

---

---

# **Bases de données avancées**

## *Bases de données XML*

---

---

*Dan VODISLAV*

**Université de Cergy-Pontoise**  
**Master Informatique M1**  
**Cours BDA**

---

---

## **Plan**

---

---

- Rappels XML: format, XPath, XQuery
- Bases de données XML
- Stockage XML
  - Stockage relationnel
  - Stockage natif
- Indexation XML
- Interrogation XML
  - SQL/XML
- XML dans Oracle

# Bases de données XML

---

---

- Base de données XML = base de données qui s'appuie sur un modèle logique de données XML
  - Représentation XML des données stockées / retournées par les requêtes
  - Langage de définition de données (LDD) spécifique à XML
  - Langage de requêtes / manipulation des données (LMD) spécifiques XML
  - Stockage physique et indexation spécifiques à XML
- Modèles de données
  - Relationnel: tables
  - Objet: objets persistants
  - Semi-structuré: arbres et texte → XML

## Rappels XML

---

---

- XML: eXtensible Markup Language
  - Langage de description de documents structurés
  - Utilisation de balises (balisage structurel)
- Standard W3C pour l'échange/publication de données sur le web
- Héritage:
  - HTML: documents publiés sur le web, mais ensemble de balises fixé, dédiées à la *présentation* du contenu → difficile à exploiter le contenu
  - Données structurées: bases de données, contenu structuré, mais mal adapté à la structure des documents web (texte, structure irrégulière)
  - XML décrit *le contenu* de façon (semi-)structurée, *flexible*, adaptée aux documents du web

# Exemple

- 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>
```

- 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>
```

## Formes sérialisée et arborescente

- Forme sérialisée d'un document/élément

- Chaîne de caractères (texte) incluant balises et contenu textuel

### Exemple

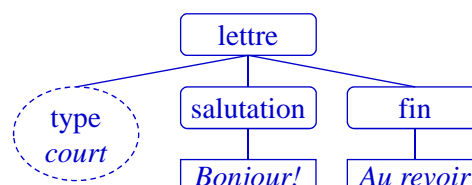
```
<lettre type='court'><salutation>Bonjour!</salutation><fin>Au
revoir</fin></lettre>
```

ou avec un peu de mise en forme

```
<lettre type='court'>
  <salutation>Bonjour!</salutation>
  <fin>Au revoir</fin>
</lettre>
```

- Forme arborescente

- Utilisée par les applications, modèle DOM (W3C)



# XPath

---

---

- Langage de sélection d'un ensemble de nœuds dans un document XML
  - Utilise des *expressions de chemin* pour désigner des nœuds dans l'arbre
- Une expression de chemin XPath: suite d'**étapes** à partir d'un **nœud contexte**  
[/]étape<sub>1</sub>/étape<sub>2</sub>/.../étape<sub>n</sub>
- Étape = axe::filtre[prédicat<sub>1</sub>]...[prédicat<sub>n</sub>]  
Exemple: child::B[position()=1] (abréviation: B[1])
- Axe: optionnel (par défaut child)
  - Spécifie *un ensemble des nœuds* par rapport au nœud contexte + *un ordre*
- Filtre: obligatoire, décrit le sous-ensemble de nœuds de l'axe retenu
- Prédicats: optionnels, décrivent les conditions à satisfaire par les nœuds
  - Combinés par l'opérateur "ET" logique

# Axes XPath

---

---

- Douze types d'axes
  - child (axe par défaut): enfants directs du nœud contexte
  - parent (..): nœud parent
  - attribute (@): nœuds attribut du nœud contexte
  - descendant (//): nœuds descendants du nœud contexte
  - descendant-or-self: descendants, y compris le nœud contexte
  - ancestor: nœuds ancêtres du nœud contexte
  - ancestor-or-self: ancêtres, y compris le nœud contexte
  - following: nœuds suivants dans l'ordre du document
  - following-sibling: frères suivants dans l'ordre du document
  - preceding: nœuds précédents dans l'ordre du document
  - preceding-sibling: frères précédents dans l'ordre du document
  - self (.): le nœud contexte lui-même
- **Attributs: seul l'axe attribute désigne des nœuds attribut !**

# Filtres et prédicats

---

- *Filtres*: deux façons de filtrer les nœuds d'un axe:
  - Par leur **nom**
    - Pour les nœuds qui ont un nom (*Element, Attribute, ProcessingInstruction*)
  - Par leur **type**
    - *text()*, *comment()*, *processing-instruction()*
    - \* : nœuds de type élément ou attribut
    - *node()* : tous les types de nœud
- *Prédicat*: tests connectés par les opérateurs logiques "ET" et "OU"
  - Négation: par la fonction *not()*
- *Test* = expression booléenne élémentaire: comparaison, fonction booléenne
  - Expression de chemin convertie en booléen : *false* si l'ensemble est vide, sinon *true*

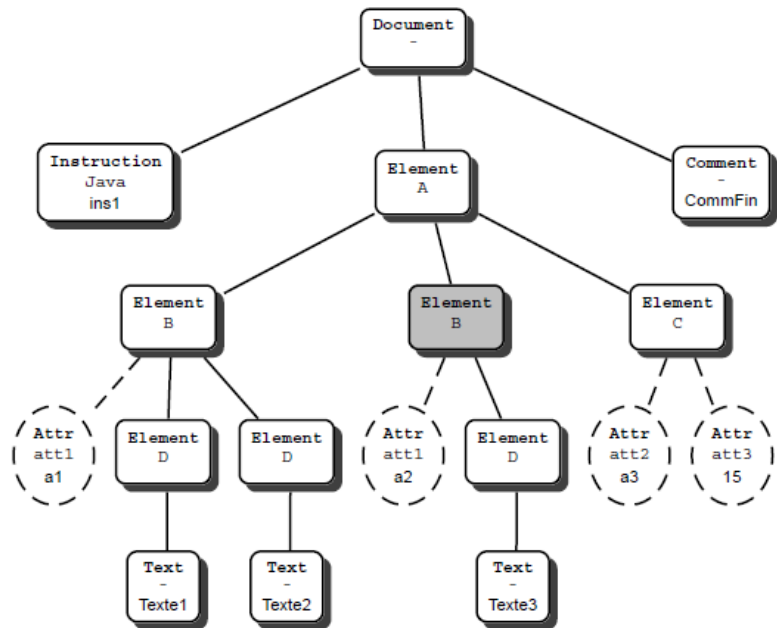
# Fonctions XPath

---

- Pour nœuds
  - *count(expr)*: nombre de nœuds dans l'ensemble produit par l'expression
  - *name()*: nom du nœud contexte
- Pour chaînes de caractères
  - *concat(ch1, ch2, ...)*: concaténation
  - *contains(ch1, ch2)*: vérifie si *ch1* contient *ch2*
  - *substring(ch, pos, l)*: extrait la sous-chaîne de *ch* de longueur *l*, commençant à la position *pos* (les positions démarrent à 1)
  - *string-length(ch)*: longueur de la chaîne
- Pour booléens
  - *true()*, *false()*: les valeurs vrai/faux
  - *not(expr)*: négation de l'expression logique
- Pour numériques
  - *sum(expr)*: somme des valeurs des nœuds produits par l'expression
  - *avg(expr)*: moyenne des valeurs des nœuds produits par l'expression

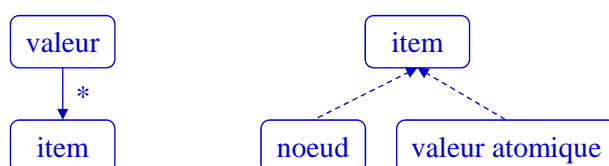
# Exemples

- D
- D/text()
- descendant::node()
- ..
- following::node()
- preceding-sibling::node()
- //@att1
- /A//D
- /comment()
- ../\*
- /A/B//text()[1]
- /A//text()[1]
- //B[@att1="a2"]/D
- /A/B[last()]
- //B[count(D)>1]
- /A/B[D="Texte2"]/D/text()



# XQuery

- Langage de requêtes pour bases de données XML
  - Langage fonctionnel typé, sans effets de bord
- Modèle de données XDM basé sur des *séquences ordonnées*
  - **Valeur** = séquence ordonnée (liste) d'*items*
  - **Item** = *nœud* (tout type DOM) ou *valeur atomique*
    - Document, élément, attribut, texte, ... + valeurs atomiques de différents types
  - Chaque nœud et chaque valeur atomique a un **type** (XML Schema)
  - Résultat de requête XQuery = valeur = séquence d'items



# Séquences

---

- Pas de distinction entre item et séquence de longueur 1
  - Ex: 39 = (39)
- Une séquence peut contenir des valeurs hétérogènes
  - Ex: (39, "toto", <toto/>)
- Pas de séquences imbriquées
  - Ex: (39, (1, 2), "toto", <toto/>) = (39, 1, 2, "toto", <toto/>)
- Une séquence peut être vide
  - Ex: ()
- L'ordre est important
  - Ex: (1, 2) ≠ (2, 1)

# Expressions XQuery simples

---

- Valeurs atomiques littérales (des types simples XML Schema)
  - Ex: 39, "toto", 3.9, etc.
- Nœuds XML sous forme littérale
  - Ex: <film annee="2007">  
<titre>La même</titre>  
</film>
- Valeurs obtenues par des constructeurs simples
  - Ex: true(), false(), date("2006-12-08")
- Collections de documents, documents
  - document(uri-document) → retourne un item de type nœud Document
  - collection(uri-collection) → retourne une séquence de nœuds Document
  - Ex: document("biblio.xml"), collection("cinema/films")
- Séquences construites
  - Ex: (1, 2, 3, 4, 5), 1 to 5, (1 to 3, 4, 5)
- Variables
  - Nom précédé du signe \$ (\$x, \$toto, etc.) + valeur (séquence d'items)

# Expressions XQuery complexes

---

- Expressions de chemin (XPath 2.0)
  - Toute expression produisant une séquence de nœuds peut être une étape
- Expressions FLWOR avec définition de variables
- Tests (*If-Then-Return-Else-Return*)
- Fonctions prédéfinies et fonctions utilisateur
- Mélange de littéraux et d'expressions complexes
  - Chaque expression doit être placée entre { } pour qu'elle soit évaluée

Ex. `<comedies>`

```
  {document("films.xml")//film[@genre="comédie"]}
</comedies>
```

---

## Exemple

---

- Document *bib.xml*

```
<bib>
  <book year="2003" title="Comprendre XSLT">
    <author><name>Amann</name><country>FR</country></author>
    <author><name>Rigaux</name><country>FR</country></author>
    <publisher>O'Reilly</publisher>
    <price>28.95</price>
  </book>
  <book year="2001" title="Spatial Databases">
    <author><name>Rigaux</name><country>FR</country></author>
    <author><name>Scholl</name><country>FR</country></author>
    <author><name>Voisard</name><country>FR</country></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>35.00</price>
  </book>
  <book year="2000" title="Data on the Web">
    <author><name>Abiteboul</name><country>FR</country></author>
    <author><name>Buneman</name><country>UK</country></author>
    <author><name>Suciu</name><country>US</country></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>
```



# Expressions FLOWR

- Une expression FLOWR ("flower")
  - Itère sur des séquences (**for**)
  - Définit des variables (**let**)
  - Trie les résultats (**order by**)
  - Applique des filtres (**where**)
  - Construit et retourne un résultat (**return**)

- Exemple

```
for $b in document("bib.xml")//book
order by $b/@year
let $al := $b/author
where $b/publisher = "Morgan Kaufmann Publishers"
return <livre nb_auteurs="{count($al)}">
    <title>{$b/@title/string()}</title>
    {$al}
</livre>
```

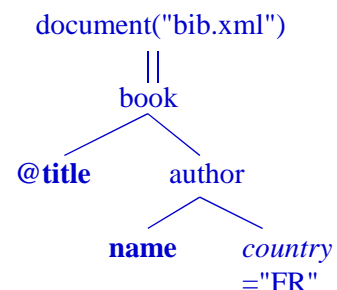
## Requêtes motif d'arbre

- Les requêtes les plus courantes
  - Correspondent aux requêtes relationnelles de sélection/projection

- Exemple : *noms des auteurs français et titres des livres qu'ils ont publiés*

- Principe

- Une variable par nœud de jonction
- Conditions dans le "where"
- Résultats dans le "return"



- Équivalent en XQuery

```
for $b in document("bib.xml")//book, $a in $b/author
where $a/country="FR"
return <resultat>
    {$a/name}
    {$b/@title}
</resultat>
```

# BD XML

- Limitations du modèle de données relationnel

- Modèle très simple: tables

- Cette simplicité a favorisé le développement des SGBD basés sur ce modèle

- Inconvénients

- Difficile de représenter des données complexes: attributs avec plusieurs valeurs, structures irrégulières, texte, images, etc.
- Difficultés à définir l'identité des données
- Traitements et données sont séparés

*Nom de la Relation*                      *Nom d'Attribut*

|                 |                     |             |              |
|-----------------|---------------------|-------------|--------------|
| <b>VEHICULE</b> | <b>Proprietaire</b> | <b>Type</b> | <b>Annee</b> |
|                 | Loic                | Espace      | 1988         |
|                 | Nadia               | Espace      | 1989         |
|                 | Loic                | R5          | 1978         |
|                 | Julien              | R25         | 1989         |
|                 | Marie               | ZX          | 1993         |

*n-uplet*

## Modèle semi-structuré / XML

- Modèle pour le contenu du web

- Mélange de texte et de structure complexe: *arbres ordonnés*

- Plus complexe que le relationnel ou l'objet

- Faiblement typé et auto-descriptif (schéma facultatif)

- Flexible pour gérer l'hétérogénéité

- Langage de requêtes XQuery/XPath

- Expressions de chemin, récursivité, requêtes texte

# Comparaison relationnel - XML

- Modèle conceptuel



- Traduction en modèle relationnel

**Acteur** (nom, pays)

**Film** (titre, année)

**Joue** (nom, titre)

- Traduction en modèle XML

- Plus proche du modèle conceptuel

```
<Acteurs>
  <Acteur nom="Daniel Auteuil">
    <pays>France</pays>
    <Film année="1986">Jean de Florette</Film>
    <Film année="1993">La Reine Margot</Film>
    <Film année="1996">Le Huitième Jour</Film>
  </Acteur>
  ...
</Acteurs>
```

## Pourquoi des bases de données XML?

- XML est un format d'échange, pourquoi le stocker?

- Éviter les (coûteuses) transformations de modèle



- Pourquoi ne pas stocker les documents XML dans des fichiers?

- Possible: fichiers texte auto-descriptifs, un fichier par document, mais...

- Fonctionnalités limitées offertes par le système de fichiers, pas d'accès rapide à des parties du document, pas de support pour concurrence, reprise, ...
- Très lent quand le nombre de fichiers augmente

- Obtenir de bonnes performances → optimiser l'organisation physique

- Stockage adapté à la navigation dans les arbres, indexation

# Stockage XML

---

- Deux catégories de BD XML
  - *Natives*: le stockage est spécifique au modèle XML
  - *Non-natives*: le stockage se fait au-dessus d'un autre modèle
    - Très utilisé à cause de la prédominance des SGBD relationnels
- Types de documents XML
  - *Orientés données*: contenu structuré, ordre peu important
  - *Orientés documents*: proche de documents consultables, ordre important
    - Beaucoup de texte et de contenu mixte (style HTML)

```
<Acteurs>
  <Acteur>
    <nom>Daniel Auteuil</nom>
    <pays>France</pays>
  </Acteur>
  <Acteur>
    <nom>Julia Roberts</nom>
    <pays>États Unis</pays>
  </Acteur>
</Acteurs>
```

```
<Acteurs>
  Parmi les acteurs présents à cette
  avant-première on peut citer <nom>Daniel
  Auteuil</nom> (<pays>France</pays>)
  ou <nom>Julia Roberts</nom> (<pays>États
  Unis</pays>), ainsi que ...
</Acteurs>
```

---

## Stockage de XML dans le relationnel

---

- Distance assez grande entre les modèles
  - Tables non ordonnées ↔ arbres ordonnés
  - Mieux adapté aux documents XML orientés données
- Problèmes
  - Trouver le schéma relationnel pour stocker du XML
  - Traduire du XQuery en SQL
  - Construire des résultats XML à partir des tables
- Deux façons de stocker du XML dans le relationnel
  - Dans des tables: contenu décomposé entre plusieurs tables
    - Adapté à l'XML orienté données
    - Avantage: accès à des parties du document
  - Dans des LOB: contenu stocké comme un bloc
    - Adapté à l'XML orienté documents
    - Avantage: reconstitution rapide du document

# Stockage relationnel générique

- Méthode simple, valable pour n'importe quel document
  - Une table pour l'arbre, une autre pour les valeurs
  - Inconvénient: beaucoup de jointures pour accéder aux fragments XML

```
<cinema>
  <film>
    <titre>Titanic</titre>
    <acteur>L. di Caprio</acteur>
    <acteur>K. Winslett</acteur>
  </film>
  <film>
    <titre>Alien</titre>
    <mes>R. Scott</pays>
    <acteur>S. Weaver</acteur>
  </film>
</cinema>
```

A

| Id | Libellé | Pos | Type  | Parent |
|----|---------|-----|-------|--------|
| 1  | cinema  | 1   | ref   | 0      |
| 2  | film    | 1   | ref   | 1      |
| 3  | titre   | 1   | cdata | 2      |
| 4  | acteur  | 2   | cdata | 2      |
| 5  | acteur  | 3   | cdata | 2      |
| 6  | film    | 2   | ref   | 1      |
| 7  | titre   | 1   | cdata | 6      |
| 8  | mes     | 2   | cdata | 6      |
| 9  | acteur  | 3   | cdata | 6      |

V

| Id | Valeur       |
|----|--------------|
| 3  | Titanic      |
| 4  | L. di Caprio |
| 5  | K. Winslett  |
| 7  | Alien        |
| 8  | R. Scott     |
| 9  | S. Weaver    |

# Stockage relationnel générique (suite)

- Requête XPath

```
//film[acteur="S. Weaver"]/titre
```
- Requête SQL équivalente

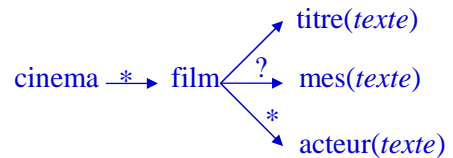
```
select vt.Valeur
from A f, A a, A t, V va, V vt
where f.Libellé="film" and
      a.Libellé="acteur" and
      t.Libellé="titre" and
      a.Parent=f.Id and
      t.Parent=f.Id and
      va.Id=a.Id and
      va.Valeur = "S. Weaver" and
      vt.Id=t.Id
```

→ 4 jointures et 4 sélections

# Stockage relationnel guidé par le schéma

- Idée: utiliser le schéma XML pour en déduire un modèle conceptuel, traduit ensuite en relationnel
  - Graphe structurel décomposé en sous-arbres → une table par sous-arbre
  - Racine sous-arbre: nœud ayant  $n \neq 1$  arcs entrants ou 1 arc entrant marqué \* ou +

```
<!ELEMENT cinema (film)*>
<!ELEMENT film (titre, mes?, acteur*)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT mes (#PCDATA)>
<!ELEMENT acteur (#PCDATA)>
```



```
Cinema(id)
Film(id, parentId, titre, mes)
Acteur(id, parentId, acteur)
```

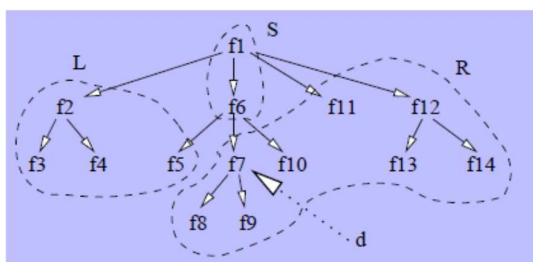
//film[acteur="S. Weaver"]/titre



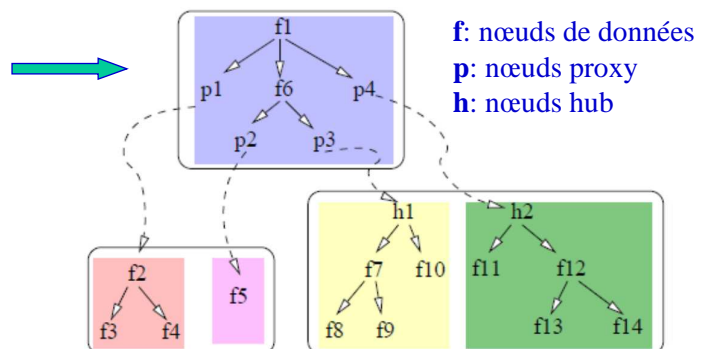
```
select f.titre
from Film f, Acteur a
where a.parentId=f.id and
a.acteur="S. Weaver"
```

# Stockage XML natif

- Stockage basé sur des structures d'arbre
  - DOM persistant → mémoire virtuelle pour DOM
  - Pages disque organisées sous forme d'arbre
- Exemple: Natix
  - Un arbre XML est découpé en articles (sous-arbres)
  - Une page = un ou plusieurs articles + liens vers les pages suivantes
  - Utilisation de nœuds proxy/hub pour créer les liens entre pages



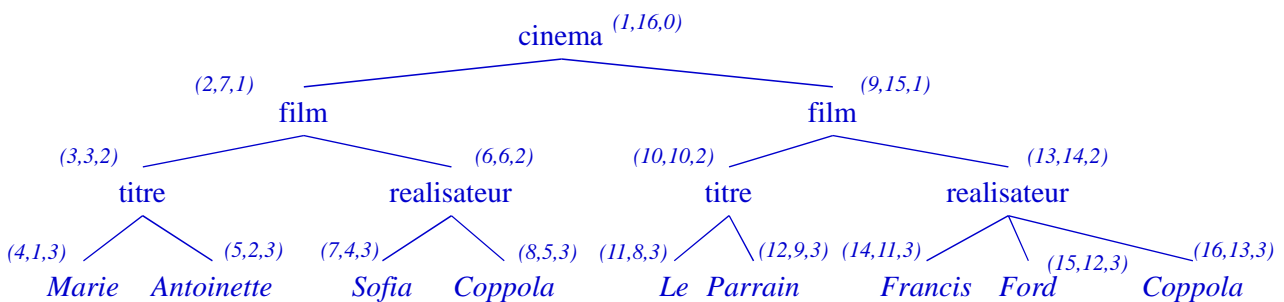
Éclatement d'une page par rapport à un nœud pivot *d*



# Indexation XML

- Particularités
  - Requêtes de chemin → indexation *de la structure* d'arbre
  - Requêtes sur le texte → indexation *des mots* du texte
- Les index relationnels (arbres B+) sont aussi utilisés
  - Éléments/attributs dont la valeur est de type "relationnel"
- Exemple d'index mixte structure/texte
  - Schéma de numérotation Dietz pour les nœuds d'un arbre
  - **Identifiant structurel** :  $IS=(pre, post, niv)$ 
    - *pre*: position du nœud dans un parcours pré-ordre de l'arbre
      - Suit l'ordre d'ouverture des balises dans le document
    - *post*: position du nœud dans un parcours post-ordre de l'arbre
      - Suit l'ordre de fermeture des balises dans le document
    - *niv*: niveau du nœud dans l'arbre
      - Permet de vérifier des relations parent-enfant entre les nœuds

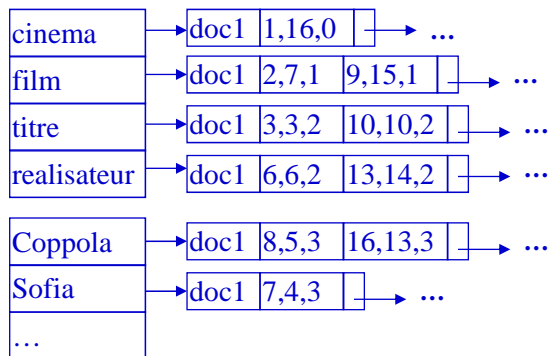
## Index XML texte/structure



- Identifiants structurels *pour tous les nœuds* de l'arbre et *pour tous les mots* du texte
  - $n2$  après  $n1$  :  $pre(n1) < pre(n2)$
  - $n2$  descendant de  $n1$  :  $pre(n1) < pre(n2) \wedge post(n1) > post(n2)$
  - $n2$  enfant de  $n1$  :  $n2$  descendant de  $n1 \wedge niv(n2) = niv(n1)+1$
  - $n$  contient *mot* : *mot* descendant de  $n$
  - $n$  contient directement *mot* : *mot* enfant de  $n$

## Index texte/structure et requêtes

- Index = balise/mot → liste IS groupés par document
- Exemple requête: `//film[realisateur ftcontains "Sofia"]/titre`
  - Retrouver les listes d'IS pour chaque balise/mot de la requête
  - Combiner ces listes par rapport aux relations de chemin



- realisateur: (6,6,2), (13,14,2)
- "Sofia": (7,4,3)
- realisateur ftcontains "Sofia": (6,6,2)
- film: (2,7,1), (9,15,1)
- film[...] : (2,7,1)
- titre: (3,3,2), (10,10,2)
- film[...] / titre: (3,3,2)

## Interrogation XML

- Langages d'interrogation
  - W3C: *XQuery*, *XPath*
  - SQL: *SQL/XML*
- XQuery, XPath
  - Langages spécifiques à XML
  - Modèle XDM (séquences de nœuds typés)
  - Extensions XQuery Full Text, XQuery Update
- SQL/XML
  - Extension de SQL pour manipuler du XML dans un SGBD relationnel
  - Mélange de SQL, XQuery et opérateurs des passage relationnel – XML



# SQL/XML

---

---

- Partie du standard SQL (*ISO/IEC 9075*)
  - SQL partie 14: "XML related specifications" (*ISO/IEC 9075-14*)
  - Plusieurs versions successives: 2003, 2006, 2008
- Caractéristiques SQL/XML
  - Construction de fragments XML à partir de tables relationnelles
    - "Fonctions" spécifiques à XML rajoutées à SQL: XMLELEMENT, XMLATTRIBUTES, XMLAGG, XMLFOREST, etc.
  - Nouveau type de données XML
    - Type d'attribut (colonne) au même niveau que INTEGER, VARCHAR, ...
  - Interrogation en XQuery des attributs XML
    - Fonction XMLQUERY
  - Transformation d'un résultat XML sous forme de table
    - Fonction XMLTABLE

---

---

## Construction de XML à partir de tables

- Exemple utilisé

Table **FILM**

| FID | TITRE   | ANNEE | GENRE | MES |
|-----|---------|-------|-------|-----|
| 1   | Vertigo | 1958  | Drame | 2   |
| 2   | Alien   | 1979  | SF    | 3   |
| 3   | Titanic | 1997  | Drame | 5   |

Table **ARTISTE**

| AID | NOM       | PRENOM | ANNEENAISS |
|-----|-----------|--------|------------|
| 1   | Stewart   | James  | 1908       |
| 2   | Hitchcock | Alfred | 1899       |
| 3   | Scott     | Ridley | NULL       |
| 4   | Depp      | Johnny | NULL       |
| 5   | Cameron   | James  | 1954       |

# XMLELEMENT

---

---

- **Création d'éléments XML**

- XMLELEMENT crée un élément XML dans le résultat

## Ex. Produire des éléments XML avec les titres des films

```
SELECT XMLELEMENT(NAME "film", TITRE) AS "Films"  
FROM FILM
```

- **Résultat:**

```
Films  
-----  
<film>Vertigo</film>  
<film>Alien</film>  
<film>Titanic</film>
```

- **Remarque:** par défaut les noms d'attributs en SQL sont en majuscules, pour avoir un nom d'élément XML en minuscules il faut le mettre entre guillemets

# XMLATTRIBUTES

---

---

- **Définition d'attributs pour un élément**

- XMLATTRIBUTES: seulement en second paramètre de XMLELEMENT

## Ex. Éléments XML plus complexes

```
SELECT XMLELEMENT(NAME "film",  
                  XMLATTRIBUTES(TITRE AS "titre", ANNEE AS "annee"),  
                  XMLELEMENT(NAME "realisateur",  
                              (SELECT NOM FROM ARTISTE a  
                               WHERE f.MES=a.AID)),  
                  XMLELEMENT(NAME "genre", GENRE)  
          ) AS "Films"  
FROM FILM f WHERE ANNEE<1980
```

```
Films  
-----  
<film titre="Vertigo" annee="1958">  
  <realisateur>Hitchcock</realisateur><genre>Drame</genre>  
</film>  
<film titre="Alien" annee="1979">  
  <realisateur>Scott</realisateur><genre>SF</genre>  
</film>
```

# XMLAGG

---

---

- Regrouper une séquence d'éléments en un seul élément
  - XMLAGG: par exemple regrouper les résultats d'une requête en un seul élément

Ex. Produire des éléments XML avec les titres des films

```
SELECT XMLELEMENT(NAME "films",
                  XMLAGG(XMLELEMENT(NAME "film", TITRE)
                          ORDER BY TITRE)
                  ) AS "Films"
FROM FILM
```

Films

-----

```
<films>
  <film>Alien</film>
  <film>Titanic</film>
  <film>Vertigo</film>
</films>
```

- Remarque: clause ORDER BY possible dans XMLAGG

# XMLAGG versus XMLELEMENT

---

---

- Question: pourquoi a-t-on besoin de XMLAGG en plus de XMLELEMENT?

Ex. La requête avec XMLELEMENT produit autre chose

```
SELECT XMLELEMENT(NAME "films",
                  XMLELEMENT(NAME "film", TITRE))
        ) AS "Films"
FROM FILM
```

Films

-----

```
<films>
  <film>Vertigo</film>
</films>
<films>
  <film>Alien</film>
</films>
<films>
  <film>Titanic</film>
</films>
```

# XMLFOREST

---

---

- Créer plusieurs éléments
  - Produit une séquence XML à partir de valeurs SQL
  - XMLFOREST: plus simple et flexible que plusieurs XMLELEMENT

Ex. Les informations sur les artistes dont le prénom commence par 'J'

## 1. Avec XMLELEMENT

```
SELECT XMLELEMENT(NAME "artiste",
                  XMLELEMENT(NAME "nom", NOM),
                  XMLELEMENT(NAME "prenom", PRENOM),
                  XMLELEMENT(NAME "naissance", ANNEENAISS))
       ) AS "Artistes"
FROM ARTISTE WHERE PRENOM LIKE 'J%'
```

Artistes  
-----  
<artiste>  
 <nom>Stewart</nom><prenom>James</prenom><naissance>1908</naissance>  
</artiste>  
<artiste>  
 <nom>Depp</nom><prenom>Johnny</prenom><naissance></naissance>  
</artiste>  
<artiste>  
 <nom>Cameron</nom><prenom>James</prenom><naissance>1954</naissance>  
</artiste>

# XMLFOREST (suite)

---

---

## 2. Même requête avec XMLFOREST

```
SELECT XMLELEMENT(NAME "artiste",
                  XMLFOREST(NOM AS "nom",
                             PRENOM AS "prenom",
                             ANNEENAISS AS "naissance"))
       ) AS "Artistes"
FROM ARTISTE WHERE PRENOM LIKE 'J%'
```

Artistes  
-----  
<artiste>  
 <nom>Stewart</nom><prenom>James</prenom><naissance>1908</naissance>  
</artiste>  
<artiste>  
 <nom>Depp</nom><prenom>Johnny</prenom>  
</artiste>  
<artiste>  
 <nom>Cameron</nom><prenom>James</prenom><naissance>1954</naissance>  
</artiste>

- Remarque: syntaxe plus simple et élimination des valeurs nulles

# XMLCONCAT

---

---

- Regrouper plusieurs éléments XML en une séquence
  - À la différence de XMLFOREST, produit une séquence XML à partir de valeurs XML (pas SQL)

Ex. Les informations sur les artistes dont le prénom commence par 'J'

```
SELECT XMLCONCAT(  
    XMLELEMENT(NAME "nom", NOM),  
    XMLELEMENT(NAME "prenom", PRENOM),  
    XMLELEMENT(NAME "naissance", ANNEENAISS))  
    ) AS "Artistes"  
FROM ARTISTE WHERE PRENOM LIKE 'J%'
```

Artistes

-----

```
<nom>Stewart</nom><prenom>James</prenom><naissance>1908</naissance>  
<nom>Depp</nom><prenom>Johnny</prenom><naissance></naissance>  
<nom>Cameron</nom><prenom>James</prenom><naissance>1954</naissance>
```

---

---

## Autres opérateurs

- XMLCOMMENT: produit un commentaire
- XMLPI: produit une instruction de traitement
- XMLPARSE: transforme une chaîne de caractères en XML
- XMLSERIALIZE: transforme du XML en chaîne de caractères
- etc.

# Type de données XML

- XML: nouveau type de données pour un attribut SQL
  - Avant: il fallait stocker le XML dans des attributs de type VARCHAR, CLOB ou BLOB
- Syntaxe: **XML**( *contenu*( *type* ) )
  - Seul le mot XML est obligatoire, le contenu et le type sont optionnels
- Contenu
  - DOCUMENT : document XML bien formé
  - CONTENT : fragment XML "enrobé" sous forme de document XML (peut avoir plusieurs éléments racine)
  - SEQUENCE : séquence de tous types de nœuds XML
- Type
  - UNTYPED: pas de schéma XML
  - XMLSCHEMA *description* : avec schéma XML spécifié par *description*
  - ANY: pas de précision (ça peut être n'importe lequel des deux)

## Exemple de table avec attribut XML

```
create table FILMS_XML(  
  FID integer primary key,  
  FILM XML( DOCUMENT( UNTYPED ) ) );
```

Table  
**FILMS\_XML**

| FID | FILM   |
|-----|--|
| 1   | <pre>&lt;film annee='1958'&gt;<br/>  &lt;titre&gt;Vertigo&lt;/titre&gt;<br/>  &lt;genre&gt;Drame&lt;/genre&gt;<br/>  &lt;mes&gt;Alfred Hitchcock&lt;/mes&gt;<br/>&lt;/film&gt;</pre> |
| 2   | <pre>&lt;film annee='1979'&gt;<br/>  &lt;titre&gt;Alien&lt;/titre&gt;<br/>  &lt;genre&gt;SF&lt;/genre&gt;<br/>  &lt;mes&gt;Ridley Scott&lt;/mes&gt;<br/>&lt;/film&gt;</pre>          |
| 3   | <pre>&lt;film annee='1997'&gt;<br/>  &lt;titre&gt;Titanic&lt;/titre&gt;<br/>  &lt;genre&gt;Drame&lt;/genre&gt;<br/>  &lt;mes&gt;James Cameron&lt;/mes&gt;<br/>&lt;/film&gt;</pre>    |

# Interrogation en XQuery

---

- On peut interroger le contenu des attributs XML avec des requêtes XQuery
- Principales commandes
  - Commande XMLQUERY à être utilisée dans la clause SELECT
    - Interrogation des données dans les attributs XML
  - Commande XMLTABLE à être utilisée dans la clause FROM
    - Transformation de données XML en table relationnelle
  - Commande XMLEXISTS à être utilisée dans la clause WHERE
    - Vérification de conditions sur les attributs XML

---

## XMLQUERY

---

- Interrogation de contenu XML
  - Dans la clause SELECT, attributs XML passés sous forme de variables

Ex. Donner les titres des films

```
SELECT XMLQUERY(  
    'for $f in $attr/film  
    return $f/titre'  
    PASSING FILM AS "attr"  
    RETURNING CONTENT  
    NULL ON EMPTY  
    ) AS "Films"  
FROM FILMS_XML  
  
Films  
-----  
<titre>Vertigo</titre>  
<titre>Alien</titre>  
<titre>Titanic</titre>
```

- Remarque: l'attribut XML FILM est transmis en tant que variable \$attr

## XMLQUERY (suite)

---

- Ex. Utilisation de la réponse avec des fonctions agrégat

```
SELECT MIN(XMLCAST(
    XMLQUERY(
        'for $f in $attr/film
        return $f/@annee'
        PASSING FILM AS "attr"
        RETURNING CONTENT
        NULL ON EMPTY
    ) AS integer
) AS "AnneeDuPlusAncien"
FROM FILMS_XML

AnneeDuPlusAncien
-----
1958
```

- Remarque: XMLCAST permet de transformer le résultat de la requête XQuery (texte ici) en entier, afin de lui appliquer la fonction agrégat (MIN)

---

## Construction de tables à partir de XML

---

- Commande XMLTABLE
  - Extrait du contenu XML avec une requête XQuery et formate le résultat sous forme de table relationnelle
  - Utilisée dans la clause FROM
- Mieux adapté que XMLQUERY pour exploiter le contenu mixte relationnel-XML des tables
  - XMLQUERY: XML → XML, exploitable avec des outils XML
  - XMLTABLE: XML → table, exploitable avec SQL



# XMLTABLE

---

---

## Ex. Donner les titres des films

```
SELECT Titres.*
FROM FILMS_XML,
     XMLTABLE(
       'for $f in $attr/film
        return $f/titre'
       PASSING FILM AS "attr"
     ) AS "Titres"
```

```
COLUMN_VALUE
-----
<titre>Vertigo</titre>
<titre>Alien</titre>
<titre>Titanic</titre>
```

### • Remarques:

- Le même effet que le premier exemple avec XMLQUERY, mais en utilisant XMLTABLE
- On est obligés de mettre aussi FILMS\_XML dans le FROM!

---

---

## XMLTABLE (suite)

### Ex. Produire une table avec le titre, l'année et le réalisateur des films sortis avant 1980

```
SELECT Films.*
FROM FILMS_XML,
     XMLTABLE(
       'for $f in $attr/film return $f'
       PASSING FILM_XML.FILM AS "attr"
       COLUMNS
         "titre" VARCHAR(40) PATH 'titre',
         "annee" INTEGER PATH '@annee',
         "realisateur" VARCHAR(40) PATH 'mes'
     ) AS "Films"
WHERE Films.annee < 1980
```

```
titre      annee  realisateur
-----
Vertigo    1958  Alfred Hitchcock
Alien      1979  Ridley Scott
```

## XMLTABLE (suite)

---

### Ex. L'année du film le plus ancien

```
SELECT MIN(Films.annee) AS "AnneeDuPlusAncien"
FROM FILMS_XML,
XMLTABLE(
  'for $f in $attr/film return $f'
  PASSING FILM_XML.FILM AS "attr"
  COLUMNS
    "titre" VARCHAR(40) PATH 'titre',
    "annee" INTEGER PATH '@annee',
    "realisateur" VARCHAR(40) PATH 'mes'
) AS "Films"
```

AnneeDuPlusAncien

-----

1958

## XMLEXISTS

---

- Test d'une condition sur un contenu XML
  - Utilisé la clause WHERE

### Ex. Donner les identifiants des films après 1970

```
SELECT FID
FROM FILMS_XML
WHERE XMLEXISTS(
  '/film[@annee>1970]'
  PASSING BY VALUE FILM)
```

FID

-----

2

3

- **Remarque:** le résultat de XMLEXISTS est faux seulement si le résultat de la requête XQuery est la séquence vide

# XML dans Oracle

---

- Type d'attribut/table XML: XMLTYPE
  - Stockage sous forme de LOB ou de tables
  - Utilisation des attributs XML indépendante du mode de stockage
  - Langage SQL/XML assez proche de la norme

- Exemple

```
create table FILMS_XML(  
  FID:integer primary key,  
  FILM: sys.XMLType);  
create table ARTISTES_XML of sys.XMLType;
```

- Insertion

```
insert into FILMS_XML values(1,  
  XMLTYPE('<film titre="Alien">  
    <acteur>S. Weaver</acteur>  
  </film>'));
```

---

## Différences par rapport au standard

---

- La norme SQL/XML a beaucoup varié et les différentes versions d'Oracle l'ont adoptée au fur et à mesure
  - Version 10.1: pas de XMLQuery, XMLTable, ...
  - Version 10.2: pas de XMlexists, XMLCast
    - XMlexists → fonction *existsNode*(attribut, XPath), qui retourne 1 si l'expression XPath appliquée à la valeur de l'attribut produit une séquence non vide
  - Version 11: proche de la norme SQL/XML actuelle

- Modification d'un attribut XML

- Fonction *updateXML*(attribut, XPath, valeur)

```
update FILMS_XML  
  set FILM = updateXML(FILM,  
    'film[@titre="Alien"]/acteur/text()',  
    'Sigourney Weaver');
```