

Cours 7

XML et les BD

BDA
Master - 2019- 2020

XML: langage de balisage extensible

OUARED Abdelkader

Plan

1. Pourquoi la BD et XML
2. Historique
3. Modèle de données
4. XML Validation (**DTD**, XML Schema,..)
5. Produits
6. Langage de requêtes
7. Conclusion

Pourquoi XML pris en charge par la base de données?

- **Données:** Structurées, semi-structurées et non structurées,
- Web et BD
 - la structuration est faible, peut être imbriquée.
 - plutôt orienté documentaire
- Intégrer la communauté de la BD et la GED (gestion électronique des docs)
- Doc XML présente une vitesse de lecture rapide
- Plusieurs domaines d'application peuvent être modélisés plus facilement en XML

XML s'impose d'une ...

- nécessité de stocker les documents XML
- nécessité de pouvoir interroger ces documents
- évolution ou révolution ?

Oracle XML DB: fusion entre XML & DBR

- Est une fonctionnalité de stockage et d'extraction de données XML
- Propose des méthodes d'accès standard pour parcourir et interroger des données XML
- Est une **fusion** entre **XML** et la technologie des **BDRs**

Exemple: Déclarer un **XMLType**

– pour une colonne dans une table relationnelle :

```
CREATE TABLE emp_resumes (  
  employee_id  NUMBER(6) PRIMARY KEY,  
  resume       XMLType);
```

– pour une table relationnelle objet :

```
CREATE TABLE emp_xmlresumes OF XMLType;
```

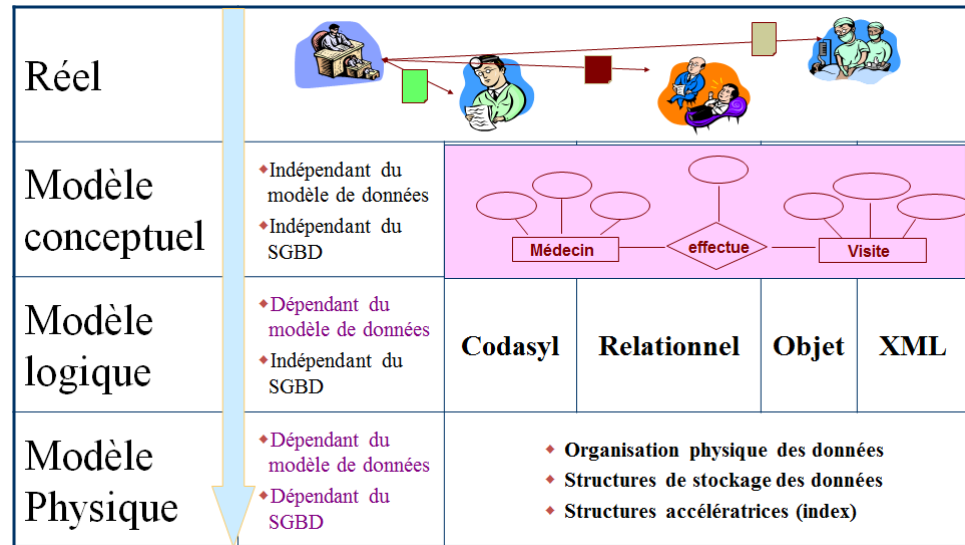
Le **XMLType** est une colonne virtuelle:

- qui encapsule un type de colonne **CLOB** réel
- qui permet d'indiquer les caractéristiques de stockage **CLOB** avec une clause **STORE AS**

```
CREATE TABLE emp_resumes_clob (  
  employee_id  NUMBER(6) PRIMARY KEY,  
  resume       XMLType)  
XMLType COLUMN resume  
STORE AS CLOB (TABLESPACE data01  
               STORAGE (INITIAL 4096 NEXT 4096)  
               CHUNK 4096 NOCACHE LOGGING);
```

XML : A Quel Niveau ?

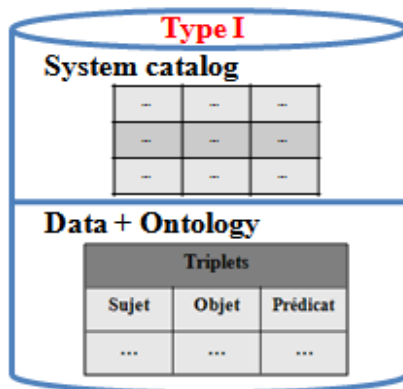
- XML au niveau Logique
- XML au niveau Technique



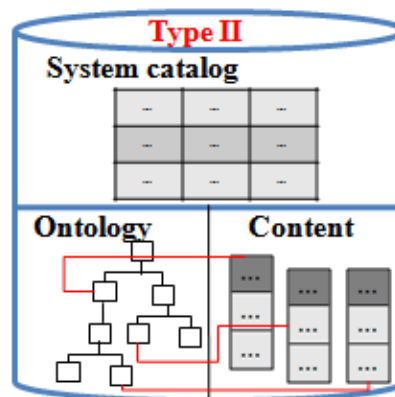
■ BD Sémantique

- **Ontologie**: donner une définition claire d'un domaine
- → Exemple (OWL (Fichier XML) Matérialise l'Ontologie)

Type I:
3store, Oracle ...



Type II:
Sesame, IBM Sor ...



ARCHITECTURE

Historique

- ❑ **SGML** 1986, pour baliser un document
 - Origine c'est industrie
 - Balises trop lourdes
- ❑ **HTML** 1989,
 - Limites HTML
- ❑ **XML** 1998 (markup language,)
 - XML: à l'intérieur stocke les données + informations de contenu (leur structure)
 - indexation est compliquée ☹
 - Pb des gestions des doublons (intégrité des données) ☹

XML Characteristics

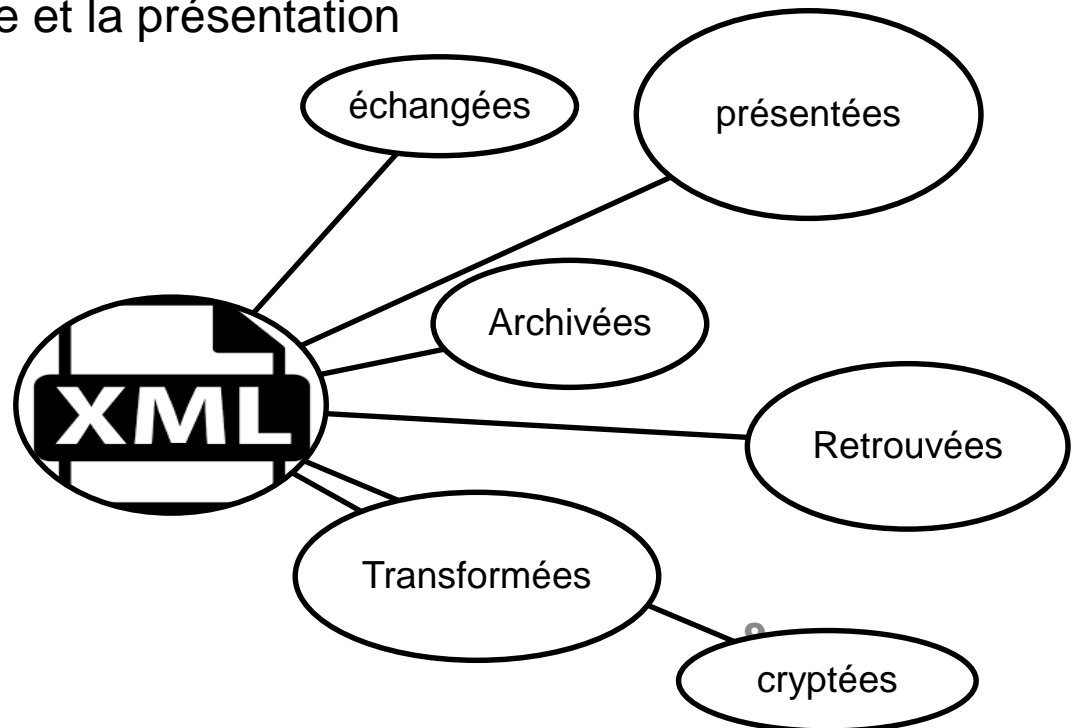
■ XML Characteristics

- XML is a markup language, like HTML
- XML primarily consists of tags, attributes and data
- XML tags are not pre-defined like HTML
- XML tags are user-defined
- While **HTML defines** the layout for data, **XML describes** the data
- XML documents need to be well formed
- It is written in plain text format
- It is platform independent, either hardware or software
- It works well with applications
- **It's a W3C standard**, which makes it universal

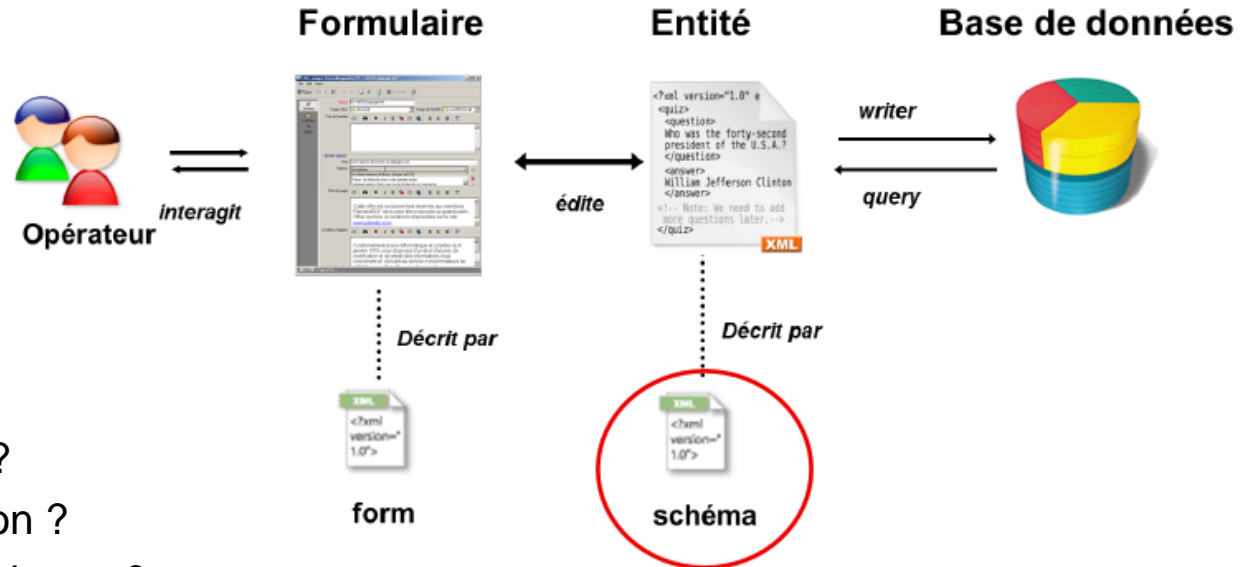
Exemples d'utilisation

- Je veux véhiculer la BD et les documents ➡ j'ai un seul fichier XML
- Il y a une sorte de sémantique dans XML ➡ Table vs Table BD dans les SRIs
 - *Annotation, définition, tag*
- Fichier XML de Config des différents systèmes ➡ XML est un standard
- Utilisable entre des machines hétérogènes
- Standard d'échange de données universel
- Séparation entre la structure et la présentation

Information



XML et BD relationnelles



Données semi structurées

- Quel modèle de données ?
- Quel langage d'interrogation ?
- Quelle intégration avec l'existant ?

SQL	XSLT, XPATH, XQUERY
Modèle relationnel	Modèle XPATH
JDBC/ODBC/SQL-CLI	DOM et SAX API
Bases de données relationnelles	Document XML

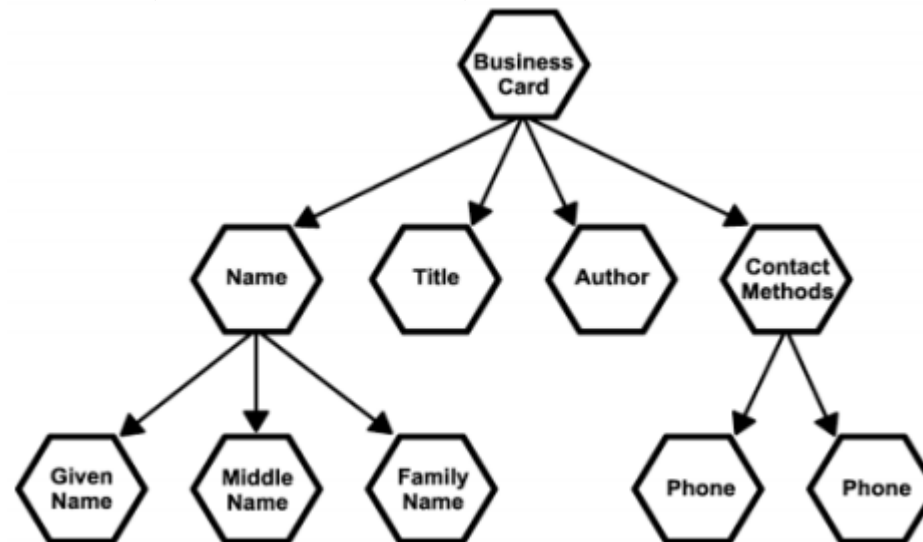
Modèle de données

Structure of XML Data

Exemple de document

<code><BusinessCard></code>	document/root element
<code><Name></code>	element content
<code><GivenName>Kevin</GivenName></code>	data content
<code><MiddleName>Stewart</MiddleName></code>	data content
<code><FamilyName>Dick</FamilyName></code>	data content
<code></Name></code>	
<code><Title></code>	
Software Technology Analyst	data content
<code></Title></code>	
<code><Author/></code>	empty content
<code><ContactMethods></code>	element content
<code><Phone>650-555-5000</Phone></code>	data content
<code><Phone>650-555-5001</Phone></code>	data content
<code></ContactMethods></code>	
<code></BusinessCard></code>	

Doc XML: est un arbre, bien formé,

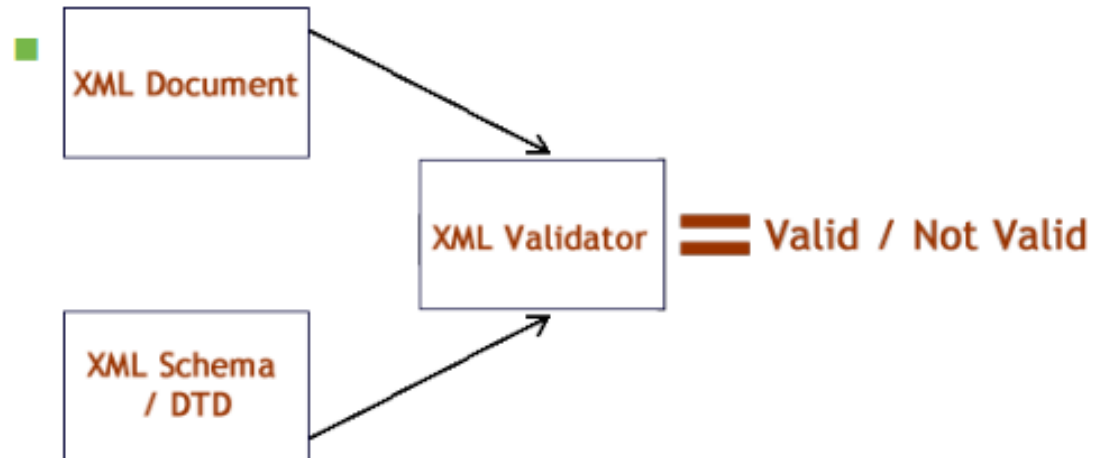


XML Validation

- ❑ Schemas are used to validate XML documents
- ❑ A Schema defines the structure and data constraints

of an instance XML document:

- **DTD** (Document Type Definition)
- **XML Schema** (très puissant)
- **RELAX NG**, etc..



DTD

Document Type Definition

Définition de type de document

Qu'est-ce qu'une définition de type de document ?

- Une définition de type de document (DTD) :
 - constitue la grammaire d'un document XML
 - contient la définition :
 - des éléments
 - des attributs
 - des notations
 - contient des instructions spécifiques qui sont interprétées par le XML Parser pour vérifier la validité du document
 - peut être stockée dans un fichier distinct (DTD externe)
 - peut être incluse dans le document (DTD interne)

Document Type Definition

```
<!ELEMENT letter ( contact+ paragraph )>
<!ELEMENT contact ( name, address, flag )>
<!ATTLIST contact type CDATA #IMPLIED>
<!ELEMENT name ( #PCDATA )>
<!ELEMENT address ( #PCDATA )>
<!ELEMENT flag EMPTY>
<!ATTLIST flag gender ( M | F ) "M">
<!ELEMENT paragraph ( #PCDATA )>
```

DTD

```
<?xml version = "1.0"?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>
  <contact type = "sender">
    <address>Box 12345</address>
    <flag gender = "F" />
  </contact>
  <contact type = "receiver">
    <name>John Doe</name>
    <address>123 Main St.</address>
    <flag gender = "M" />
  </contact>
  <paragraph>It is our privilege to inform
  you about our new database managed with XML.
  </paragraph>
</letter>
```

XML

Exemple de déclaration dans une DTD simple

- Exemple d'une DTD simple avec des déclarations d'éléments :

```
<!ELEMENT employees (employee)>  
<!ELEMENT employee (name)>  
<!ELEMENT name (#PCDATA)>
```

- Exemple de document XML valide basé sur la DTD :

```
<?xml version="1.0"?>  
<employees>  
  <employee>  
    <name>Steven King</name>  
  </employee>  
</employees>
```

- Remarque : Tous les éléments enfant doivent être définis.

Référencer la DTD

- Le document XML référence la DTD :
 - après la déclaration XML et avant la racine en utilisant :

```
<!DOCTYPE employees [ ... ]>
```

- de façon externe avec les mots-clés SYSTEM ou PUBLIC :

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

- de façon interne dans l'entrée <!DOCTYPE root [...]> :

```
<?xml version="1.0"?>  
<!DOCTYPE employees [  
  <!ELEMENT employees (#PCDATA)>  
<employees>Employee Data</employees>
```

Remarque : Utilisez le nom de l'élément racine après <!DOCTYPE.

Déclarations d'éléments

– Syntaxe de déclaration d'un élément :

```
<!ELEMENT element-name content-model>
```

– Quatre types de modèle de contenu :

```
<!ELEMENT job EMPTY> <!-- Empty --> ①
```

```
<!-- Elements: single, ordered list, or choice -->
```

```
<!ELEMENT employees (employee)> ②
```

```
<!ELEMENT employee (employee_id,last_name,job_id)>
```

```
<!ELEMENT job_id (manager | worker)>
```

```
<!-- Mixed -->
```

```
<!ELEMENT last_name (#PCDATA )> ③
```

```
<!ELEMENT hire_date (date| (day,month,year))>
```

```
<!ELEMENT employee_id ANY> <!-- Any --> ④
```

Indiquer la cardinalité des éléments

- Les symboles de cardinalité
 - indiquent le nombre d'éléments enfant autorisés
 - apparaissent sous la forme de suffixes :

Aucun symbole (par défaut)	Obligatoire (un et un seul)
? (point d'interrogation)	Zéro ou un (facultatif)
* (astérisque)	Zéro ou plus (facultatif)
+ (signe plus)	Un ou plus (obligatoire)

- sont placés après :
 - un élément ou un groupe dans le modèle de contenu
 - le modèle de contenu
- Remarque : Pour former un groupe, utilisez des parenthèses.

Déclarations d'attributs

- La syntaxe de déclaration d'un attribut est la suivante :

```
<!ATTLIST element-name attrib-name type default>
```

- La déclaration d'un attribut nécessite :
 - un nom d'élément
 - un nom d'attribut
 - un type d'attribut indiqué par :
CDATA, énumération, ENTITY, ENTITIES, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS et NOTATION
 - un type d'attribut par défaut indiqué par :
#IMPLIED, #REQUIRED, #FIXED ou une valeur littérale
- Exemple :

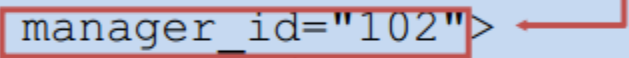
```
<!ELEMENT employee (employee_id, last_name)>  
<!ATTLIST employee manager_id CDATA #IMPLIED>
```

Types d'attribut CDATA et énumération

- CDATA : pour des valeurs de données alphanumériques

```
<!ELEMENT employee (employee_id, last_name)>  
<!ATTLIST employee manager_id CDATA #IMPLIED>
```

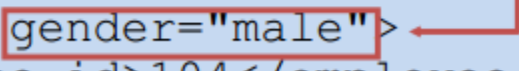
```
<employee manager_id="102">  
  <employee_id>104</employee_id>  
  <last_name>Ernst</last_name>  
</employee>
```



- Énumération : pour permettre un choix dans une liste de valeurs

```
<!ELEMENT employee (employee_id, last_name)>  
<!ATTLIST employee gender (male|female) #IMPLIED>
```

```
<employee gender="male">  
  <employee_id>104</employee_id>  
  <last_name>Ernst</last_name>  
</employee>
```



Définir des valeurs d'attribut par défaut

- Une valeur d'attribut par défaut entre apostrophes :
 - peut être spécifiée dans la DTD après le type de l'attribut :

```
<!ELEMENT employee (employee_id, last_name)>  
<!ATTLIST employee department_id (10|60|90) '90'>
```

- n'est pas précisée en cas d'utilisation des mots-clés `#IMPLIED` ou `#REQUIRED`
- est nécessaire dans la DTD si le mot-clé `#FIXED` est utilisé


```
<!ELEMENT employee (employee_id, last_name)>  
<!ATTLIST employee manager_id CDATA #IMPLIED>  
<!ATTLIST employee min_salary CDATA #FIXED '4000'>
```

- Une valeur d'attribut est obligatoire dans le document XML lorsque le mot-clé `#REQUIRED` est utilisé.

DTD complète : exemple

Fichier : employees.dtd

```
<!ELEMENT employees (employee+)>
<!ELEMENT employee (employee_id, last_name)>
<!ATTLIST employee manager_id CDATA #IMPLIED
                  department_id (10|60|90) '90'>
<!ELEMENT employee_id (#PCDATA )>
<!ELEMENT last_name (#PCDATA )>
```



```
<?xml version="1.0"?>
<!DOCTYPE employees SYSTEM "employees.dtd" [
<!ENTITY title "Mr">
]>
<employees>
  <employee manager_id="100" department_id="10">
    <employee_id>100</employee_id>
    <last_name>&title; king</last_name>
  </employee>
</employees>
```


Valider un document XML par rapport à une DTD

- L'utilitaire de ligne de commande `oraxml` ou le XML Parser :
 - nécessite la présence de `xmlparserv2.jar` dans CLASSPATH
 - utilise l'option `-dtd` pour une validation complète avec une DTD :

```
java oracle.xml.parser.v2.oraxml -dtd emp.xml
```

- génère le message suivant lorsque le document XML est valide :

```
Aucune erreur n'a été trouvée dans le fichier XML  
en entrée lors de son analyse par rapport à la DTD.
```

- peut être appelé en tant qu'outil externe dans Oracle JDeveloper

Langage de Requêtes

XML



- XPATH (1.0 puis 2.0)
 - langage commun de navigation, sélection, extraction
 - Utilisé dans XSLT, XQUERY, XPOINTER, ...
- XSLT 2.0 : XML vers XML, HTML, texte
 - Langage à typage faible, orienté transformation
- XQUERY 1.0 : XML vers XML
 - Langage fonctionnel à typage fort (entrées et sorties)
 - Orienté accès BD

Qu'est-ce que le langage XML Path ?

- Le langage XML Path (XPath) :
 - sert principalement à adresser (référencer) les noeuds d'un document XML représenté sous la forme d'un arbre de noeuds
 - tire son nom du fait qu'il utilise une notation sous forme de chemin pour parcourir la structure hiérarchique d'un document XML
 - utilise une syntaxe compacte et non-XML pour former des expressions à employer dans les URI (Uniform Resource Identifier) et les valeurs d'attributs XML
 - prend en charge les espaces de nom XML
 - est conçu pour être utilisé par des applications XML comme XSLT et XQuery

XPath expression table

Expression	Description
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

L'expression de type chemin de localisation

XPath = *Path expressions* + Conditions

Think of XML as a tree

- Le chemin de localisation peut être :

`/departments/department/department_id`

- un chemin de localisation absolu
- un chemin de localisation relatif qui :
 - comprend une ou plusieurs étapes fournissant des directions de navigation vers les noeuds d'un document XML
 - est défini par rapport au point de départ qu'est le noeud contextuel

```
<?xml version="1.0"?>
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
  </department>
</departments>
```

Expressions XPath abrégées : exemples

```
//last_name
```

Résultat : <last_name>King</last_name>
 <last_name>Kochhar</last_name>

```
employees/employee/name/first_name/text()
```

Résultat : Steven
 Neena

```
employees//salary
```

Résultat : <salary>24000</salary>
 <salary>18000</salary>

```
employees/employee/@employee_id
```

Résultat : employee_id="100"
 employee_id="101"

```
comment()
```

Résultat : <!-- Employee data -->

Remarque : La racine du document est le noeud contextuel.

Prédicats XPath

XPath = Path expressions + Conditions

Think of XML as a tree

Basic
Constructs
[C]
@Price<50

- Un prédicat XPath :

- est une expression booléenne entre crochets évaluée pour chaque noeud de l'ensemble de noeuds

```
//department[department_name="Administration"]
```

- avec une valeur numérique renvoie un résultat numérique qui est converti en valeur booléenne à l'aide de la fonction position()

```
/departments/department[2]  
/departments/department[position()=2]
```

- peut être fourni avec chaque étape de localisation

```
//department[@num<3]/department_id[.="10"]
```

- peut être combiné avec des opérateurs logiques

```
//department[@num>2 and @num<=4]/department_name
```

Fonctions XPath

- Les fonctions XPath :

- sont utilisées dans les prédicats et les expressions

```
function-name(arguments, ...)
```

- se présentent sous la forme d'un nom suivi d'une parenthèse et de zéro, un ou plusieurs arguments.
Par exemple :

```
position()
```

- peuvent renvoyer l'un des quatre types suivants :
 - valeur booléenne
 - nombre
 - ensemble de noeuds
 - chaîne

Exemple des Fonctions de chaîne

Fonction	Description
<code>string(o)</code>	Convertit un objet en chaîne
<code>concat(s,s,...)</code>	Concatène les arguments
<code>substring(s,n,n)</code>	Renvoie une sous-chaîne d'un argument de chaîne, depuis une position de départ jusqu'à une longueur donnée
<code>contains(s,s)</code>	Accepte deux arguments et renvoie la valeur <code>true</code> si le premier argument contient le second
<code>starts-with(s,s)</code>	Comporte deux arguments et renvoie la valeur <code>true</code> si le premier commence comme le second, sinon renvoie la valeur <code>false</code>
<code>string-length(s)</code>	Renvoie la longueur d'une chaîne

- Exemple :

```
//department[starts-with(department_name,'A')]
```

Langage de Requêtes

XQuery



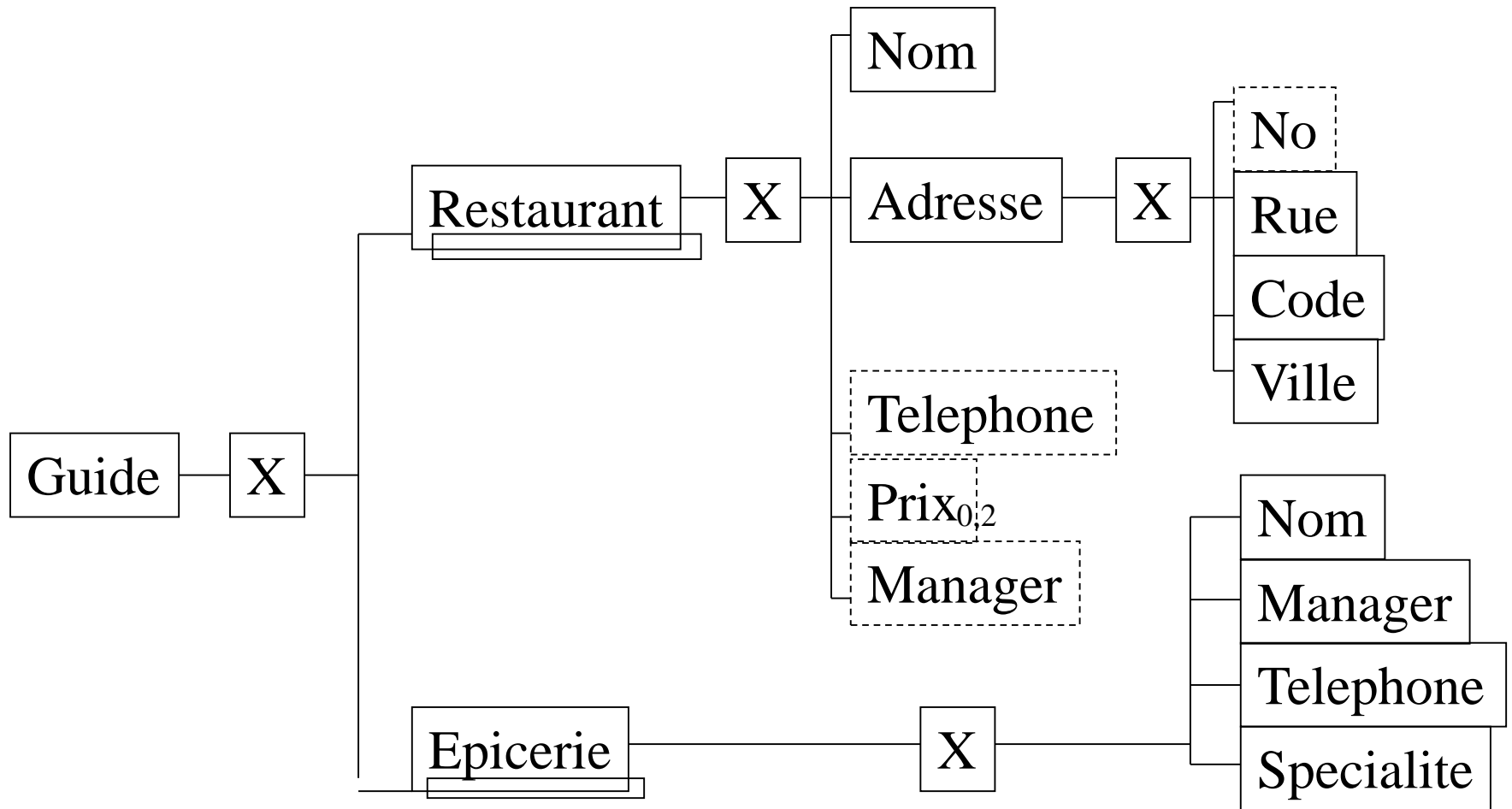
Qu'est ce que XQuery ?

- XQuery est le langage de requêtes pour XML défini et standardisé par le W3C
- XQuery s'impose comme le langage de requêtes:
 - Pour les bases de données XML natives
 - Pour les documents XML textuels (XQuery Text)
 - Pour l'intégration de données (bases de données virtuelles)
- Le besoin d'interroger les bases relationnelles en XQuery existe
 - Pour l'intégration et la publication de données
 - Compétition avec les extensions de SQL

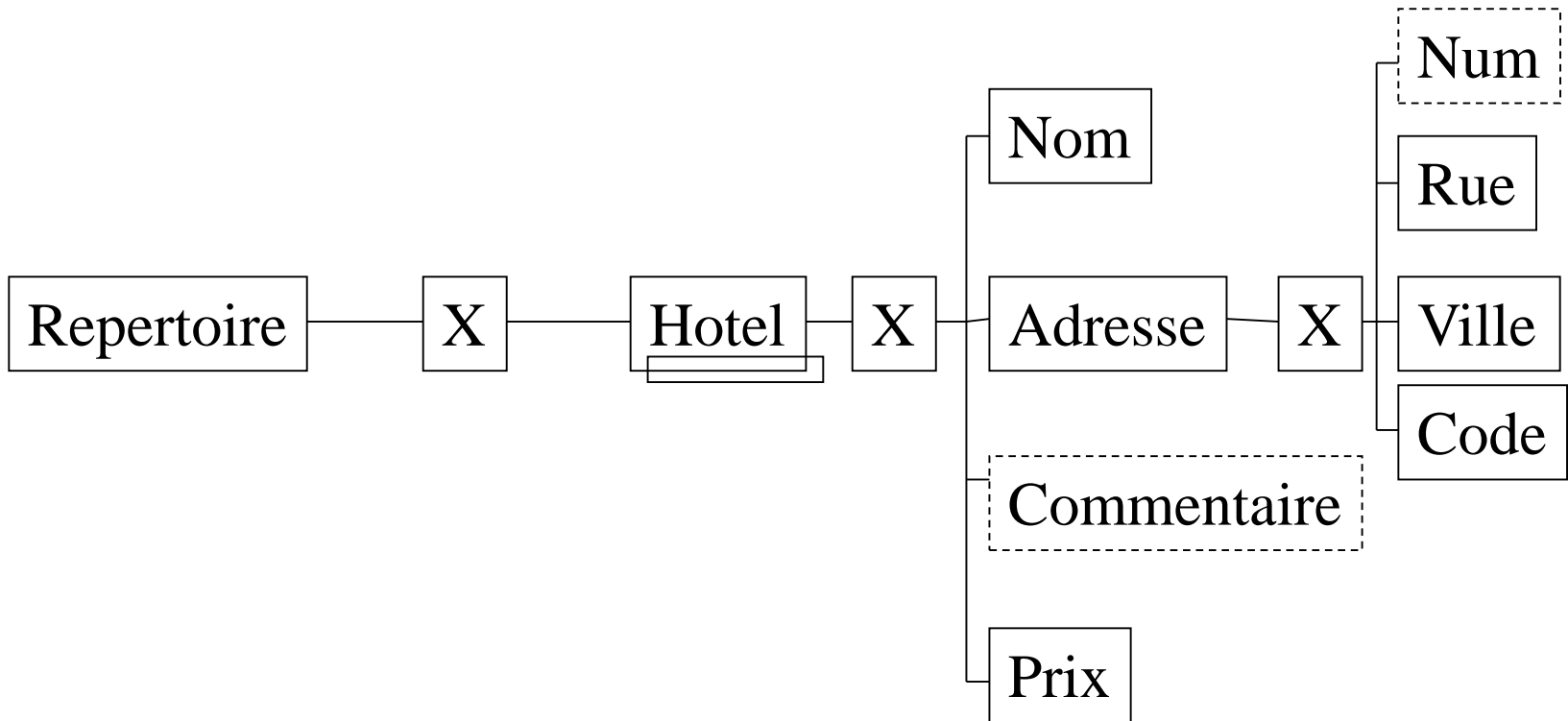
Expression XQuery

for \$var in <forêt> [... \$var in <forêt>]...	→	SQL from
// itération		
let \$var := <sous-arbre> // assignation	→	Variable
where <condition> // élagage		temporaire
return <résultat> // construction	→	SQL
		Select

Guide de données de Guide



Guide de données répertoire

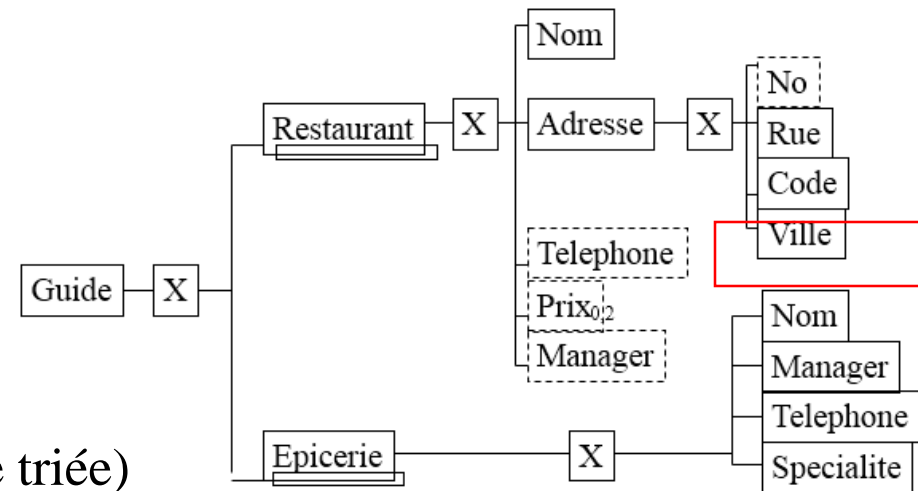


Exemples: BD Guide

GuideNormand:

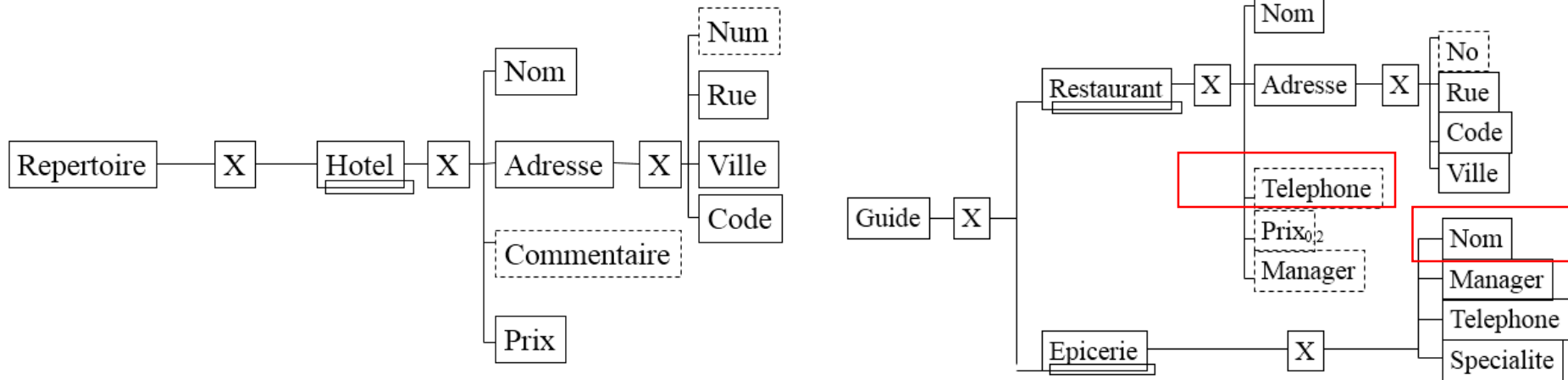
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Guide region="normandie" version="2.0">
  <Restaurant type="français" categorie="***">
    <Nom>Le Grand Hotel</Nom>
    <Adresse>
      <Rue>Promenade M. Proust</Rue><Ville>Cabourg</Ville>
    </Adresse>
    <Prix menu="midi">200</Prix> <Prix menu="soir">300</Prix>
  </Restaurant>
  <Restaurant type="français" categorie="**">
    <Nom>L'absinthe</Nom>
    <Adresse>
      <No>10</No><Rue>Quai Quarantaine</Rue><Ville>Honfleur</Ville>
    </Adresse>
    <Prix menu="midi">150</Prix> <Prix menu="soir">250</Prix>
    <Telephone>0234142189</Telephone>
    <Specialité>Fruits de Mer</Specialite>
  </Restaurant>
</Guide>
```

Exemples (1)



- **Q1** : nom des restaurants de Blida (liste triée)
for \$R **in** **collection**("Guide")/Restaurant
where \$R/Adresse/Ville="Blida"
return \$R/Nom
sortby Nom **descending**

Exemples (3)



- **Q4** : Noms et téléphones des restaurants situés dans la même ville que l'hotel NDJ

for \$R **in** collection("Guide")//Restaurant,

\$H **in** collection("Repertoire")/Hotel

where \$R//Ville=\$H//Ville **and** \$H//Nom=" NDJ"

return <RestauHRN>

<Nom>{ \$R/Nom/text() } </Nom>

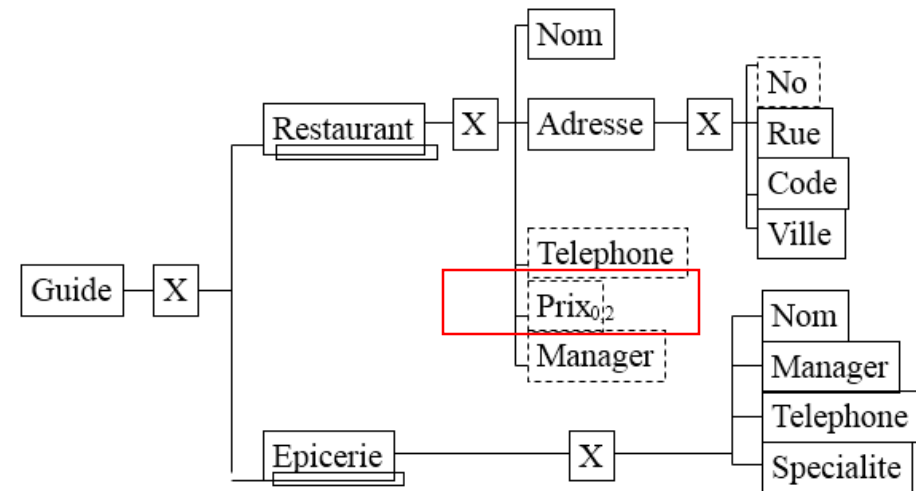
<Tel> { \$R/Telephone/text() } </Tel>

</RestauHRN>

Exemples (4)

- **Q5** : nombre de restaurants dans la collection Guide
let \$R := **collection**("Guide")/Restaurant
Return <NbRest> {count(\$R)}</NbRest>
- **Q6** : noms et adresses des restaurants dont la rue contient la chaine « R1 »
for \$R **in** **collection**("Guide")/Restaurant
where **contains**(\$R//Rue, « R1 »)
return <Res>\$R/Nom
 <Adr>{ \$R/Adresse//text() } </Adr>
 </Res>

Exemples (8)



- Q10** : noms et adresses des restaurants ayant au moins un prix supérieur à 200
for \$R **in** **collection**("Guide")/Restaurant
where some \$P **in** \$R/Prix **satisfies** (**number**(200)<\$P)
return
 <RestC>{ \$R/Nom } { \$R/Adresse } </RestC>

Exemples (7)

- **Q9** : Nom de chaque restaurant avec le prix moyen proposé
for \$R **in** **collection**("Guide")/Restaurant
let \$A := \$R//Prix
return <res>
 { \$R/Nom }
 <MoyPrix>{ avg(\$A) }</MoyPrix>
 </res>

Exemples (9)

- **Q11** : noms et adresses des restaurants ayant tous les prix inférieurs à 100
for \$R **in** **collection**("Guide")/Restaurant
where every \$P **in** \$R/Prix **satisfies**
 (number(100)>\$P)
return
 <RestPC>{ \$R/Nom } { \$R/Adresse } </RestPC>