Cours No 3 - Stockage de données XML

Stocker des documents XML

On peut distinguer entre trois types de documents XML :

- Données XML : typiquement export d'une base de données
  - Structure régulière (schéma)
  - Requêtes "sélection-projection-jointure"
  - Opérations de tri et de regroupement
- Documents XML (données mixtes) : texte + données
  - Fragments de texte de taille importante
  - L'ordre des éléments est important
  - Structure irrégulière
  - Requêtes plein-texte et "sélection-projection"

- Flux de données : service Web
  - Beaucoup de petits fragments indépendantes
  - Requêtes de filtrage (sélection)

Stockage de XML: Critères et Choix

- Temps d'exécution de requêtes
- Complexité des mises-à-jour
- Taille des données stockées
- Intégration avec applications existantes
- Schéma de stockage :
  - indépendante de la DTD/du schéma XML (générique)
  - guidé par la DTD/le schéma XML

Généralement il faut faire un choix parmi ces propriétés $\Rightarrow$
Problème d'optimisation

Choix 1 : Fichiers

Fichiers 'plats' :

- Petits documents
- Langage de requêtes : `grep`, index plein texte
- Avantages : temps de chargement/reconstruction

Choix 2 : Bases de données étendues

SGBD (objet-) relationnelle étendu avec des outils pour le traitement de documents XML :

- Définition d'un schéma relationnel pour stocker des documents XML
- Nouveau type d'attributs XML
- Interrogation avec SQL

Avantages :

- On peut traiter en même temps des données XML et des tables classiques
- Passage doux du Relationnel vers XML

Choix 3 : Bases de données XML natives (NXD)

Bases de données spécifiquement conçues pour XML

- Modèle conçu pour le stockage et l'accès à des arbres ordonnées.
- Le document XML est l'entité centrale de la base (comme une relation dans une BD relationnelle)

Avantages :

- Chargement efficace de gros documents
- Mises-à-jour efficaces

Stockage dans une BD relationnelle

Stockage dans une BD relationnelle

Systèmes :

- Oracle XML
- IBM DB2 XML Extender
- Microsoft OpenXML
- Excellon

Caractéristiques :

- Importation générique/guidé par le schéma
- Langages: SQL + TAD pour XML

Mapping XML ↔ tables

Il faut définir un mapping qui permet

- l'importation de XML vers des tables
- l'exportation du relationnel vers XML

Il faut satisfaire les contraintes de la BD :

- exemple : le nombre d'attributs par table

Documents XML et Relations

XML :

- modèle d'arbres ordonnés
- structure irrégulière : éléments/attributs optionnels, éléments multiples

Relations :

- modèle ensembliste (relation = ensemble de n-uplets)
- absence d'ordre
- schéma obligatoire

Stocker un arbre dans une BD relationnelle

Mapping générique :

- Deux tables : une table binaire pour stocker l'ordre, les balises et la relation parent/enfant, une table unaire pour les valeurs.
- Une table binaire pour chaque type de chemin : schéma Monet
- Attributs de type XML

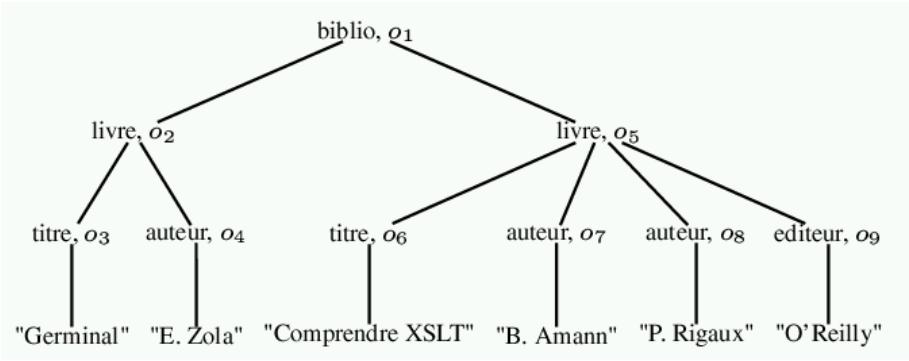
Mapping guidé par la DTD :

- Inlining
- Tables objet/relationnel

Exemple : document XML

```
<?xml version='1.0'?>
<biblio>
  <livre>
    <titre>Germinal</titre>
    <auteur>E. Zola</auteur>
  </livre>
  <livre>
    <titre>Comprendre XSLT</titre>
    <auteur>B. Amann</auteur>
    <auteur>P. Rigaux</auteur>
    <editeur>O'Reilly</editeur>
  </livre>
</biblio>
```

Arbre XML



Relation d'arcs parent/enfant

R:

Part	Pos	Lab	Type	Id
o_0	1	biblio	ref	o_1
o_1	1	livre	ref	o_2
o_1	2	livre	ref	o_5
o_2	1	titre	cdata	o_3
o_2	2	auteur	cdata	o_4
o_5	1	titre	cdate	o_6
o_5	2	auteur	cdata	o_7
o_5	3	auteur	cdata	o_8
o_5	4	editeur	cdata	o_9

S:

Noeud	Val
o_3	Germinal
o_4	E. Zola
o_6	Comprendre XSLT
o_7	B. Amann
o_8	P. Rigaux
o_9	O'Reilly

Requête

XQuery : Les titres de livres d'Emile Zola :

```
For $l in document("biblio.xml")/biblio/livre,
Where $l/auteur = "E. Zola"
Return $l/titre
```

SQL : 4 sélections et 4 jointures :

```
select V2.val
  from R L, R A, R T, S V1, S V2
 where L.lab = 'livre' and A.lab = 'auteur' and
       T.lab = 'titre' and
       L.id = A.par and L.id = T.par and
       A.id = V1.noeud and T.id = V2.noeud and
       V1.val = 'E. Zola';
```

Relation d'arcs parent/enfant

Avantages :

- Format de stockage générique
- Espace utilisée est faible

Inconvénients :

- "Scan" sur une seule grande table
- Beaucoup de jointures

Le modèle Monet

Fragmentation avec classification des noeuds (classe = type de chemin) :

- Chaque type de chemin correspond à une table binaire
- Types d'associations stockées :
  - père/fils
  - noeud/valeur
  - noeud/attribut
  - noeud/rang

Monet: Exemple

biblio.livre

Par	Id
o_1	o_2
o_1	o_5

biblio.livre.titre

Par	Id
o_2	o_3
o_5	o_6

biblio.livre.auteur

Par	Id
o_2	o_4
o_5	o_7
o_5	o_8

biblio.livre.editeur

Par	Id
o_5	o_9

biblio.livre.titre.val

Id	Val
o_3	$''Germinal''$
o_6	$''ComprendreXSLT''$

biblio.livre.auteur.val

Id	Val
o_4	$''E.Zola''$
o_7	$''B.Amann''$
o_8	$''P; Rigaux$

biblio.livre.editeur.val

Id	Val
o_9	$''O.Reilly''$

Monet: Exemple

XQuery :

```
For $l in document("biblio.xml")/biblio/livre,
Where $l/auteur = "E. Zola"
Return $l/titre
```

SQL : 1 sélection et 3 jointures

```
select V2.val
  from biblio.livre.titre A,
       biblio.livre.auteur B,
       biblio.livre.auteur.val V1,
       biblio.livre.titre.val V2
 where A.Par = B.par and
       A.Id = V1.Id and B.Id = V2.Id and
       V2.val = 'E. Zola';
```

Monet: Analyse

Avantages :

- Requêtes avec expressions de chemins
- Exceptions sont traitées naturellement (petites relations)
- Classification des noeuds

Inconvénients :

- Le nombre des relations est linéaire dans la taille du document

Stockage guidé par la DTD: Inlining

Principes :

- Utiliser la DTD pour créer le schéma.
- Décider quand un élément est “mis” dans la table de son parent (“inlining”) et quand il faut créer une table séparée.
- Types d’éléments peuvent être partagés $\Rightarrow$ Redondance
- Trois approches: basic, shared et hybrid

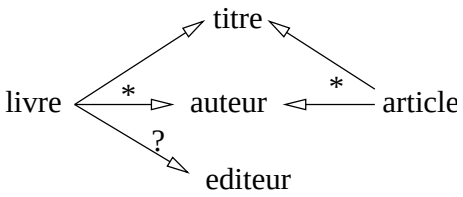
Basic inlining

Créer une ou plusieurs relations pour chaque type d'élément :

- Extraire le sous-graphe qui contient tous les noeuds qu'on peut atteindre à partir du type d'élément.
- Créer récursivement les tables à partir de ce sous-graphe :
  - une table pour la racine.
  - une table pour chaque noeud cible d'un arc $*$
- Les autres noeuds sont transformés en attributs.

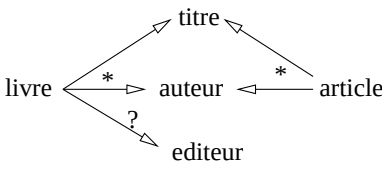
DTD de l'exemple

```
<!ELEMENT livre (titre, auteur*, editeur?) >
<!ELEMENT article (titre, auteur*) >
<!ELEMENT titre #PCDATA >
<!ELEMENT auteur #PCDATA >
<!ELEMENT editeur #PCDATA >
```



Basic Inlining : Exemple

livre(id, parId, titre, editeur)
livre_auteur(id, parId, auteur)
article(id, parId, titre)
article_auteur(id, parId, auteur)
titre(id,titre)
editeur(id, parId, editeur)
auteur(id, parId, auteur)



Basic Inlining : Exemple

Livres de Zola :

```
For $l in document("biblio.xml")/biblio/livre
Where $l/auteur = "E. Zola"
Return $l/titre
```

SQL : 1 jointure et 1 sélection

```
select titre
  from livre, livre_auteur
 where livre.id = livre_auteur.parId
   and auteur = ''E.Zola''
```

Basic Inlining : Exemple

Noms des auteurs :

```
For $a in $biblio//auteur
Return $a
```

SQL :

```
select auteur from auteur
     union
select auteur from livre_auteur
     union
select auteur from article_auteur;
```

Trop de tables $\Rightarrow$ shared/hybrid inlining

La solution d'Oracle9i

Deux solutions :

- Attributs de type *XMLType* :
  - interrogation avec XPath
  - fonctions de transformation relation $\leftrightarrow$ XML
- Mapping canonique dans une table objet-relationnel

Attributs de types XMLType

```
SQL> create table PURCHASEORDER (PODOCUMENT sys.XMLTYPE);
Table created.
SQL> insert into PURCHASEORDER (PODOCUMENT) values (
  2  sys.XMLTYPE.createXML(
  3  '
  4  <PurchaseOrder>
  5     <Refernce>2365</Reference>
  6     <Actions>
  7        <Action>
  8           <User>KING</User>
  9           <Date>12/11/02</DATE>
 10        </Action>
 11     </Actions>
 12  </PurchaseOrder>
 13  '));
```

Requêtes SQL/XML

Requête :

```
SQL> select p.podocument.getClobVal() from purchaseorder p

P.PODOCUMENT.GETCLOBVAL()
---------------------------------
<PurchaseOrder>
   <Reference>2365</Reference>
   <Actions>
      <Action>
         <User>KING</User>
         <Date>12/11/02</DATE>
      </Action>
   </Actions>
</PurchaseOrder>
```

Requête SQL/XML avec extraction XPath

Requête :

```
SQL> select P.PODOCUMENT.extract(
  2         '/PurchaseOrder//User).getClobVal()
  3  from PURCHASEORDER P
  4  where P.PODOCUMENT.extract(
  5         '/PurchaseOrder//Date/text()'
  6         ).getStringVal() = '12/11/02'

P.PODOCUMENT.Extract('/PurchaseOrder//User').GETClobVal()
----------------------------------------
<User>KING</User>
```

Mapping XML ↔ Relationnel/objet

XML:	SQL:
<row> <Person> <Name>Toto</Name> <Addr> <City>Paris</City> <Street>...</Street> </Addr> </Person> </row>	create type AddrType as OBJECT (City VARCHAR2(32) Street VARCHAR2(32)); create table Person (Name VARCHAR2(32) Addr AddrType);

Ce mapping est utilisé pour l'importation et pour l'exportation.

Mapping canonique

Avantages :

- Interrogation “directe” avec SQL
- Vue objet-relationnel de données XML
- Transformation dans la forme canonique avec XSLT

Inconvénients :

- Difficile à mettre en oeuvre sur des documents irrégulière
- SQL n’a pas la puissance de XPath pour les expressions de chemin

Stratégies d’exportation

Trois étapes : extraction-structuration-balisage (ESB)

- Choix 1 : Composition des opérations :
  - ESB table par table
  - E globale suivi de SB
  - ES globale suivi de B
- Choix 2 ; Intégration au moteur de requêtes :
  - SB interne :
    - extension moteur de requêtes
    - meilleure performance (deux fois plus rapide)
  - SB externe :
    - procedures stockées, XSLT, ..
    - pas d’extension du moteur de requêtes

Construction d'un document XML

Oracle fournit une solution ESB interne :

```
SELECT XMLElement("Department",
     XMLForest(d.deptno "DeptNo",
               d.dname "DeptName",
               d.loc "Location"),
      (SELECT XMLAGG(XMLElement("Employee",
                     XMLForest(e.empno "EmployeeId",
                               e.ename "Name",
                               e.job "Job",
                               e.comm "Commission")))
       FROM emp e
       WHERE e.deptno = d.deptno))
FROM dept d;
```

Fonctions SQL/XML d'Oracle

- `XMLAgg` : agrégation de fragments XML
- `XMLConcat` : concaténation de fragments XML
- `XMLElement` : création d'un élément XML
- `XMLForest` : conversion d'une séquence de valeurs en XML
- `XMLColAttVal` : conversion d'une valeur d'attribut en XML
- `XMLSequence` : transformation d'une relation en XML
- `XMLTransform` : application d'une feuille de style XSL
- `ExtractValue` : extraction d'une valeur avec XPath
- `ExtractXML` : extraction d'un fragment XML avec XPath

Stockage Natif

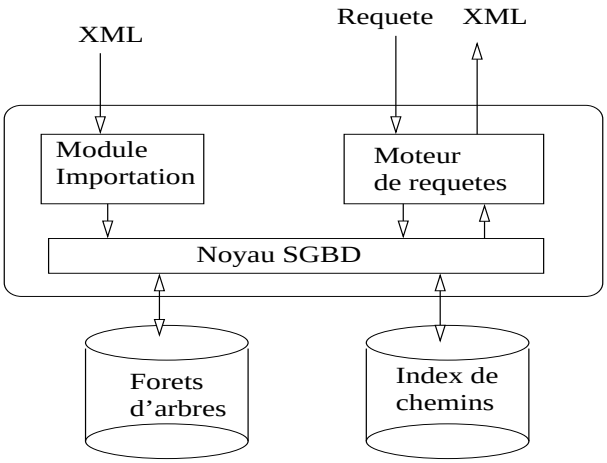
Stockage Natif

- Spécifiquement conçu pour XML
- Entité logique : document XML
- Techniques d’indexation d’arbres

Systèmes de stockage “natives”

- Tamino de Software AG
- Xyleme/Natix de l’Université de Mannheim
- XIndice de Apache
- X-Hive DB
- IXIA Soft TextML

Architecture



Exemple: Natix

Modèle :

- Niveau logique : arbre XML
- Niveau physique : arbre construit à partir de l'arbre XML + noeuds supplémentaires

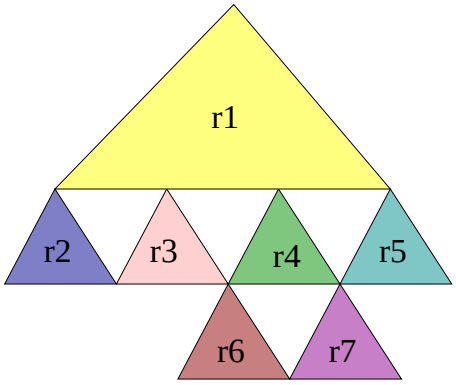
Schéma physique :

- Une page est de taille fixe et contient plusieurs enregistrements de taille variable
- Un enregistrement est un espace de mémoire continu qui peut se déplacer à l'intérieur d'une page et entre les pages.
- Un enregistrement ne peut pas dépasser la taille d'une page.

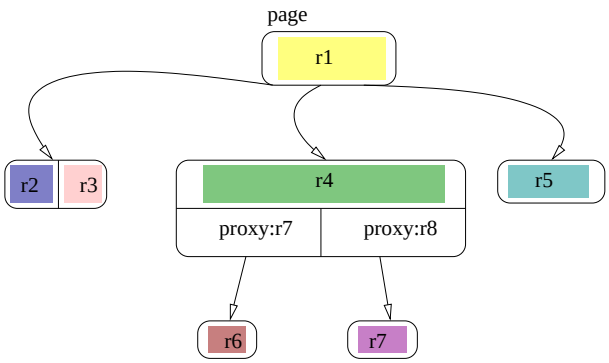
Enregistrement = sous-arbre

- Un enregistrement stocke un sous-arbre d'un document XML
- Enregistrement = le seuil de passage entre la représentation plate et la représentation structurée d'un fragment XML
- Les enregistrements/sous-arbres sont reliés par des noeuds "proxy"

Exemple d'un document XML

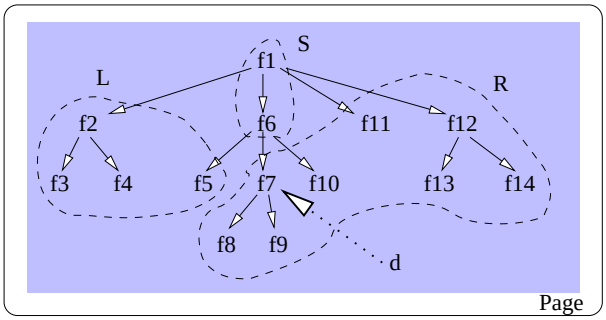


Arbre Natix



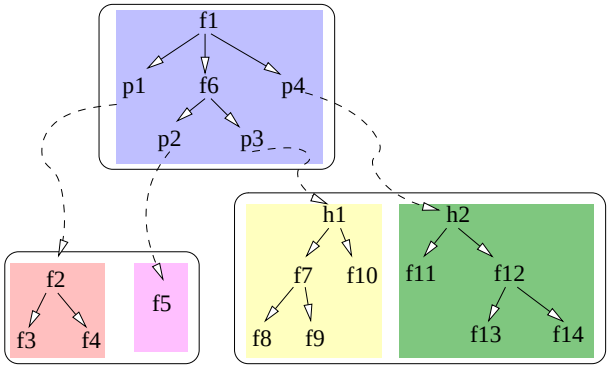
Les opérations de modification sont similaire aux opérations sur un arbre B ⇒ comment trouver le séparateur dans un arbre

Éclatement d'un enregistrement Natix



- le noeud d définit une coupure
- séparateur = les noeuds de la racine jusqu'à d
- comment trouver d: descendre dans le milieu de le l'arbre

Assemblage



Natix : Split Matrix

Une matrice à deux dimensions qui exprime le comportement de regroupement entre les parents et leurs enfants :

- dimensions = types d'éléments
- $M_{A,B} = 0$: séparer A et B si possible
- $M_{A,B} = \infty$: garder A et B dans le me enregistrement si possible
- Sinon: le système décide

Mises-à-jour : XUpdate de XML:DB

XUpdate

- Working Draft XML:DB
- Langage de mise-à-jour déclarative
- Programme de mise-à-jour = document XML

Ordres de mise-à-jour

Un programme de mise-à-jour est un élément de type `modifications` qui contient une séquence d'*ordres de mise-à-jour*.

Chaque ordre de mise-à-jour a un attribut `select` qui permet de choisir des noeuds contextes pour les mises-à-jour.

- `insert-before`, `insert-after`
- `append`, `update`, `remove`, `rename`
- `variable`
- `value-of`
- `if`

Exemple : Document d'origine

```
<?xml version="1.0"?>
 <addresses version="1.0">
   <address id="1">
     <name>Andreas Laux</name>
     <born day='1' month='12' year='1978'/>
     <town>Leipzig</town>
   </address>
 </addresses>
```

Exemple : Programme de mise-à-jour

```
<xup:modifications version="1.0"
    xmlns:xup="http://www.xmldb.org/xup">
   <xup:remove select="/addresses/address[1]/born"  />
   <xup:rename select="/addresses/address[1]/name"  >
     fullname
   </xup:rename>
   <xup:append select="/addresses/address[1]"  >
     <xup:element name="country">
       Germany
     </xup:element>
   </xup:append>
   <xup:insert-after select="/addresses/address[1]" >
     <xup:element name="address">
       <xup:attribute name="id">
         <xup:value-of select="/addresses/address[1]/@id+1"/>
       </xup:attribute>
       <fullname>Lars Martin</fullname>
       <born day='2' month='12' year='1974'/>
       <town>
         <xup:value-of select="/addresses/address[1]/city"/>
       </town>
     </xup:element>
   </xup:insert-after>
 </xup:modifications>
```

Exemple : Document résultat

```
<?xml version="1.0"?>
 <addresses version="1.0">
     <address id="1">
       <fullname>Andreas Laux</fullname>
       <town>Leipzig</town>
       <country>Germany</country>
     </address>
     <address id="2">
       <fullname>Lars Martin</fullname>
       <born day='2' month='12' year='1974'/>
       <town>Leipzig</town>
     </address>
 </addresses>
```

Prochain Cours

- Le Web Sémantique