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/schemas/miage-
fa-fc.xsd"
    annee="2011">
  <creneaux>
    <creneau>
      <trimestre>T1</trimestre>
        <jour>lundi</jour>
        <de>09:00:00</de>
        <a>12:00:00</a>
        <salle>M5-A2</salle>
        ...
```

Espaces de noms et schémas

attribut `elementFormDefault`

Quand on exporte un espace de noms,

- Les déclarations globales appartiennent à l'espace de nom
- Les déclarations locales n'appartiennent pas à l'espace de nom, sauf si on ajoute l'attribut `elementFormDefault="qualified"`
- on dispose aussi de l'attribut `attributeFormDefault`

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                       http://www.w3.org/2001/XMLSchema.xsd"
   xmlns="http://fil.univ-lille1.fr/miage-fa-fc"
   targetNamespace="http://fil.univ-lille1.fr/miage-fa-fc"
   elementFormDefault="qualified">
  <!-- description de la miage fa fc -->
```

Inclusions de schémas

On peut assembler plusieurs composants de schémas (définitions de types, déclarations d'éléments, ...), provenant de plusieurs documents.

- élément `include` qui permet d'inclure les définitions provenant d'autres schémas mais pas de plusieurs espaces de noms.

- Les schémas inclus doivent avoir
  1. soit le même espace de noms cible que le document qui les inclut
  2. soit pas d'espace de noms, dans ce cas, c'est l'espace de noms du schéma qui inclut tous les autres qui est pris en compte.

Exemple d'inclusion sans espace de noms cible

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="dept.xsd"/>
  <xsd:include schemaLocation="emp.xsd"/>
  <xsd:element name="ent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="employes"/>
        <xsd:element ref="departements"/>
      </xsd:sequence>
    </xsd:complexType>
  ... les clefs étrangères ...
  </xsd:element>
</xsd:schema>
```

- `dept.xsd` et `emp.xsd` sont des fichiers dans le même répertoire.
- `dept.xsd` (resp. `emp.xsd`) contient les déclarations de l'élément `departements`(resp. `employes`) et de tous ses sous-éléments.

Exemple d'instance du schéma précédent

```
<?xml version="1.0" encoding="utf-8"?>
<ent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="entreprise.xsd">
  <employes>
    <emp num="1">
      <nom>toto</nom><prenom>jules</prenom>
      <salaire>3452</salaire>
      <dept>informatique</dept>
    </emp>
  </employes>
  <departements>
    <dept>
      <nom>informatique</nom>
      <contact>Mme Machin 45-76-77-09-54</contact>
      <chef>1</chef>
    </dept>
  </departements>
</ent>
```

Exemple d'inclusion avec espace de noms cible

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.lifl.fr/~yroos/schema"
      targetNamespace="http://www.lifl.fr/~yroos/schema"
      elementFormDefault="qualified"
>
  <xsd:include schemaLocation="dept.xsd"/>
  <xsd:include schemaLocation="emp.xsd"/>

  <xsd:element name="ent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="employes"/>
        <xsd:element ref="departements"/>
      </xsd:sequence>
    </xsd:complexType>
    ... et les clefs étrangères ...
  </xsd:element>
</xsd:schema>
```

Instance du schéma précédent

```
<?xml version="1.0" encoding="utf-8"?>
<ent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.lifl.fr/~yroos/schema"
      xsi:schemaLocation="http://www.lifl.fr/~yroos/schema
      http://saxo.lifl.fr/~yroos/schema/entreprise.xsd">
  <employes>
    <emp num="1">
      <nom>toto</nom> <prenom>jules</prenom>
      <salaire>3452</salaire> <dept>informatique</dept>
    </emp>
  </employes>
  <departements>
    <dept>
      <nom>informatique</nom>
      <contact>Mme Machin 45-76-77-09-54</contact>
      <chef>1</chef>
    </dept>
  </departements>
</ent>
```

Importation de schémas

- Un schéma est associé à un espace de noms cible
- L'élément `import` permet de faire référence à des composants d'un schéma qui appartient à un autre espace de noms que le schéma dans lequel on fait référence à ces composants.
- Dans l'exemple qui suit, on utilise un composant du schéma de XHTML pour notre propre schéma.

Exemple d'importation de schémas

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.lifl.fr/~yroos/schema"
      xmlns:art="http://www.lifl.fr/~yroos/schema"
      xmlns:html="http://www.w3.org/1999/xhtml"
      elementFormDefault="qualified"
>


  <xsd:import namespace="http://www.w3.org/1999/xhtml"
  schemaLocation=
   "http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd"/>
...
```

Exemple d'importation de schémas

```
...
 <xsd:element name="dansRevue">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="auteur" maxOccurs="unbounded"
             type="xsd:string"/>
       <xsd:element name="revue" type="xsd:string"/>
       <xsd:element name="titre" type="xsd:string"/>
       <xsd:element minOccurs="0" name="resume"
        type="html:Block"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Exemple d'instance de ce schéma

```
<?xml version="1.0" encoding="utf-8"?>
<bibliographie xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.lifl.fr/~yroos/schema"
    xmlns:html="http://www.w3.org/1999/xhtml"
    xsi:schemaLocation="http://www.lifl.fr/~yroos/schema
    http://saxo.lifl.fr/~yroos/schema/articles.xsd"
>
  <dansRevue>
    <auteur>Tryphon Tournesol</auteur>
    <revue>Revue de Physique</revue>
    <titre>Ma machine à voyager dans le temps</titre>
    <resume><html:div>et patati
    <html:br/>et patata</html:div></resume>
  </dansRevue>
</bibliographie>
```

En conclusion

Bonne pratique

L'utilisation des espaces de noms peut parfois être un peu compliquée. Quand un espace de nom est très clairement majoritaire dans le document XML (que ce soit une instance ou un schéma), on le définit comme espace de nom par défaut, sinon une bonne pratique est de lier (dans un premier temps) tous les espaces de noms à des préfixes et de ne définir un espace de nom par défaut que lorsqu'il y en a un qui se détache (c.a.d. quand on tape majoritairement toujours le même préfixe)

Compléments sur XPath

Compléments sur XPath

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Paul</prenom>
      <age>5</age>
      <poids>10</poids>
```

Instruction conditionnelle `if`

```
"if" "(" Expr ")" "then" Expr "else" Expr
```

Exemple :

```
if (count(//femme)> count(//homme)) then
  //homme/prenom
else
  //femme/prenom
```

Expression quantifiée

C'est une expression booléenne dont la syntaxe est :

```
("some" | "every") "$"VarName
"in" Expr ("," "$"VarName "in" Expr)*
"satisfies" Expr
```

Exemple :

```
some $x in //femme, $y in //homme satisfies $x/age + $y/age = 88
every $x in //femme, $y in //homme satisfies $x/age + $y/age = 88
```

La première expression vaut vrai, la seconde vaut faux.

La boucle `for`

```
"for" "$"VarName "in" Expr ("," "$"VarName "in" Expr)*
"return" Expr
```

Exemple :

```
for $p in /famille/*, $f in /famille/*[$p/@id eq pere]
return ($p/prenom , $f/prenom)
```

Composabilité / transparence référentielle

Exemple :

```
if (some $x in //*,  $y in //* satisfies $x/@id eq $y/pere)
then
  for $p in /famille/*, $f in /famille/*[$p/@id eq pere]
  return ($p/prenom , $f/prenom)
else <result>pas de résultat</result>
```

Composabilité / transparence référentielle

Exemple :

```
if (for $p in /famille/*, $f in /famille/*[$p/@id eq pere]
     return ($p/prenom , $f/prenom)) then
  for $p in /famille/*, $f in /famille/*[$p/@id eq pere]
  return ($p/prenom , $f/prenom)
else <result>pas de résultat</result>
```

Composabilité / transparence référentielle

Exemple :

```
for $n in
   for $p in /famille/*, $f in /famille/*[$p/@id eq pere]
   return ($p/prenom , $f/prenom)
return $n/../@id
```

Typage

XPATH 2.0 est typé!

- `xs = http://www.w3.org/2001/XMLSchema`
- `fn = http://www.w3.org/2005/xpath-functions`

Tout item a une valeur typée et une valeur textuelle.

La valeur typée d'un item est une séquence de valeurs atomiques qui peuvent être extraites avec la fonction `fn:data`

La valeur textuelle d'un noeud est une chaîne de caractères qui peut être extraite avec la fonction `fn:string`

Typage des séquences

Les valeurs typées s'appuient sur les types prédéfinis de XML-Schema. S'y ajoutent des informations de types spécifiques aux nœuds ou aux séquences.

`xs:integer`	`empty-sequence()`
`xs:decimal`	`document-node()`
`xs:boolean`	`item()`
`xs:string`	`node()`
`xs:date`	`element()`
`xs:time`	`element(`*`name`*`)`
`xs:dateTime`	`attribute()`
`...`	`+ , * , ?`

Typage des séquences

Type d'une séquence
=
séquence des types des éléments de la séquence

Quelques exemples :

- `xs:date` type d'une séquence contenant un seul item de type `date`
- `attribute()?` type d'une séquence contenant 0 ou 1 noeud attribut
- `element()` type d'une sequence contenant 1 élément quelconque
- `element(prenom)*` type d'une sequence contenant un nombre arbitraire d'éléments de nom `prenom`
- `node()*` type d'une sequence contenant un nombre arbitraire de noeuds quelconques
- `item()+` type d'une sequence contenant un nombre arbitraire non nul de noeuds ou de valeurs atomiques

Plein de fonctions!

http://www.mulberrytech.com/quickref/functions.pdf

Exemples:

```
current-date() as xs:date
current-time() as xs:time
avg(xs:anyAtomicType*) as xs:anyAtomicType
max(xs:anyAtomicType*) as xs:anyAtomicType?
sum(xs:anyAtomicType*) as xs:anyAtomicType
node-name(node()?) as xs:QName?
distinct-values(xs:anyAtomicType*) as xs:anyAtomicType*
doc(xs:string?) as document-node?
(node()) << (node()) as xs:boolean
(node()) >> (node()) as xs:boolean
```

Typage : opérateurs

```
instance of
    ‘1’ instance of xs:string
    1.0 instance of xs:decimal
    true() instance of xs:boolean
    '1' instance of xs:decimal


cast as
    10.1 cast as xs:integer
    '1.2' cast as xs:decimal
    '1' cast as xs:boolean


castable as
    'x1.2' castable as xs:decimal
    '12' castable as xs:integer
```

Typage : opérateurs

```
is
    //*[@id=’4’] is //femme[prenom=‘Marie’]
    //*[@id=’4’]/pere is //*[@id=‘3’]/pere
```

```
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
```