www.mas-ict.ch

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Module: XML et les bases de données

XML ← Relationnel

SQL2003: SQL/XML
Partie 1

Houda Chabbi Drissi

houda.chabbi@hefr.ch

Besoins

Obtenir:

des tables relationnelles → du XML

donc

structure plate → structure de graphe:

- Nécessité d'un **langage** qui spécifie la conversion

- Nécessité d'une stratégie d'**implémentation**

Réponse: SQL2003-partie 14 – SQL/XML

SQL/XML (1)

- Publier le contenu de tables (ou d'une BD) en documents XML:
  - Nécessite le *mapping* de types SQL en XSD
- Créer des documents XML à partir des résultats de requêtes SQL
  - Nécessite l'extension des requêtes SQL pour créer des éléments XML
- Stoker des documents XML dans les SGBDR et les requêter
  - Nécessité de faire supporter XPATH/XQUERY à travers SQL

SQL/XML (2)

- Intégration de fonctionnalités XML à SQL
- Offre:
  - Type de donnée natif XML Type (colonnes XML)
  - Fonctions d'extraction XPath et Xquery
  - Fonctions de construction de XML (pont relationnel)
  - Insertion et Màj de XML en colonne(s)
- Intégré à Oracle et DB2 ☺
- Différent de SQLXML de Microsoft ☹

SQL/XML

- Avantages:
  - Un standard utilisable avec
    - ✓différents SGBDs
    - ✓les APIs JDBC ou ADO
  - Peut générer n'importe quelle structure XML via une requête SQL.
  - Pour les habitués de SQL: peu d'apprentissage ☺
- Désavantages:
  - Son implémentation peut-être inefficace ☹
  - Reste orienté très relationnel

Annexe: Evolution de SQL / XML

- 1er édition de SQL:2003 XML ← Relationnel
  - Part 14 of the SQL standard
  - Pre-dates XQuery standard!!!
  - Limited functionality - storage and publishing

- 2ieme édition: work in progress (2006) XML → Relationnel
  - More complete integration of XQuery + XQuery Data Model
  - Advanced Query capabilities

SQL2003: SQL/XML (Part. 14) (1)

- Un nouveau type: XML basé sur le modèle *infoset*.
- Des "publishing functions" qui engendrent des valeurs XML à partir d'expressions SQL: XMLELEMENT, XMLFOREST, XMLATTRIBUTE, XMLNAMESPACES, XMLAGG, XMLCONCAT
- Des opérateurs: XMLPARSE, XMLSERIALIZE, XMLROOT
- Un prédicat, IS DOCUMENT, teste si une valeur XML a un élément racine unique.

Annexe: Infoset : XML Information Set

- The XML 1.0 + Namespaces abstract data model
- Defines a modest number of *information items*
  - Element, attribute, namespace declaration, ...
- Each has required and optional properties
  - Name, children, …
  - provides a consistent set of definitions to refer to the information in **a well-formed XML document** →The Schema-**validity** is defined over XML document Infosets

http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

SQL2003: SQL/XML (Part. 14) (2)

- Règles de mapping de SQL à XML :

  ✓SQL identifiers to XML names,
  ✓XML names to SQL identifiers,
  ✓ SQL types to XML Schema types,
  ✓SQL values to XML values,
  ✓SQL tables, schemas, and catalogs to XML values

SQL2003: SQL/XML (Part. 14)

- Un nouveau type: XML.
- Des "publishing functions"
- Des opérateurs
- Un prédicat
- Règles de mapping de SQL à XML

Un nouveau type de donnée: XMLType

- Un nouveau type de donné (comme *varchar*)

- Type qui respecte *infoset (*XML document or XML element or Sequence of XML elements)

[Infoset] *(Recommendation) XML Information Set*, 24 October, 2001
http://www.w3.org/TR/2001/REC-xml-infoset-20011024

- Supporte Xquery

XMLType: exemple

```
Create table Facture(
    idClient            int,
    detail_facture      XMLType
)
```

SQL2006: Specific XML Data Type

- XML(DOCUMENT) : XML bien formé
- XML(CONTENT) : un fragment XML
- XML(SEQUENCE) : chaque valeur XML dans SQL / XML: soit le (SQL) valeur nulle ou d'une séquence à la XQuery.

Refinement:

- XML(DOCUMENT(UNTYPED)): non validée - Pas de XSD associé.
- XML(DOCUMENT(XMLSCHEMA 'http://...')): avec XSD associée
- XML(DOCUMENT(ANY)): mélange des deux précédents

Annexe: type Document

The first node in any document is the **document node**, which contains the entire document. The document node does not correspond to anything visible in the document; it represents the document itself

A ***document* node** is a specialized kind of **element node**. It has **a type *p* but no attributes**. Instead it has an optional URL *u*. The intent of the URL is to specify a specialized data model for this node and its children. A document node looks like this:

- <!doctype *p* "*u*">$c_1 \ldots c_m$ for $m>0$ Exactly **one of the c_i** must be an **element node** and furthermore **it must have type *p***, the same as the document type. The other children, if any, must be either **comment nodes or processing instruction** nodes; **data nodes are not possible**.
- Also, if this document node is not the root node of the document, then $i=m$. In other words, if this document node is not the root, its one child that is an element node must be its last child. *(pas de commentaries après la racine par exemple…)*
- There is one exception to the rule that a document node must have a type. The root node of the XML tree may be an anonymous document node, without a type and without a URL. Such a document node is represented in the document by the *absence* of a `<!doctype>' expression. In other words, if the first expression in the document is not `<!doctype...>', the document has an anonymous root.

Annexe: type Content

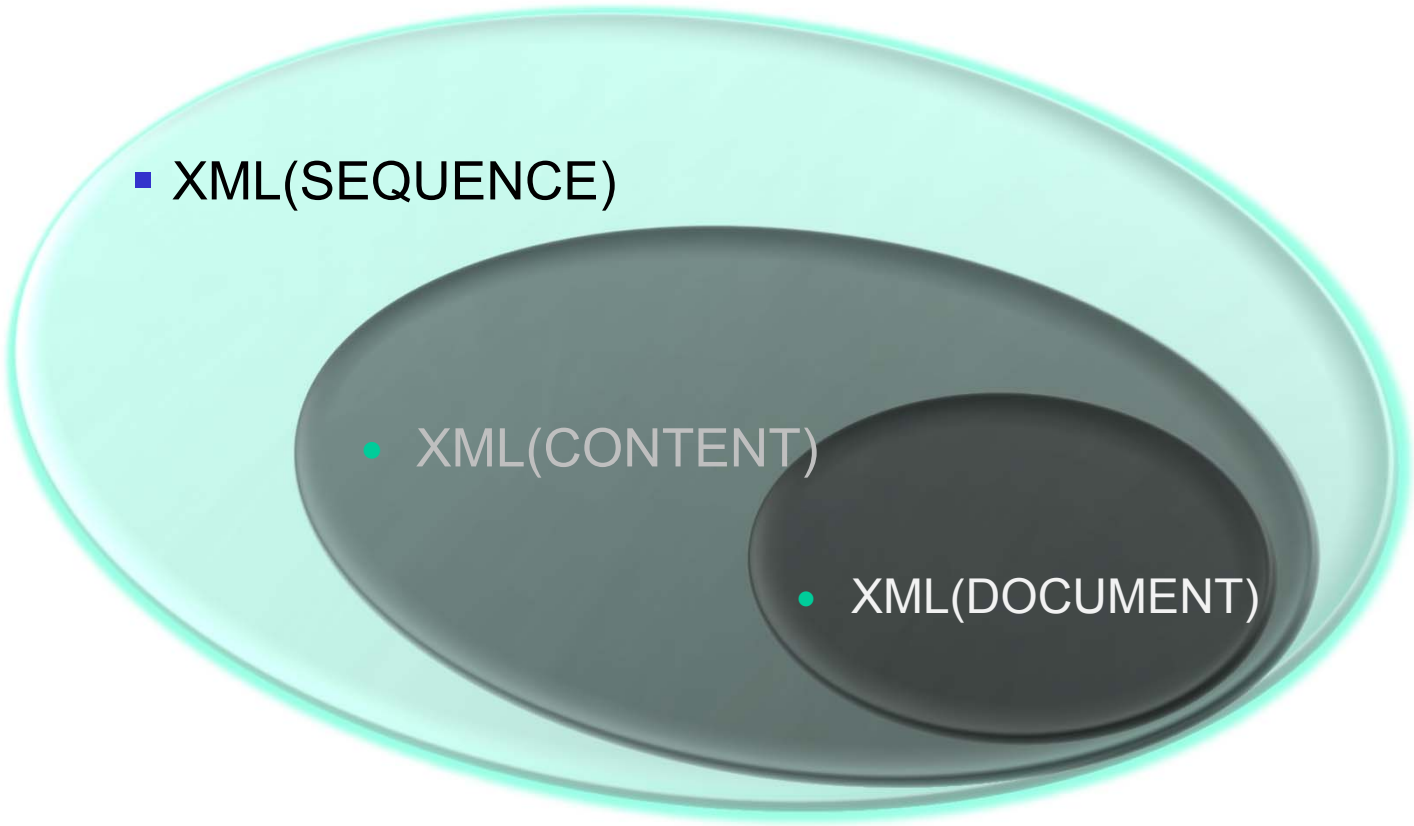
XML fragment refer to part of an XML document, plus possibly some extra information, that may be useful to use and interchange in the absence of the rest of the XML document. There may be multiple top level elements or text nodes wrapped in a document node.

Annexe: type Sequence

A series of items is known as a sequence. An **item** is **a single node** or **atomic value**:

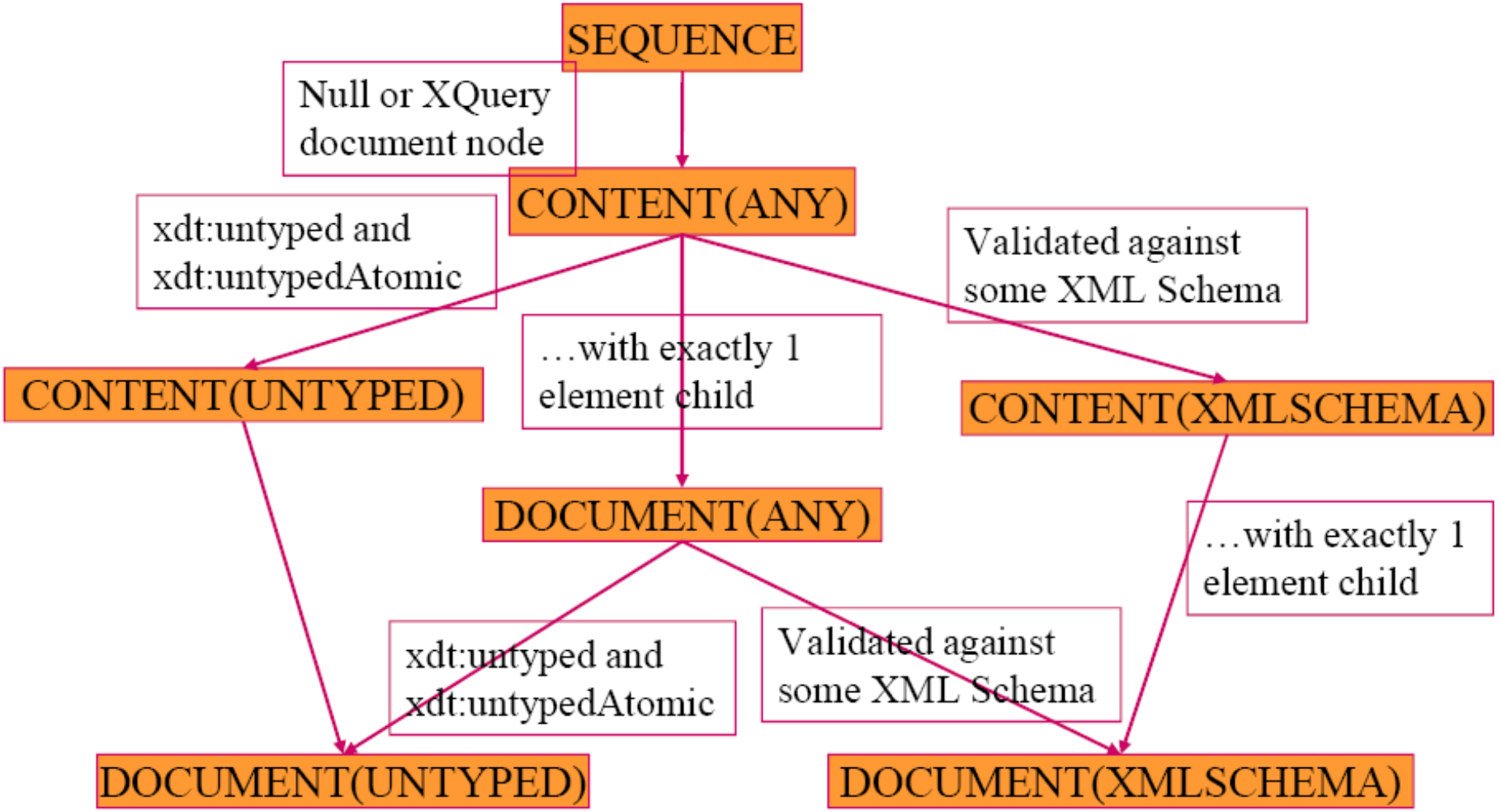
- The kinds of **nodes** are: **document, element, attribute, text, name-space, processing instruction, and comment**. Every node has a **unique node identity** that distinguishes it from other nodes-even from other nodes that are otherwise identical.
- **atomic values** are single values that correspond to the **simple types** defined in the W3C Recommendation, "XML Schema, Part 2" [SCHEMA],

La hiérarchie des types (1)



An XML(SEQUENCE) that is a **Document node** is an XML(CONTENT).
If it has **legal Document children** than it is an XML(DOCUMENT)

La hiérarchie des types (2)



Source http://www.cs.helsinki.fi/u/mplehton/xmldb10/lectures/xmldb04.pdf

XMLType: exemples

```
CREATE TABLE Facture (
      ID integer, detail_facture
      XML(DOCUMENT(UNTYPED)))


CREATE TABLE Facture (
      ID integer, detail_facture
      XML(DOCUMENT(XMLSCHEMA URI 'http://...'))


CREATE TABLE Facture (
      ID integer,
      detail_facture  XML(DOCUMENT(ANY)))
```

SQL2003: SQL/XML (Part. 14)

- Un nouveau type: XML.
- Des "publishing functions"
- Des opérateurs
- Un prédicat
- Règles de mapping de SQL à XML

Attention: SQL vs XML

Une requête SQL ne retourne pas du XML directement:

On obtient une relation (*table*) qui a des colonnes de type XML.

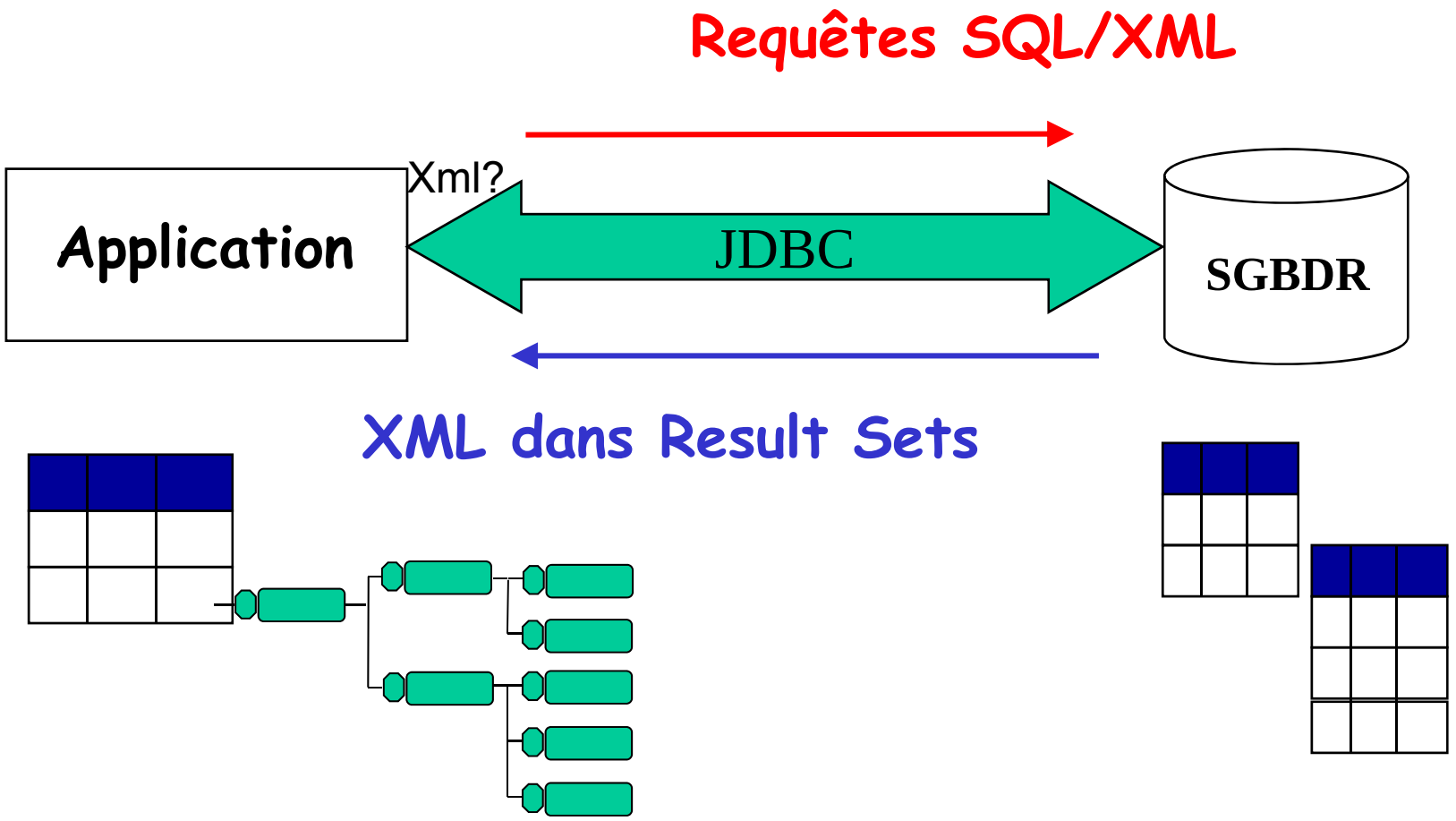
Une première requête SQL/XML

```
select
  c.CustId,
  xmlelement(name customer,
    xmlattributes(c.CustId as id,c.Name as name),
    xmlelement(name projects,
    (select xmlagg(xmlelement(name project,
                xmlattributes(p.ProjId as id,p.Name as name)))
     from Projects p
     where p.CustId=c.CustId))) as "customer-projects"
from Customers c
```

Tables →XML (table avec colonne(s) XML)

CustId	customer-projects
1	<customer id="1" name="Woodworks"> <projects> <project id="1" name="Medusa"/> </projects> </customer>
…	…
4	<customer id="4" name="Hardware Store"> <projects> <project id="2" name="Pegasus"/> <project id="8" name="Typhon"/> </projects> </customer>
…	…

Utilisation ?

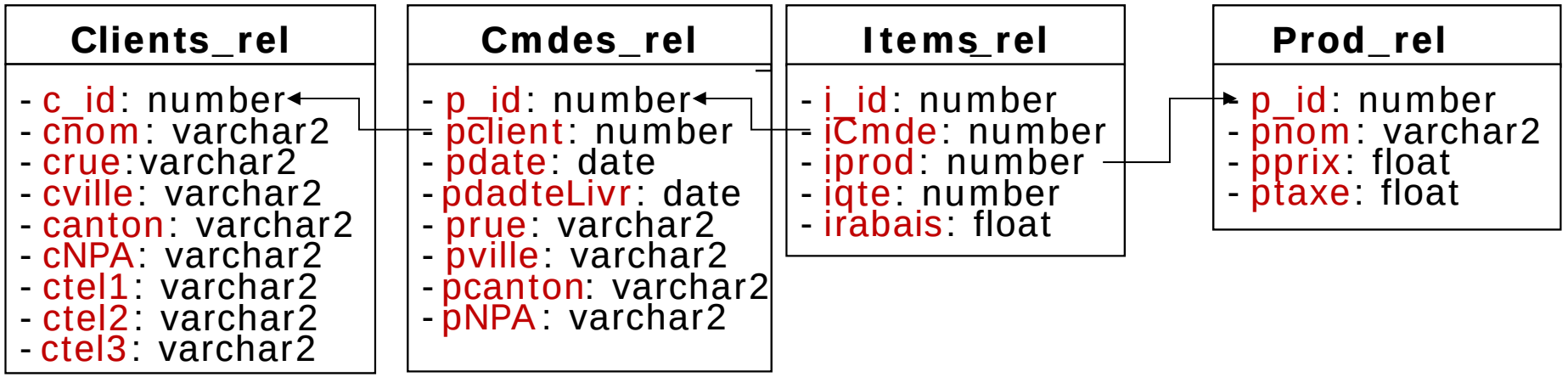


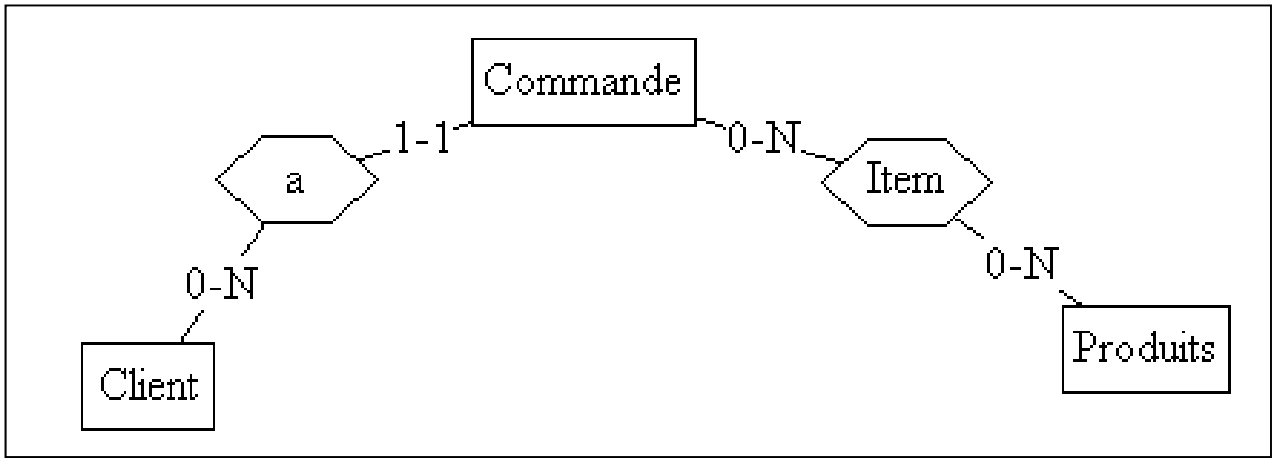
Fonctions SQL/XML pour la publication

Fonction	Effet
Xmlagg()	Prend en argument une collection de fragments et retourne un document XML agrégé ;
xmlconcat()	Reçoit en argument une série d'instances XMLType correspondant aux valeurs d'une colonne pour les lignes d'une table et retourne les instances concaténées ;
xmlelement()	Prend en arg. un nom d'élt, une collection d'attributs optionnels, un contenu d'élt et retourne une instance XMLType ;
Xmlforest()	Convertit la suite de ses argument en XML et retourne un fragment XML concaténation des arguments convertis ;
xmlattributes()	Créer des attributs à partir des colonnes, le nom de chaque colonne étant le nom d'un l'attribut ;
xmlpi()	Créer une "processing instruction"
xmlcomment()	Créer un "commentaire"
xmlgen()	Créer un document XML en utilisant Xquery

Exemple: Schéma relationnel

- Des clients qui ont des commandes.
- Une commande concernent plusieurs produits





Contenu de la base (1)

select * from **Clients_rel**;

C_ID	CNOM	CRUE	CVILLE	CANTON	CNPA	CTEL1	CTEL2	CTEL3
1	Dupont		Fribourg	FR	1700			
3	Martin		Fribourg	FR	1705			
2	Muller		Vaud	VD	1900			
4	Jeannet		Vaud	VD	1905			

select * from **prod_rel**;

P_ID	PNOM	PPRIX	PTAXE
1	Moniteur	589.5	
2	Mouse	50.25	
3	Modem	250.25	
4	Clavier	165.88	

Contenu de la base (2)

select * from **Cmdes_rel**;

P_ID	PCLIENT	PDATE	PDATELIVR	PRUE	PVILLE	PCANTON	PNPA
3	1					FR	

select * from **Items_rel**;

I_ID	ICMDE	IPROD	IQTE	IRABAIS
1	3	2		
2	3	1		
3	3	3		

Le «client» Dupont a une «commande» de «Mouse/Moni-teur/Modem»

XMLElement()

- Pour créer des éléments XML.
- Permet de *XMLiser* les valeurs extraites des tables.

XMLELEMENT (
[NAME] id
[, *XMLAttributes()*]
[, (instance_elements_XML)+]
)

1er argument : nom de l'élément à créer
2ème argument : les attributs (optionel)
3ième argument : contenu de l'élément

XMLElement()

(colonne → élément)

```
SELECT
       XMLELEMENT(NAME "NOM_CLIENT",
          CLI.cnom
       )
FROM Clients_rel CLI
WHERE CLI.canton = 'FR'
```

Res
<NOM_CLIENT>Dupont</NOM_CLIENT>
<NOM_CLIENT>Martin</NOM_CLIENT>

XMLELEMENT peuvent être imbriqués.

XMLElement(): imbriqués

(colonnes →hiérarchie élément)

```
SELECT XMLELEMENT("client_FRIBOURG",
            XMLELEMENT("nom",
               CLI.cnom
            ),
            XMLELEMENT("ville",
               CLI.Cville
            )
       )
FROM Clients_rel CLI
WHERE CLI.canton = 'FR';
```

Res

```
<client_FRIBOURG>
        <nom>Dupont</nom>
        <ville>Fribourg</ville>
</client_FRIBOURG>
```

```
<client_FRIBOURG>
        <nom>Martin</nom>
        <ville>Fribourg</ville>
</client_FRIBOURG>
```

XMLAttributes()

- Pour créer des attributs à un élément

```
XMLATTRIBUTES (
        expr_val [AS alias]
        [, expr_val [AS alias]]*
)
```

Si pas d'alias le nom de l'attribut = le nom de la colonne

XMLAttributes()

(colonnes → attribut)

```
SELECT XMLELEMENT("client_FRIBOURG",
          XMLATTRIBUTES(
             CLI.C_id AS "id"
          ),
          XMLELEMENT("nom",
             CLI.cnom
          ),
          XMLELEMENT("ville",
             CLI.Cville
          )
       )
FROM Clients_rel CLI
WHERE CLI.canton = 'FR'
```

```
<client_FRIBOURG id="1">
        <nom>Dupont</nom>
        <ville>Fribourg</ville>
</client_FRIBOURG>
```

```
<client_FRIBOURG id="3">
        <nom>Martin</nom>
        <ville>Fribourg</ville>
</client_FRIBOURG>
```

XMLAttributes()

- Pour créer des namespace

```
SELECT
      XMLELEMENT(NAME "hr:NOM_CLIENT",
       XMLATTRIBUTES('http://www.hr.com/hr'
                     AS "xmlns:hr" ),

        CLI.cnom
        )
FROM Clients_rel CLI
WHERE CLI.canton = 'FR'
```

Res
<hr:NOM_CLIENT xmlns:hr= 'http://www.hr.com/hr'>Dupont</hr:NOM_CLIENT>
<hr:NOM_CLIENT xmlns:hr= 'http://www.hr.com/hr'>Martin</hr:NOM_CLIENT>

XMLForest()

- Génère une forêt d'éléments XML à partir d'une liste de **valeurs tabulaires**.

```
XMLFOREST (
        expr_val [AS alias]
        [, expr_val [AS alias]]*
)
```

Si pas d'alias le nom de l'élément= le nom de la colonne

XMLForest()

(colonnes → forêt élément)

```
SELECT XMLFOREST(
          CLI.cnom AS "nom",
          CLI.Cville AS "ville",
          CLI.CNPA
       )
FROM Clients_rel CLI
WHERE CLI.canton = 'FR'
```

Nom explicite et
Nom implicite

```
<nom>Dupont</nom>
<ville>Fribourg</ville>
<CNPA>1700</CNPA>
```

```
<nom>Martin</nom>
<ville>Fribourg</ville>
<CNPA>1705</CNPA>
```

XMLAgg()

- Fonction d'agrégation qui retourne une valeur par groupe de lignes impliquées (utilisation éventuelle du **GROUP BY**).

XMLAGG (
XML_value_expr
[**ORDER BY** liste_atributs]
)

XMLAgg(): fonctionnement

- Pour chaque ligne d'un groupe G, l'expression est évaluée et les résultats sont concaténés en un seul résultat pour G: donne une forêt d'éléments XML.

- L'ORDER BY permet d'ordonner les résultats avant leur concaténation.

- Les valeurs nulles ne sont pas conservées

*XMLAgg(): algorithme

- If <ORDER BY> is specified, then let *K* be the number of <sort key>s; otherwise, let *K* be 0
- Let *TXA* be the table of *K*+1 columns obtained by applying <XML_value_expr> to each row of *T1* to obtain the first column of *TXA*, and, for all *i* between 1 (one) and *K*, applying the <value_expr> simply contained in the *i*-th <sort key> to each row of *T1* to obtain the (*i*+1)-th column of *TXA*.
- Every row of *TXA* in which the value of the first column is NULL is removed.
- Let *TXA* be ordered according to the values of the <sort key>s found in the second through (*K*+1)-th columns of *TXA*. If *K* is 0 (zero), then the ordering of *TXA* is implementation-dependent. Let *N* be the number of rows in *TXA*. Let *Ri*, 1 (one) ≤ *i* ≤ *N*, be the rows of *TXA* according to the ordering of *TXA*.
- Case:
  - If *TXA* is empty, then the result of <XML aggregate> is the null value.
  - Otherwise:
    - ✓ Let *V*1 be the value of the first column of *R*1.
    - ✓ Let *Vi*, 2 ≤ *i* ≤ *N*, be the concatenation of *Vi*-1 and the value of the first column of *Ri*, according to the General Rules of Subclause 10.12, “Concatenation of two XML values”.
    - ✓ The result is *VN*.

XMLAgg()

(ens. éléments → forêt éléments)

```
SELECT XMLELEMENT("clientS",
          XMLAGG(
             XMLELEMENT("client",
                CLI.cnom )
          )
       )
FROM Clients_rel CLI;
```

! Ici le groupe G concerne la table entière

```
<clientS>
        <client>Dupont</client>
        <client>Martin</client>
        <client>Muller</client>
        <client>Jeannet</client>
</clientS>
```

XMLAgg() + GROUP BY

```
SELECT XMLELEMENT("clientS",
          XMLATTRIBUTES(CLI.Cville AS "ville"),
          XMLAGG(
             XMLELEMENT("client",
                CLI.cnom )
          )
       )
FROM Clients_rel CLI
GROUP BY CLI.Cville;
```

On a 2 groupes concernés

```
<clientS ville="Fribourg">
          <client>Dupont</client>
          <client>Martin</client>
</clientS>
<clientS ville="Vaud">
          <client>Muller</client>
          <client>Jeannet</client>
</clientS>
```

XMLAgg()

```
SELECT XMLELEMENT("Cmde_FRIBOURG",
          XMLATTRIBUTES(
             CMDE.P_id AS id
          ),
          XMLELEMENT("Items",
             (SELECT XMLAGG(
                        XMLELEMENT("ITEM",
                           PROD.PNom
                        )
                     )
              FROM Item_REL IT, Prod_REL PROD
              WHERE IT.ICmde = CMDE.P_id AND
                    IT.IProd = PROD.P_id)
          )
       )
FROM CmdeS_REL CMDE
WHERE CMDE.pcanton = 'FR'
```

```
<Cmde_FRIBOURG id="3">
   <Items>
      <ITEM>Mouse</ITEM>
      <ITEM>Moniteur</ITEM>
      <ITEM>Modem</ITEM>
   </Items>
</Cmde_FRIBOURG>
```

Pour chaque commande de Fribourg regrouper tous les produits qui la concerne

XMLConcat()

- Produit une forêt en concaténant une liste **d'éléments XML**.

XMLConcat (
instance_element_XML
[, instance_element_XML]*
)

XMLConcat()

(ens. éléments → forêt éléments)

```
SELECT
    XMLELEMENT("Prod_nom", PROD.Pnom),
    XMLELEMENT("Prod_prix", PROD.Pprix)
FROM Prod_REL PROD
```

Le résultat a 2 colonnes

XMLELEMENT("Prod_nom", PROD.Pnom)	XMLELEMENT("Prod_prix", PROD.Pprix)
<Prod_nom>Moniteur</Prod_nom>	<Prod_prix>589,50</Prod_prix>
<Prod_nom>Mouse</Prod_nom>	<Prod_prix>50,25</Prod_prix>
<Prod_nom>Clavier</Prod_nom>	<Prod_prix>165,88</Prod_prix>

XMLConcat()

```
SELECT XMLCONCAT(
    XMLELEMENT("Prod_nom", PROD.Pnom),
    XMLELEMENT("Prod_prix", PROD.Pprix)
)
FROM Prod_REL PROD
```

Le résultat a 1 colonne

XMLCONCAT(XMLELEMENT("Prod_nom", PROD.Pnom), XMLELEMENT("Prod_prix", PROD.Pprix))
<Prod_nom>Moniteur</Prod_nom> <Prod_prix>589,50</Prod_prix>
<Prod_nom>Mouse</Prod_nom> <Prod_prix>50,25</Prod_prix>
<Prod_nom>Clavier</Prod_nom> <Prod_prix>165,88</Prod_prix>

XMLConcat() vs XMLAgg

- Les deux produisent une forêt en concaténant une liste **d'éléments XML**.

- XMLagg() est une fonction d'aggrégat!

XMLGen()

- Produit du XML à partir d'un constructeur Xquery.

XMLGen (
xquery-constructor-with-substitution-variable (ayant la forme"{$name}"),

content-expression [AS variable-name],...)

XMLGEN très proche d'un constructeur XQuery

XMLGen()

```
SELECT XMLGEN( '<a>{$x}</a>', 1 AS x );
```

```
<a>1</a>
```

XMLGen()

(colonne → arbre)

```
SELECT
    XMLGEN(<Produit id={$p_id}>
           {$Prod_nom} </Produit>,
           p_id, p_nom as Prod_nom) as Prod
FROM Prod_REL
```

Prod
<Produit id="1">Moniteur</Produit>
<Produit id="2">Mouse</Produit>
<Produit id="3">Modem</Produit>
<Produit id="4">Clavier</Produit>

select * from prod_rel;

P_ID	P_NOM	PPRIX	PTAXE
1	Moniteur	589.5	
2	Mouse	50.25	
3	Modem	250.25	48
4	Clavier	165.88	

Exercice-oracle XMLCOMMENT

```
XMLELEMENT ( NAME = "emp",
      XMLCOMMENT ( "Example 1" ),
      XMLATTRIBUTES
        ( EMP_ID AS "id" ),
      XMLELEMENT ( NAME = "name", NAME ),
      XMLELEMENT ( NAME = "sal", SALARY)
)
```

Solution

```
XMLELEMENT ( NAME = "emp",
      XMLCOMMENT ( "Example 1" ),
      XMLATTRIBUTES
        ( EMP_ID AS "id" ),
      XMLELEMENT ( NAME = "name", NAME ),
      XMLELEMENT ( NAME = "sal", SALARY)
)
```

```
<emp id="473">
<!-- Example 1 -->
  <name>toto</name>
  <sal>3500.00</sal>
</emp>
```

SQL2003: SQL/XML (Part. 14)

- Un nouveau type: XML.
- Des "publishing functions"
- Des opérateurs
- Un prédicat
- Règles de mapping de SQL à XML

Annexe: Opérateurs SQL/XML

Fonction	Effet
XmlParse()	Génère une *valeur* XML à partir d'une *chaîne de caractères SQL*
XmlSerialize()	Rend une valeur de type« *chaîne* de caractères» SQL à partir d'une valeur XML
XmlRoot()	Permet d'ajouter le prolog XML
IS DOCUMENT	Permet de savoir si on a un document XML (racine unique)

XMLParse()

(Chaine SQL→ XML)

- Génère une *valeur* XML à partir d'une *chaîne de caractères SQL*

XMLParse ({ DOCUMENT | CONTENT }
<string value expression>
[{ PRESERVE | STRIP } WHITESPACE])

- Si DOCUMENT alors on doit avoir une racine unique.

XMLParse()

(Chaine SQL→ XML)

```
INSERT INTO employees ( id, xvalue)
VALUES (102,
        XMLPARSE (DOCUMENT
            '<Emp> Dupont toto </Emp>'))
```

```
INSERT INTO employees ( id, xvalue)
VALUES (102,
        XMLPARSE (CONTENT 'Dupont toto'))
```

Permet l'insertion/modification des colonnes XML

XMLSerialize()

(XML→ Chaine SQL)

- Rend une valeur de type« *chaîne* de caractères» SQL à partir d'une valeur XML

XMLSerialize ({ DOCUMENT | CONTENT }

<XML value expression> [AS <datatype>])

datatype pouvant être *Varchar2* ou *CLOB*

*XMLSerialize()

(XML→ Chaine SQL)

```
XMLSERIALIZE( DOCUMENT
   XMLPARSE (DOCUMENT '<Emp> John Smith </Emp>')
   AS VARCHAR(100))
```

Qui donne `<Emp> John Smith </Emp>`

si pas de encoding spécifier sinon on peut avoir

```
<?xml encoding="UTF-8" version="1.0"?>
<Emp> John Smith </Emp>
```

XMLSerialize()

```
XMLSERIALIZE( DOCUMENT
   XMLPARSE (DOCUMENT '<Emp> John Smith </Emp>')
   AS VARCHAR(100))
```

(XML→ Chaine SQL)

(Chaine SQL→ XML)

Qui donne `<Emp> John Smith </Emp>`

XMLRoot()

- Permet d'ajouter le prolog XML

XMLROOT (<XML value expression> ,
VERSION {<string value expression>|
NO VALUE }
[, STANDALONE { YES | NO | NO VALUE }]

XMLRoot ()

```
XMLROOT(
          XMLELEMENT (
              NAME Elt1,
              XMLATTRIBUTES ('val' AS name, 1 + 1 AS num ),
                    XMLELEMENT ( NAME elt2, 'coucou' ) ),
          VERSION '1.0',
          STANDALONE YES))
```

```
<?xml version='1.0'standalone='yes'?>
<elt1 name='val' num='2'>
  <qux>foo</qux>
</elt1>
```

SQL2003: SQL/XML (Part. 14)

- Un nouveau type: XML.
- Des "publishing functions"
- Des opérateurs
- Un prédicat
- Règles de mapping de SQL à XML

IS DOCUMENT

- Permet de savoir si on a un document XML (racine unique)

<XML_value_expr> **IS [NOT] DOCUMENT**

IS DOCUMENT

(XML→ Chaine SQL)

```
SELECT XMLSERIALIZE (DOCUMENT xvalue AS CLOB)
FROM employees
WHERE xvalue IS DOCUMENT;
```

SQL2003: SQL/XML (Part. 14)

- Un nouveau type: XML.
- Des "publishing functions"
- Des opérateurs
- Un prédicat
- Annexe: Règles de mapping de SQL à XML

Règles de mapping: exemple

- CHAR_*len*: standard conversion dans SQL/XML pour le type CHAR(*len*) de SQL.

CHAR_50 est définit comme:

```
<simpleType>
 <restriction base="string">
  <length value="50">
 </restriction>
</simpleType>
```

SQL	XML Schema	Restrictions
BOOLEAN	Xsd:boolean	
CHAR	Xsd:string	Xsd:length
VARCHAR	Xsd:string	Xsd:maxLength
BLOB	Xsd:base64binary	Xsd:maxLength
SMALLINT, ...	Xsd:integer	min-maxInclusive
NUMERIC, DEC.	Xsd:decimal	precision, scale
FLOAT	Xsd:float	
DOUBLE	Xsd:double	
DATE	Xsd:date	Xsd:pattern
TIME	Xsd:time	Xsd:pattern
TIMESTAMP	Xsd:dateTime	Xsd:pattern
INTERVAL	Xsd:duration	Xsd:pattern

Synthèse: SQL/XML création de vue

1. Construction d'une vue XML sur des données relationnelle

   → (SQL/XML 1ere partie)

2. Construction d'une vue XML sur des données XML.

   → (SQL/XML 1ere et 2ieme partie)

Dans les 2 cas les données deviennent accessible via du Xquery

Synthèse (1)

Une vue XML sur des données relationnelle:

Personne

Id	Prenom
1	Toto
2	Martin

```
SELECT XMLGEN(
        <E id = "{$Id }">
                { $Prenom }
        </E>) as eleve
FROM Personne
```

eleve
<E id="1">Toto</E>
<E id="2">Martin</E>

Synthèse (2)

Une vue XML sur des données XML:

Personne

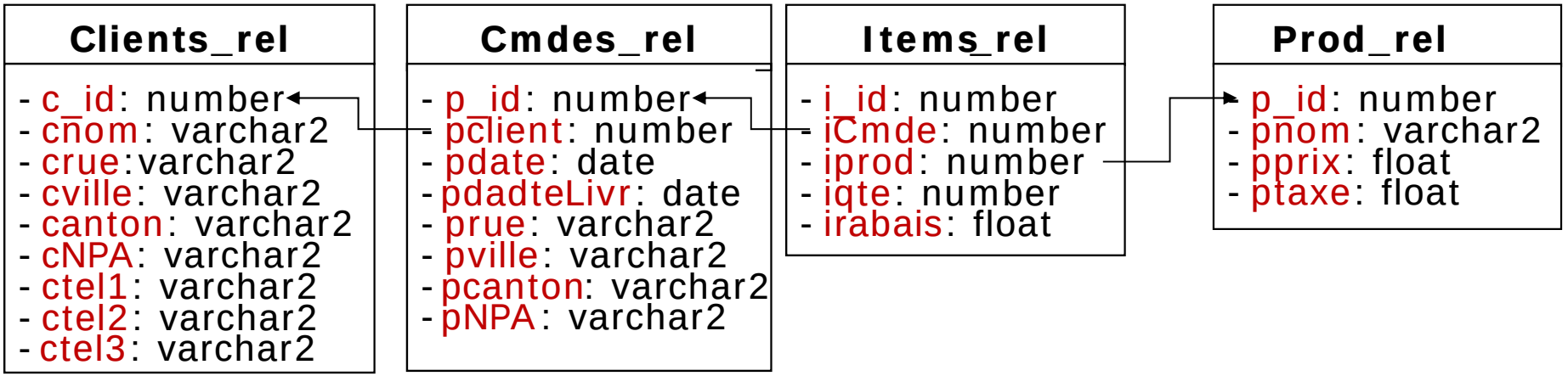
Id	Prenom
1	Toto1 Toto2
2	Martin

```
SELECT Id,
  XMLGEN(<prenom>{$Prenom[1]/text()}</prenom>) as P
FROM  Personne;
```

Id	P
1	<prenom>Toto1</prenom>
2	<prenom>Martin</prenom>

Exercice-exemples

Quelques exemples de l'utilisation de SQL/XML pour créer **des vues XML** (oracle ☺).



Présentation de l'organisation des slides de l'exercice proposé

Pour chaque requête, il y en a 5, on présente:

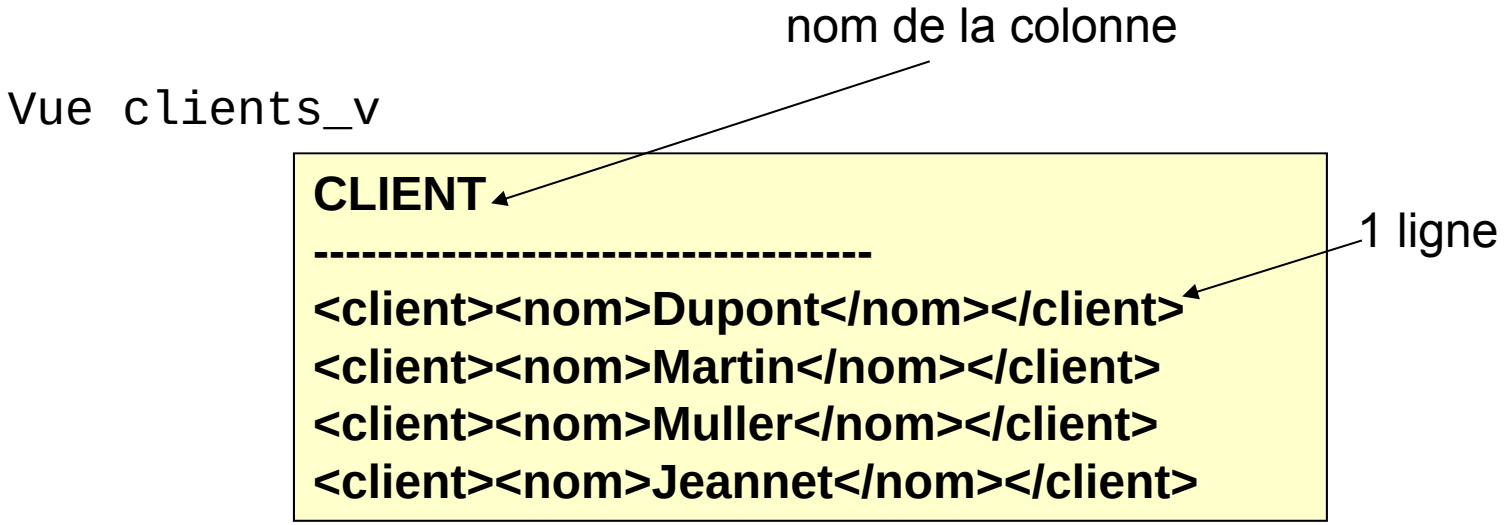
1. D'abord la DTD/XSD du résultat XML attendue
2. Un exemple factuel du contenu de la vue
3. La solution donc la requête SQL/XML

1- Création d'éléments de types simples

```
ELEMENT client(nom)
ELEMENT    nom #PCDATA
```

```
<schema xmlns:xs="...">
   <element name="client" type="TClient"/>
   <complexType name="TClient">
      <sequence>
         <element name="nom" type="xs:string"/>
      </sequence>
   </complexType>
</schema>
```

1- Résultat attendu de la requête



(1) XMLELEMENT

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLELEMENT("nom", C.cnom)
) AS client
FROM Clients_rel C
```

nom de la colonne

OU

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLFOREST(C.cnom AS "nom")
) AS client
FROM Clients_rel C
```

2- Définition d'attributs

```
ELEMENT client(nom)
   ATTLIST client _id CDATA
ELEMENT   nom #PCDATA
```

```
<xs:schema xmlns:xs="...">
   <xs:element name="client" type="TClient"/>
   <xs:complexType name="TClient">
      <xs:sequence>
         <xs:element name="nom" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name= "_id" type="xs:integer"/>
   </xs:complexType>
</xs:schema>
```

2- Résultat attendu de la requête

Vue clients_v

CLIENT

<client _id="1"><nom>Dupont</nom></client>
<client _id="3"><nom>Martin</nom></client>
<client _id="2"><nom>Muller</nom></client>
<client _id="4"><nom>Jeannet</nom></client>

1 ligne

(2) XMLATTRIBUTES

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom")
) AS client
FROM Clients_rel C
```

3- Definintion d'éléments de types complexes: DTD

```
ELEMENT client(nom, adresse)
 ATTLIST client _id CDATA
ELEMENT nom #PCDATA
ELEMENT adresse (rue,ville,canton,NPA)
ELEMENT rue #PCDATA
...
```

3- Definition d'éléments de types complexes: XSD

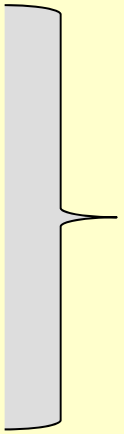
```
<schema xmlns:xs="..." >
   <element name="client" type="Tclient"/>
   <complexType name="Tclient">
      <sequence>
         <element name="nom" type="xs:string"/>
         <element name="adresse" type="TAdresse"/>
      </sequence>
   </complexType>
   <complexType name="TAdresse">
      <sequence>
         <element name="rue" type="xs:string"/>
         <element name="ville" type="xs:string"/>
         <element name="canton" type="xs:string"/>
         <element name="NPA" type="xs:string"/>
      </sequence>
   </complexType>
</schema>
```

3- Résultat attendu de la requête

Vue clients_v

CLIENT

```
<client _id="1">
    <nom>Dupont</nom>
    <adresse>
        <rue></rue>
        <ville>Fribourg</ville>
        <canton>FR</canton>
        <NPA>1700>/NPA>
    </adresse>
</client>
...
<client _id="4">
    <nom>Jeannet</nom>
    <adresse>
        <rue></rue>
        <ville>Vaud</ville>
        <canton>VD</canton>
        <NPA>1905>/NPA>
    </adresse>
</client>
```



1 ligne

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom"),
   XMLELEMENT(“adresse",
      XMLELEMENT("rue", C.Crue),
      XMLELEMENT("ville", C.cville),
      XMLFOREST(C.canton AS “canton"),
      XMLFOREST(C.CNPA AS "NPA")
   )
) AS client
FROM Clients_rel C
```

OU

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom"),
   XMLELEMENT(“adresse",
      XMLFOREST(C.Crue AS "rue",
                C.ville AS "ville",
                C.canton AS “canton",
                C.CNPA AS "NPA")
   )
) AS client
FROM Clients_rel C
```

4- Définition avec eléments multiples : DTD

```
ELEMENT client(nom, adresse, tel*)
 ATTLIST client _id CDATA
ELEMENT nom #PCDATA
ELEMENT adresse (rue,ville,canton,NPA)
ELEMENT rue #PCDATA
...
ELEMENT tel #PCDATA
```

4- Définition avec eléments multiples: XSD

```
<xs:schema xmlns:xs="...">
   <xs:element name="client" type="Tclient"/>
   <xs:complexType name="Tclient">
      <xs:sequence>
         <xs:element name="nom" type="xs:string"/>
         <xs:element name="Adresse" type="TAdresse"/>
         <xs:element name="tel" type="xs:string"
                     maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="_id" type="xs:integer"/>
   </xs:complexType>
   <xs:complexType name="TAdresse">...</xs:complexType>
</xs:schema>
```

4- Résultat attendu de la requête

```
CLIENT
-----------------------------------------------------------------
<client _id="1">
    <nom>Dupont</nom>
    <adresse>
        <rue></rue>
       <ville>Fribourg</ville>
       <canton>FR</canton>
       <NPA>1700>/NPA>
    </adresse>
    <tel></tel><tel></tel><tel></tel>
</client>
…
<client _id="4">
    <nom>Jeannet</nom>
    <adresse>
        <rue></rue>
       <ville>Vaud</ville>
       <canton>VD</canton>
       <NPA>1905>/NPA>
    </adresse>
 <tel></tel><tel></tel><tel></tel>
</client>
```

(4) XMLELEMENT() ou XMLFOREST()

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom"),
   XMLELEMENT("adresse",
      XMLELEMENT("rue", C.Crue),
      XMLELEMENT("ville", C.ville),
         XMLFOREST(C.canton AS “canton
                   C.CNPA AS "NPA")
   ),
   XMLELEMENT("tel", C.Ctel1),
   XMLELEMENT("tel", C.Ctel2),
   XMLELEMENT("tel", C.Ctel3)
) AS client
FROM Clients_rel C
```

OU

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom"),
   XMLELEMENT("adresse",
      XMLELEMENT("rue", C.Crue),
      XMLELEMENT("ville", C.cville),
      XMLFOREST(C.canton AS "canton
                C.CNPA AS "NPA")
   ),
   XMLFOREST(C.Ctel1 AS "tel",
             C.Ctel2 AS "tel",
             C.Ctel3 AS "tel")
) AS client
FROM Clients_rel C
```

5- Avec SQL imbriqué: DTD

```
ELEMENT client(nom, adresse, tel*, Cmde)
 ATTLIST client _id CDATA
ELEMENT nom #PCDATA
ELEMENT adresse (rue,ville,canton,NPA)
ELEMENT rue #PCDATA
...
ELEMENT tel #PCDATA
ELEMENT Cmde(dateC)
 ATTLIST Cmde id CDATA
```

5- Avec SQL imbriqué: XSD

```
<xs:schema xmlns:xs="...">
   <xs:element name="client" type="Tclient"/>
   <xs:complexType name="Tclient">
      <xs:sequence>
         <xs:element name="nom" type="xs:string"/>
         <xs:element name="Adresse" type="TAdresse"/>
         <xs:element name="teletel" type="xs:string" .../>
         <xs:element name="Cmde" type="TCmde"
                     maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="_id" type="xs:integer"/>
   </xs:complexType>
   <xs:complexType name="TAdresse">...</xs:complexType>
   <xs:complexType name="TCmde">
      <xs:sequence>
         <xs:element name="dateC" type="xs:date"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:integer"/>
   </xs:complexType>
</xs:schema>
```

5- Résultat attendu de la requête

```
CLIENT
-----------------------------------------------------------
<client _id="1">
    <nom>Dupont</nom>
    <adresse>
        ....
    </adresse>
    <tel></tel><tel></tel><tel></tel>
   <Cmde id="3">
     <dateC></dateC>
   </Cmde>
</client>
...
<client _id="4">
    <nom>Jeannet</nom>
    <adresse>
        ....
    </adresse>
 <tel></tel><tel></tel><tel></tel>
</client>
```

```
CREATE VIEW clients_v AS
SELECT XMLELEMENT("client",
   XMLATTRIBUTES(C.C_id AS "_id"),
   XMLFOREST(C.cnom AS "nom"),
   XMLELEMENT("adresse", ... ),
   XMLFOREST(C.Ctel1 AS "tel", ...),
   (SELECT XMLAGG( XMLELEMENT("Cmde",
       XMLATTRIBUTES(P.P_id AS "_id"),
       XMLELEMENT("dateC", P.pdate)
    ))
    FROM Cmdes_rel P
    WHERE P.Pclient = C.C_id)
) AS client
FROM Clients_rel C
```

Exercice

On dispose de 5 tables:

- *Dept -- les départements*
- *Emps -- les employés de chaque département*
- *Schools -- les écoles de formation de chaque employé*
- *Kids -- les enfants de chaque employé*
- *Equipments -- Les équipement de chaque département*

De ces tables on désire obtenir le xml suivant qui regroupe pour chaque département toutes les informations le concernant.

Ecrire la requête SQL permettant cette transformation.

Sources:
Article: Towars an industrial strength SQL/XML Infrastructure
M. Krishnaprasad, Z. Hua Liu, A. Manikutty, J.-W. Warner, V. Arora
ICDE'05 Pages: 991 - 1000

Résultat attendu de la requête

```
<Dept>
 <DeptInfo DeptNo="10">
  <DeptName>Accounting</DeptName>
     <Location>Building 300</Location>
  </DeptInfo>
 <Employee>
  <Ename>Smith</Ename>
  <Job>VP</Job>
   <kid>
     <KidName>Peter</KidName>
   </kid>
   <school>
     <SchoolName>CMU</SchoolName>
   </school>
   <school>
     <SchoolName>UCSD</SchoolName>
   </school>
 </Employee>
```

```
 <Employee>
   <Ename>Clark</Ename>
   <Job>Manager</Job>
   <kid>
     <KidName>Grace</KidName>
   </kid>
   <kid>
     <KidName>Mark</KidName>
   </kid>
   <school>
     <SchoolName>UCB</SchoolName>
   </school>
 </Employee>
 <Equipment eid="1">
   <Eqpname>Auditor tool</Eqpname>
 </Equipment>
 <Equipment eid="2">
   <Eqpname>financial caculator</Eqpname>
 </Equipment>
</Dept>
```

Résultat de la requête: simili DTD

Dept → *DeptInfo, Employee*, equipement**

DeptInfo → *DeptName, Location*
→ *@DeptNo*

Employe → *Ename, Job, Kid*,school**

Kid → *KidName*

School → *SchoolName*

Equipement → *Eqpname*
→ *@eid*

Dept → DeptInfo, Employee*, equipement*

```
SELECT
   XMLElement("Dept"), -- creation elt dept
      XMLElement("DeptInfo",XMLAttributes(d.deptno as "DeptNo",
        XMLForest(d.dname as "DeptName", d.location as "Location")),
      XMLAgg(XMLElement("Employe", XMLForest(e.ename as "Ename", e.job as "Job"),
        (SELECT XMLAgg(XMLElement("Kid", k.kname))
         FROM kids k WHERE k.empno = e.empno),
        (SELECT XMLAgg(XMLElement("School", s.sname))
         FROM schools s WHERE s.empno = e.empno)))
   FROM emp e WHERE e.deptno = d.deptno),
   (SELECT XMLAgg(XMLElement("Equipement", XMLAttributes(equipid as "eid"),
      equipname as "Eqpname")))
   FROM equipements WHERE equipements.equipdeptid = d.deptno))
FROM depts d
```

DeptInfo → DeptName, Location
→ @DeptNo

-- dept info

-- elts employe

Employe → Ename, Job, Kid*,school*
Kid → KidName
School → SchoolName

Equipement → Eqpname
→ @eid

-- elts equip

```
SELECT
   XMLElement("Dept"), -- creation elt dept
      XMLElement("DeptInfo", XMLAttributes(deptno as "DeptNo",
        XMLForest(dname as "DeptName", loc as "Location")),          } -- dept info
      XMLConcat(
        (SELECT XMLAgg(XMLElement("Employee",
                                  XMLForest(ename as "Ename", job as "Job"),
                                  (SELECT XMLAgg(
                                              XMLElement("kid", XMLForest(name as "KidName")))
                  -- elts kid      FROM kids
                                   WHERE kids.empno = emps.empno),
-- elts                             (SELECT XMLAgg( XMLElement("school",
employe                                                 XMLForest(name as "SchoolName")))
                  -- elts school   FROM schools
                                   WHERE schools.empno = emps.empno) )
         FROM emps
         WHERE emps.deptno = d.deptn ),
        (SELECT XMLAgg(XMLElement("Equipment", XMLAttributes(equipid as "eid"),
                                  XMLForest(equipname as "Eqpname")))          } -- elts equip
         FROM equipments
         WHERE equipments.equipDptid = d.deptno)))
FROM depts d
```

Exercice-suite: vue

On dispose de 5 tables:

- *Dept -- les départements*
- *Emps -- les employés de chaque département*
- *Schools -- les ecoles fréquentées par chaque employé*
- *Kids -- les enfants de chaque employé*
- *Equipments -- Les équipement de chaque département*

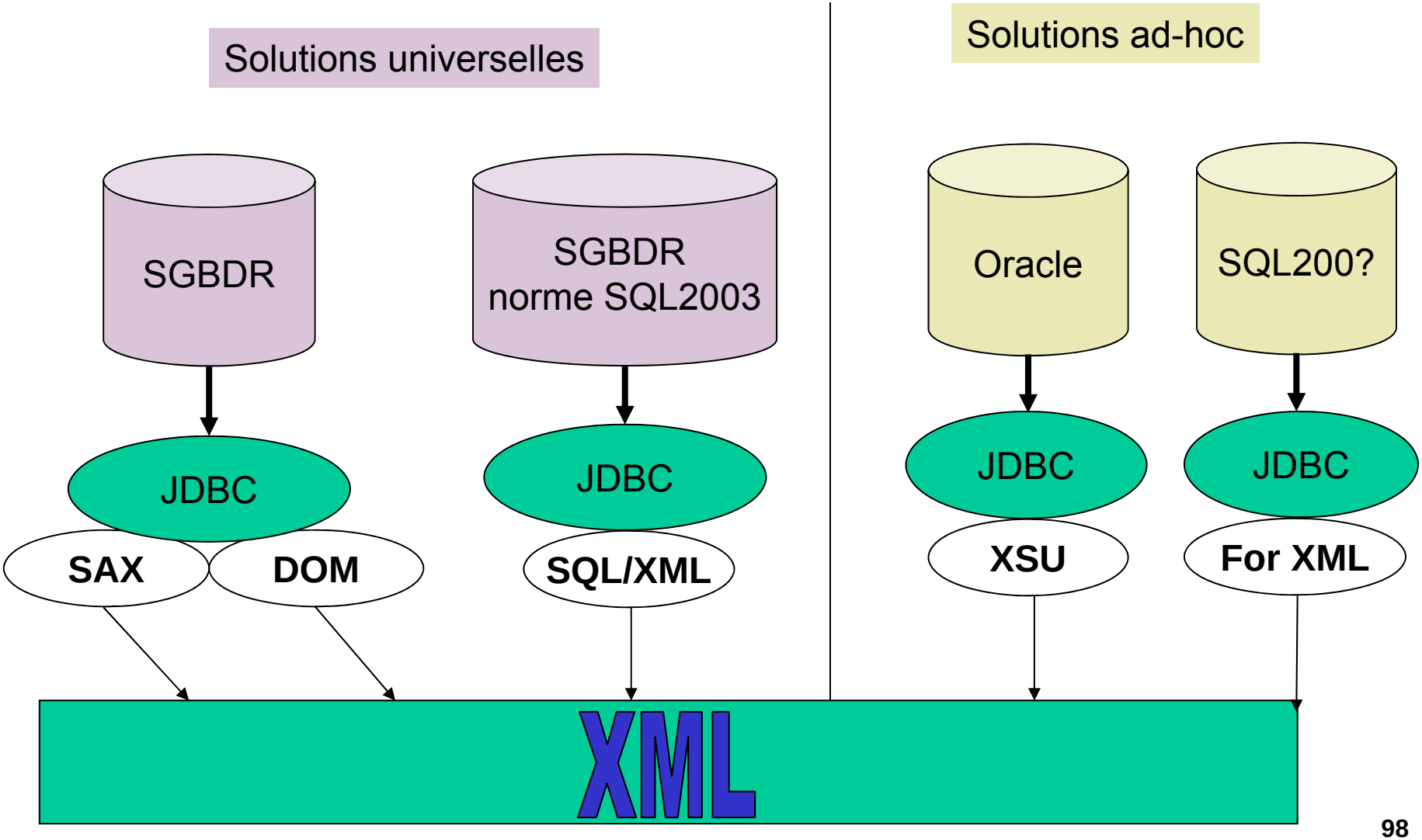
De ces tables on désire obtenir le xml suivant qui regroupe pour chaque département toutes les informations le concernant.

Ecrire la requête SQL permettant cette transformation, et de les conserver dans une vue.

Sources:
Article: Towars an industrial strength SQL/XML Infrastructure
M. Krishnaprasad, Z. Hua Liu, A. Manikutty, J.-W. Warner, V. Arora
ICDE'05 Pages: 991 - 1000

```
CREATE VIEW depts_xml_view (dept_xml) AS
SELECT
   XMLElement("Dept", XMLAttributes(deptno as "DeptNo"), -- creation elt dept
      XMLElement("DeptInfo",                                                    } -- dept info
        XMLForest(dname as "DeptName", loc as "Location")),
      XMLConcat(
        (SELECT XMLAgg(XMLElement("Employee",
                                  XMLForest(ename as "Ename", job as "Job"),
                                  (SELECT XMLAgg(
                                                XMLElement("kid", XMLForest(name as "KidName")))
                  -- elts kid {      FROM kids
-- elts                              WHERE kids.empno = emps.empno),
employe                           (SELECT XMLAgg( XMLElement("school",
                  -- elts school {                  XMLForest(name as "SchoolName")))
                                    FROM schools
                                    WHERE schools.empno = emps.empno) )
         FROM emps
        WHERE emps.deptno = d.deptn ),
        (SELECT XMLAgg(XMLElement("Equipment", XMLAttributes(equipid as "eid"),
                                  XMLForest(equipname as "Eqpname")))                } -- elts
         FROM equipments                                                               equip
        WHERE equipments.equipDptid = d.deptno)))
FROM depts d
```

SQL/XML et ses alternatives?



SQL/XML et ses alternatives?

- Une référence:

"SQL/XML in JDBC Applications: The simple way for Java applications to generate XML from SQL queries using the SQL/XML features of SQL 2003", by Jonathan Robie and Peter Coppens.

http://www.datadirect.com/products/connectsqlxml/docs/sqlxml_whitep.pdf.

Compares the code needed to publish relational data as XML using SQL/XML, JDBC+DOM+SQL, and the proprietary extensions of IBM DB2 UDB, Oracle XSU, and Microsoft SQL Server.

SQL/XML vs Xquery

- XQuery

  ---- **langage** centré XML

- SQL/XML

  ---- **langage** centré SQL

Deux standards: **Complémentaires qui ont été réunis dans XMLQUERY et XMLTABLE**

SQL/XML vs. XQuery

SQL/XML	XQuery
Extension of SQL, part of SQL 2003	New query language being designed in the W3C
SQL-centric	XML-centric
Uses JDBC as Java API	Java API expected 2H 2004 – XQJ (JSR 225)
Less learning for SQL programmers	More natural for XML programmers
Full database support	Full XML support
SQL has 35 years experience with query optimization	XQuery has 5 years experience in query optimization
SQL is a finished language – including full text search, updates	XQuery does not yet have full text search, updates, or many other features
Implementations from Oracle and IBM, not Microsoft	Support from all vendors – Oracle and IBM are much further along on SQL/XML than on XQuery, Microsoft is implementing only XQuery