XML avec Oracle

XML DB est le nom de l'outil d'Oracle destiné à gérer des contenus XML en base de données relationnelle. Cet article fait le point sur cette technologies et en présente les avantages illustré de nombreux exemples.

Par Christian Soutou

Maître de conférences, IUT de Blagnac,
Université de Toulouse Le Mirail

Oracle XML DB

Sommaire

Généralités

XML à été pris en compte il y a une dizaine d'années par Oracle8*i* avec l'apparition de plusieurs paquetages PL/SQL (dont `DBMS_XMLSAVE` et `DBMS_XMLQUERY` qui composaient l'offre XSU : *XML SQL Utility*). Depuis la *Release 1* de la version 9*i*, le type de données `XMLType` est dédié à la gestion de contenus XML. Avec la *Release 2* de la version 9*i*, il est possible d'y associer des grammaires *XML Schema* et de travailler avec le produit *XML Repository*. Depuis la version 10*g*, des efforts ont surtout été réalisés à propos de l'évolution des grammaires *XML Schema*. La version 11*g* propose le mode de stockage *binary XML* un accès par *Web Services*.

XML DB est le nom de la technologie d'Oracle qui permet de gérer du contenu XML en base (stockage, mises à jour et extractions). Alors que la plupart des SGBD natifs XML ne permettent que la persistance, XML DB offre en plus de nombreuses fonctionnalités (contrôle des transactions, intégrité des données, réplication et indexation).

Les deux principales caractéristiques de XML DB sont d'une part l'interopérabilité entre SQL et XML (documents et grammaires), et, d'autre part la gestion de ressources XML dans un contexte multiutilisateurs avec *XML Repository*. Sans XML DB, on peut toutefois stocker du contenu XML (sans pouvoir bien le manipuler par la suite) soit par une API (XML Developer's Kit), soit en utilisant des colonnes de type *Large Object Binary* : `CLOB`, `BLOB`, `BFILE` ou même `VARCHAR`.

Cet article présente les principales fonctionnalités de ce produit, certaines figures sont extraites de la documentation *Oracle XML DB Developer's Guide*.

Comment disposer de XML DB

XML DB est opérationnel si vous avez choisi les options par défaut (*general purpose*) lors de l'installation. Dans le doute, vous pouvez interroger le dictionnaire de données pour constater la présence de l'utilisateur `XDB` (le compte doit être déverrouillé) `SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS` ou de la vue `RESOURCE_VIEW`.
La figure suivante illustre l'architecture générale de XML DB.

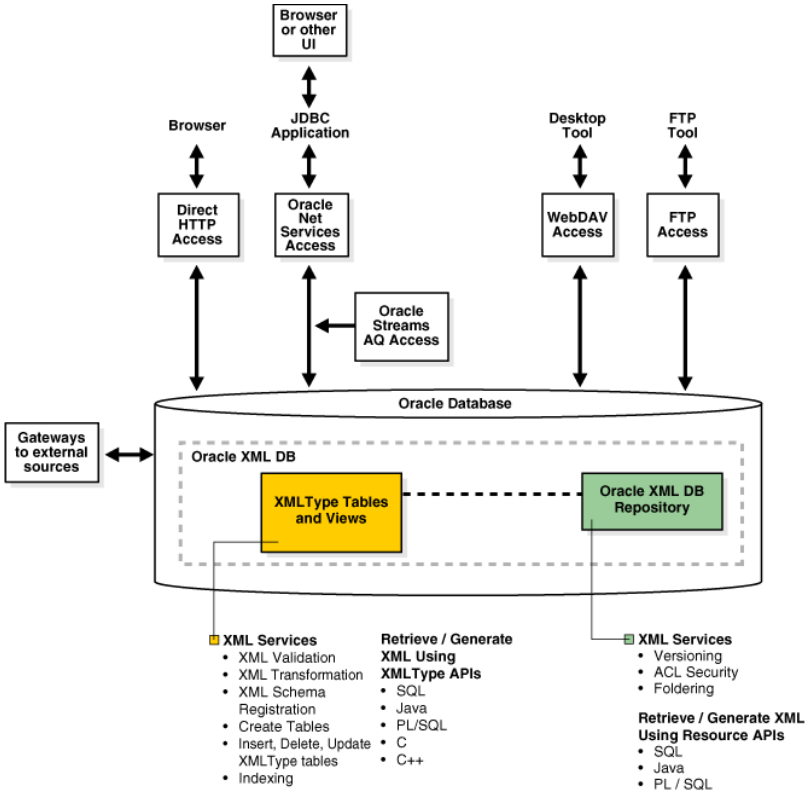


Fig.-1 Architecture de XML DB

Le type de données XMLType

Le type de données `XMLType` fournit de nombreuses fonctionnalités, la plupart relatives à XML (validation de schéma XML et transformation XSL), d'autres qui concernent SQL :

- Définition d'une colonne d'une table (jouant le rôle d'un type de données) ;
- Définition d'une table (jouant le rôle d'un type d'objet) ;
- Déclaration de variables PL/SQL ;
- Appel de procédures PL/SQL cataloguées.

Par défaut, une table (ou colonne) `XMLType` peut contenir n'importe quel document XML bien formé. De plus, le document peut être contraint selon une spécification *XML Schema* avec les avantages suivants :

- Le SGBD s'assure de la validité du document XML avant de le stocker dans une ligne (ou colonne) d'une table.
- Comme le contenu d'une table (ou colonne) est conforme à une structure bien connue, XML DB peut optimiser les requêtes et mises à jour du document XML en fonction du mode de stockage choisi.

Modes de stockage

Suivant le contenu XML à stocker, Oracle fournit différents mécanismes pour enregistrer et indexer les données. Le contenu d'un document XML peut être :

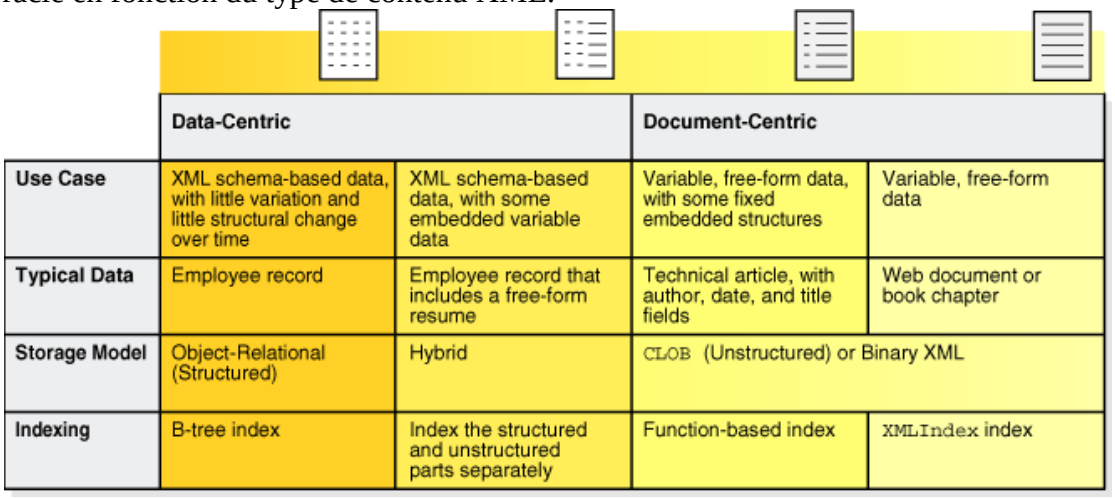
- orienté « données » (*data-centric*), structure régulière des données à granularité fine (la plus petite unité indépendante de donnée est située au niveau d'un élément PCDATA ou d'un attribut), contenant peu ou pas du tout de contenus mixtes (éléments mêlant contenu et balises ainsi qu'attributs).

- orienté « document » (*document-centric*), structure moins régulière des données qui présentent une granularité importante avec de nombreux contenus mixtes. L'ordre dans lequel les éléments apparaissent est très significatif (page Web par exemple).

Plusieurs modes de stockage d'un type `XMLType` sont possibles :

- Stockage non structuré – les données sont enregistrées en tant que CLOB, l'index est de nature *function-based*.
- Stockage binaire (*binary XML*) – les données sont aussi enregistrées dans en tant que `CLOB` mais dans un format binaire conçu pour des données XML, l'index est de nature XMLIndex (les contenus XML doivent être associés à des grammaires *XML Schema*).
- Stockage structuré ou hybride – les données sont enregistrées comme un ensemble d'objets (au sens *object-relational*), l'index est de nature *B-tree*.

A priori, les documents orientés données devrait être stockés selon un mode structuré ou hybride. Le stockage non structuré conviendrait davantage aux autres documents. Quelque soit le mode de stockage, les fonctions d'extraction XML DB produisent les mêmes résultats. Il n'en va pas de même pour la manipulation de La figure suivante résume les préconisations d'Oracle en fonction du type de contenu XML.



	Data-Centric		Document-Centric	
Use Case	XML schema-based data, with little variation and little structural change over time	XML schema-based data, with some embedded variable data	Variable, free-form data, with some fixed embedded structures	Variable, free-form data
Typical Data	Employee record	Employee record that includes a free-form resume	Technical article, with author, date, and title fields	Web document or book chapter
Storage Model	Object-Relational (Structured)	Hybrid	`CLOB` (Unstructured) or Binary XML	
Indexing	B-tree index	Index the structured and unstructured parts separately	Function-based index	`XMLIndex` index

Fig.-2 Préconisations Oracle du mode de stockage

Stockages XMLType

La figure suivante présente les différents accès et les modes de stockage possibles.

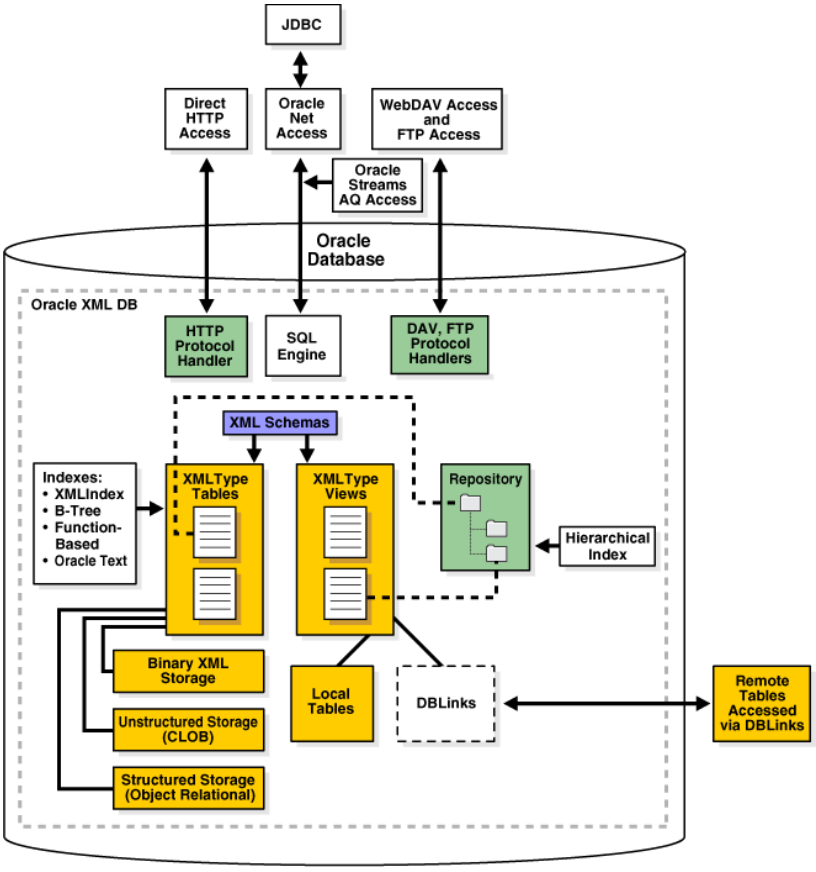


Fig.-3 Stockage avec XMLType

Le tableau suivant résume les points forts et les faiblesses (notés « + » et « - ») de chaque stockage. Le mode *binary XML* semble le plus polyvalent. D'une manière générale, si le contenu XML à stocker ne doit pas être associé à une grammaire, choisissez le format CLOB.

Tableau- Comparatif des modes de stockage

	Structuré (objet-relationnel)	**`CLOB`**	***Binary XML***
Extraction	– (composition structuré de contenus XML)	+	++
Espace disque	++	– (prise en compte de caractères non significatifs et répétition de balises)	+
Flexibilité des données	– (stockage de documents conformes à une grammaire)	+	+
Flexibilité des grammaires *XML Schema*	– (données et grammaires stockés indépendamment, pas plusieurs grammaires pour une table XMLType)	+ (données et grammaires stockées ensemble, pas plusieurs grammaires pour une table XMLType)	++ (données et grammaires stockées ensemble, plusieurs grammaires possibles pour une table XMLType)
Mises à jour (UPDATE, DELETE)	++	– (une modification d'un contenu entraîne la mise à jour de tout le document)	+

	Structuré (objet-relationnel)	**`CLOB`**	***Binary XML***
Requête de type *XPath*	++	–	+
SQL contrainte	+	– (pas possible)	+
Optimisation des opérations XML	+	–	+
Validation après insertion	Partielle (données XML)	Partielle (données XML basées sur des grammaires)	+ validation totale (données XML basées sur des grammaires)

Création d'une table

La syntaxe simplifiée de création d'une table `XMLType` est la suivante

```
CREATE TABLE [schéma.] nomTable OF XMLTYPE
        XMLTYPE [STORE AS
             { OBJECT RELATIONAL | CLOB | BINARY XML }
             [ [ XMLSCHEMA nomXMLSchema ]
                  ELEMENT { élément | nomXMLSchema # élément …} ]
             [ VIRTUAL COLUMNS ( colonne1 AS (expression1),...) ] ];
```

- La clause `OBJECT RELATIONAL` doit être associée à l'option `XMLSCHEMA` qui associe une grammaire *XML Schema*.
- La clause `CLOB` doit être associée soit à la spécification de paramètres de stockage ou à l'option `XMLSCHEMA`.
- La clause `VIRTUAL COLUMNS` est plutôt réservée au mode `BINARY XML`, sert à construire des index ou des contraintes (définies ultérieurement par `ALTER TABLE`).
- En l'absence de clause `STORE AS` le mode de stockage par défaut est structuré (objet-relationnel).

Ces options peuvent permettre également de définir une colonne `XMLType`. On utilise une colonne (aspect étudié plus loin) plutôt qu'une table quand le contenu XML est fortement associé à des données relationnelles (exemple : si on veut stocker la date d'ajout et de dernière mise à jour pour chaque document). La figure suivante illustre les différents conteneurs de données qu'il est possible de mettre en œuvre avec XML DB.

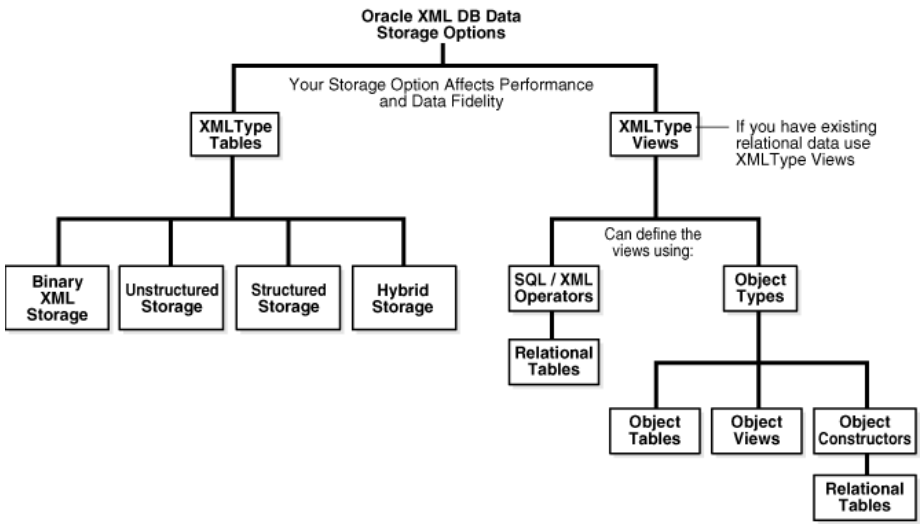


Fig.-4 Conteneur de données

Décrivons à présent une méthodologie de travail pour étudier les différentes fonctionnalités de stockage et de manipulation de documents XML. Nous considèrerons, dans un premier temps une table de stockage structuré. Nous aborderons ensuite les particularités des autres modes de stockage (*binary XML* et `CLOB`).

Répertoire de travail

Si vous n'utilisez pas l'environnement *XML DB Repository*, la création d'un répertoire logique qui référence celui qui contient les documents XML est nécessaire. Pensez également à positionner certaines variables d'environnement SQL*Plus (`SET LONG 10000` et `SET PAGESIZE 100`) pour que vos états de sortie ne soient pas tronqués du fait des valeurs par défaut.

```
CREATE DIRECTORY repxml AS 'C:\sourcesXML';
```

Grammaire XML Schema

Considérons le document `compagnies.xml` présenté au tableau suivant. La grammaire est générée ici à l'aide de l'outil *XML Schema Generator* (http://www.xmlforasp.net/).

Tableau- Exemple de contenu et sa grammaire

Document XML	Grammaire *XML Schema* (compagnies.xsd)

`<?xml version="1.0"`
`  encoding="ISO-8859-1"?>`
`<compagnie>`
`  <comp>AB</comp>`
`  <pilotes>`
`  <pilote brevet="PL-1">`
`  <nom>C. Sigaudes</nom>`
`  <salaire>4000</salaire>`
`  </pilote>`
`  <pilote brevet="PL-2">`
`  <nom>P. Filloux</nom>`
`  <salaire>5000</salaire>`
`  </pilote>`
`  </pilotes>`
`  <nomComp>Air Blagnac</nomComp>`
`</compagnie>`
	`<?xml version="1.0" encoding="utf-16"?>`
	`<xsd:schema attributeFormDefault="unqualified"`
	`  elementFormDefault="qualified" version="1.0"`
	`  xmlns:xsd="http://www.w3.org/2001/XMLSchema">`
	`  <xsd:element name="compagnie" type="compagnieType" />`
	`  <xsd:complexType name="compagnieType">`
	`  <xsd:sequence>`
	`  <xsd:element name="comp" type="xsd:string" />`
	`  <xsd:element name="pilotes" type="pilotesType" />`
	`  <xsd:element name="nomComp" type="xsd:string" />`
	`  </xsd:sequence>`
	`  </xsd:complexType>`
	`  <xsd:complexType name="pilotesType">`
	`  <xsd:sequence>`
	`  <xsd:element maxOccurs="unbounded"`
	`  name="pilote" type="piloteType" />`
	`  </xsd:sequence>`
	`  </xsd:complexType>`
	`  <xsd:complexType name="piloteType">`
	`  <xsd:sequence>`
	`  <xsd:element name="nom" type="xsd:string" />`
	`  <xsd:element name="salaire" type="xsd:decimal" />`
	`  </xsd:sequence>`
	`  <xsd:attribute name="brevet" type="xsd:string" />`
	`  </xsd:complexType>`
	`</xsd:schema>`

Annotation de la grammaire

Il est nécessaire d'annoter la grammaire pour faire correspondre le modèle de documents XML (éléments et attributs) avec les colonnes du SGBD (nom et type). L'espace de nom utilisé par Oracle est `http://xmlns.oracle.com/xdb`. Préfixés par `xdb`, de nombreux éléments sont proposés pour rendre compatible la grammaire avec XML DB et l'enrichir de caractéristiques concernant le SGBD. Le tableau suivant présente les principaux éléments d'annotation d'Oracle. Tous ne sont pas forcément applicables aux différents modes de stockage (le mode `CLOB` est le plus restrictif).

Tableau- Eléments d'annotation

Nom de l'élément	Commentaires et exemple
`xdb:defaultTable`	Nom de la table par défaut générée automatiquement et exploitable avec XML DB Repository.
`xdb:defaultTableSchema`	Nom du schema Oracle.
`xdb:SQLName`	Nom d'une colonne donné à un élément ou attribut XML.
`xdb:SQLType`	Nom du type Oracle.
`xdb:SQLCollType`	Nom du type de la collection.
`xdb:storeVarrayAsTable`	Valeur par défaut `true` (la collection est stocké comme un ensemble de lignes d'une table (*ordered collection table* : OCT). Si `false`, la collection est sérialisée et stockée dans une colonne *LOB*.
`xdb:columnProps`	Précise les caractéristiques des colonnes de la table par défaut. Utile pour déclarer une clé primaire, clé étrangère ou contrainte de vérification.
`xdb:tableProps`	Indique les caractéristiques de stockage de la table par défaut.

Considérons les annotations suivantes apportées à la grammaire initiale. Les types et colonnes sont notés en majuscules d'une part pour mieux les différencier des éléments et attributs XML et, d'autre part, car c'est ainsi qu'Oracle les stocke en interne. Déclarons le code et le nom de la compagnie obligatoires et que si une collection de pilotes existe, elle n'est pas vide (`minOccurs="1"` pour l'élément `pilote`).

Tableau- Grammaire annotée

XML Schema (compagniesannote.xsd)	Commentaires

`<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
`xmlns:xdb="http://xmlns.oracle.com/xdb"`
`xdb:storeVarrayAsTable="true" version="1.0">`
	Espaces de nom.

`<xsd:element name="compagnie" type="compagnieType"/>`	
`<xsd:complexType name="compagnieType"` `xdb:SQLType="COMPAGNIE_TYPE">`	Élément du premier niveau.
`<xsd:sequence>`	
`<xsd:element name="comp" type="compType"` `minOccurs="1" xdb:SQLName="COMP"/>`	Éléments du second niveau
`<xsd:element name="pilotes" type="pilotesType"` `xdb:SQLName="PILOTES"/>`	
`<xsd:element name="nomComp" type="nomCompType"` `minOccurs="1" xdb:SQLName="NOMCOMP"/>`	
`</xsd:sequence>`	
`</xsd:complexType>`	
`<xsd:complexType name="pilotesType"` `xdb:SQLType="PILOTES_TYPE">`	Composition de la collection.
`<xsd:sequence>`	

`<xsd:element minOccurs="1" maxOccurs="unbounded"`
`name="pilote" type="piloteType"`
`xdb:SQLName="PILOTE"`
`xdb:SQLCollType="PILOTE_VRY"/>`	

`</xsd:sequence>`	
`</xsd:complexType>`	
`<xsd:complexType name="piloteType"` `xdb:SQLType="PILOTE_TYPE">`	Élément de la collection.
`<xsd:sequence>`	
`<xsd:element name="nom" type="nomType"` `xdb:SQLName="NOM" xdb:SQLType="VARCHAR2"/>`	

`<xsd:element name="salaire" type="salaireType"`
`minOccurs="0" xdb:SQLName="SALAIRE"`
`xdb:SQLType="NUMBER"/>`	

`</xsd:sequence>`	
`<xsd:attribute name="brevet" xdb:SQLName="BREVET"` `xdb:SQLType="VARCHAR2">`	Précision de la taille de la colonne BREVET.
`<xsd:simpleType>`	
`<xsd:restriction base="xsd:string">`	
`<xsd:minLength value="1"/>`	
`<xsd:maxLength value="4"/>`	
`</xsd:restriction>`	
`</xsd:simpleType>`	
`</xsd:attribute>`	
`</xsd:complexType>`	

```
  <xsd:simpleType name="compType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="6"/> </xsd:restriction>
  </xsd:simpleType>
 <xsd:simpleType name="nomCompType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="40"/> </xsd:restriction>
  </xsd:simpleType>
 <xsd:simpleType name="nomType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="30"/> </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="salaireType">
    <xsd:restriction      base="xsd:decimal">
      <xsd:fractionDigits value="2"/>
      <xsd:totalDigits    value="6"/> </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Typage des autres colonnes.

Enregistrement de la grammaire

La phase suivante enregistre la grammaire dans la base (*repository*) à l'aide de la procédure `REGISTERSCHEMA` du paquetage `DBMS_XMLSCHEMA`. Dé-commentez la première instruction si vous souhaitez relancer à la demande cet enregistrement (après avoir modifié votre grammaire par exemple).

```
BEGIN
-- DBMS_XMLSCHEMA.DELETESCHEMA(
--  'http://www.soutou.net/compagnies.xsd',DBMS_XMLSCHEMA.DELETE_CASCADE_FORCE);
  DBMS_XMLSCHEMA.REGISTERSCHEMA(
    SCHEMAURL => 'http://www.soutou.net/compagnies.xsd',
    SCHEMADOC => BFILENAME('REPXML', 'compagniesannote.xsd'),
    LOCAL => TRUE, GENTYPES => TRUE, GENTABLES => FALSE,
    CSID =>      NLS_CHARSET_ID('AL32UTF8'));
END;
/
```

- `SCHEMAURL` spécifie l'URL logique de la grammaire.
- `SCHEMADOC` référence le fichier lui-même (notez le nom du répertoire logique en majuscules dans la fonction `BFILENAME`).
- `LOCAL` précise que la grammaire est locale (enregistrement dans le répertoire `/sys/schemas/`*`username`*`/`... de *XML DB Repository*). Le cas contraire la grammaire serait globale et se trouverait dans le répertoire `/sys/schemas/PUBLIC/`...).
- `GENTYPES` génère des types objet (dans le cas de stockage *binary XML*, affectez `FALSE`).
- `GENTABLES` permet de générer une table (évite de la créer à part) dont le nom doit se trouver dans la grammaire en tant qu'attribut de l'élément racine `xdb:defaultTable="…"`.
- `CSID` indique le jeu de caractères associé (AL32UTF8 est approprié au type de données XMLType et équivaut au standard UTF-8).

Après la vérification de la grammaire, Oracle génère les types suivants. La colonne `SYS_XDBPD$` est réservée à un usage interne (*positional descriptor*).

Tableau- Structures obtenues

Code SQL	Résultats

`SELECT OBJECT_NAME`
`  FROM USER_OBJECTS`
`  WHERE OBJECT_TYPE='TYPE';`
	`OBJECT_NAME`
	`------------------------`
	`COMPAGNIE_TYPE`

```
                                        PILOTE_VRY
                                        PILOTE_TYPE
                                        PILOTES_TYPE
DESCRIBE COMPAGNIE_TYPE;                COMPAGNIE_TYPE est NOT FINAL
                                        Nom               NULL ?   Type
                                        ----------------- -------- ------------------
                                        SYS_XDBPD$                 XDB.XDB$RAW_LIST_T
                                        COMP                       VARCHAR2(6 CHAR)
                                        PILOTES                    PILOTES_TYPE
                                        NOMCOMP                    VARCHAR2(40 CHAR)
DESCRIBE PILOTES_TYPE                   PILOTES_TYPE est NOT FINAL
                                        Nom               NULL ?   Type
                                        ----------------- -------- ------------------
                                        SYS_XDBPD$                 XDB.XDB$RAW_LIST_T
                                        PILOTE                     PILOTE_VRY
DESCRIBE PILOTE_VRY;                    PILOTE_VRY VARRAY(2147483647) OF PILOTE_TYPE
                                        PILOTE_TYPE est NOT FINAL
                                        Nom                NULL ?   Type
                                        ------------------ ------- ---------------
                                        SYS_XDBPD$                  XDB.XDB$RAW_LIST_T
                                        BREVET                      VARCHAR2(4 CHAR)
                                        NOM                         VARCHAR2(30 CHAR)
                                        SALAIRE                     NUMBER(8,2)
```

Stockage structuré (object-relational)

Une fois la grammaire enregistrée, il est possible de créer explicitement une table (ou colonne) pour stocker des documents XML respectant cette grammaire. Il faut renseigner la grammaire, le nom de l'élément racine et de(s) éventuelle(s) collection(s) *varray*. La table de stockage de la collection est nommée ici `pilote_table`.

```
CREATE TABLE compagnie_OR_xmlschema OF XMLType
       XMLTYPE STORE AS OBJECT RELATIONAL
       XMLSCHEMA "http://www.soutou.net/compagnies.xsd" ELEMENT "compagnie"
       VARRAY "XMLDATA"."PILOTES"."PILOTE"
       STORE AS TABLE pilote_table
             ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)));
```

Par défaut chaque collection (élément XML ayant un attribut `maxOccurs` >1) est sérialisé en tant que LOB (bien adapté pour la gestion de plusieurs documents mais mal adapté à la mise à jour d'éléments particuliers d'une collection donnée). La clause `VARRAY` définit une table de stockage pour chaque collection (bien adaptée à la mise à jour). La directive `XMLDATA` précise un chemin dans une arborescence XML. L'affichage de la structure de cette table rappelle en partie ses caractéristiques.

```
DESCRIBE compagnie_OR_xmlschema;
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------------------
TABLE of SYS.XMLTYPE(XMLSchema "http://www.soutou.net/compagnies.xsd" Element
"compagnie") STORAGE Object-relational TYPE "COMPAGNIE_TYPE"
```

Validation partielle

Bien que la grammaire soit associée à la table, il est toutefois toujours possible de stocker du contenu XML ne respectant que partiellement la grammaire (une compagnie sans pilote ou sans nom, etc.). En revanche, il n'est pas possible d'insérer du contenu XML plus riche (éléments ou attributs) ou dont l'élément racine n'est pas celui défini dans la grammaire (ici compagnie).
Le tableau suivant présente des insertions tout à fait valides par rapport à la grammaire. Dans la première instruction, on retrouve la fonction d'extraction d'un fichier. Le constructeur

XMLType transforme un document XML en `CLOB`, `BFILE` ou `VARCHAR`. Dans la seconde insertion, la fonction `CREATEXML` retourne un type XMLType.

Tableau- Insertions de contenu totalement valide

Code SQL	Commentaires

`INSERT INTO compagnie_OR_xmlschema VALUES`
` (XMLType(BFILENAME('REPXML','compagnie.xml'),`
`  NLS_CHARSET_ID('AL32UTF8')));`
 	
`INSERT INTO compagnie_OR_xmlschema VALUES`
` (XMLTYPE.CREATEXML('<?xml version="1.0"`
`  encoding="ISO-8859-1"?>`
`  <compagnie>`
`  <comp>AC</comp>`
`  <pilotes>`
`  <pilote brevet="PL-3">`
`  <nom>G. Diffis</nom>`
`  <salaire>5000</salaire>`
`  </pilote>`
`  <pilote brevet="PL-4">`
`  <nom>S. Lacombe</nom>`
`  </pilote>`
`  </pilotes>`
`  <nomComp>Castanet Lines</nomComp>`
`</compagnie>'));`
	Insertion du fichier `compagnie.xml`.
	
	Insertion d'un document passé directement en paramètre.

Le tableau suivant illustre des insertions qui ne respectent la grammaire qu'en partie.

Tableau- Insertion de contenu partiellement valide

Code SQL	Commentaires

`INSERT INTO compagnie_OR_xmlschema VALUES`
` (XMLTYPE.CREATEXML(`
`  '<?xml version="1.0" encoding="ISO-8859-1"?>`
`  <compagnie>`
`  <comp>NoPil</comp>`
`  <pilotes></pilotes>`
`  <nomComp>No pilot</nomComp>`
`  </compagnie>'));`
`INSERT INTO compagnie_OR_xmlschema VALUES`
` (XMLTYPE.CREATEXML(`
`  '<?xml version="1.0" encoding="ISO-8859-1"?>`
`  <compagnie>`
`  <comp>PasPil</comp>`
`  <nomComp>Pas de pilote</nomComp>`
`  </compagnie>'));`
`INSERT INTO compagnie_OR_xmlschema VALUES`
` (XMLTYPE.CREATEXML(`
`  '<?xml version="1.0" encoding="ISO-8859-1"?>`
`  <compagnie> <comp>seul!</comp> </compagnie>'));`
	Collection `pilotes` vide.
	
	Absence de collection `pilotes`.
	
	Compagnie sans nom et sans pilote.

La fonction `ISSCHEMAVALID` retourne 1 si l'objet est valide par rapport à sa grammaire, 0 sinon. La requête du tableau suivant est bien utile à vérifier le contenu de la table. Les autres fonctions de la requête seront expliquées plus loin.

Tableau- Détermination des documents invalides

Code SQL	Résultat

`SELECT EXTRACT`
` (c.OBJECT_VALUE,'compagnie/comp') "Comp",`
`  c.ISSCHEMAVALID()`
` FROM compagnie_OR_xmlschema c;`
	`Comp  C.ISSCHEMAVALID()`
	`----------  ----------`
	`<comp>AB</comp>  1`
	`<comp>AC</comp>  1`
	`<comp>NoPil</comp>  0`
	`<comp>PasPil</comp>  0`
	`<comp>seul!</comp>  0`

Le tableau suivant illustre des tentatives d'insertions qui ne respectent pas la grammaire. Différents messages d'Oracle sont retournés en fonction de l'erreur. Pour les ajouts provenant de fichiers extérieurs invalides, les erreurs seraient similaires.

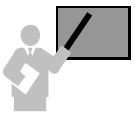
Tableau-Insertion de contenu invalide

Code SQL	Commentaires

`INSERT INTO compagnie_OR_xmlschema VALUES (XMLTYPE.CREATEXML(`
`'<?xml version="1.0" encoding="ISO-8859-1"?>`
`<compagnie>`
`<comp>PB</comp>`
`<pilote brevet="PL-5">`
`<nom>L. Schneider</nom> <salaire>8000</salaire>`
`</pilote> <nomComp>Manque élément pilotes</nomComp>`
`</compagnie>'));`
`ERREUR ORA-30937: Absence de définition de schéma pour 'pilote' (espace de noms '##local') dans le parent '/compagnie'`
	Il manque la collection `pilotes`.

`INSERT INTO compagnie_OR_xmlschema VALUE (XMLTYPE.CREATEXML('<?xml version="1.0" encoding="ISO-8859-1"?>`
`<compagnie>`
`<comp>1234567</comp> <pilotes></pilotes>`
`<nomComp>Trop long le code comp!</nomComp>`
`</compagnie>'));`
`ERREUR ORA-30951: L'élément ou l'attribut (Xpath 1234567) dépasse la longueur maximale`
	Erreur de type du code de la compagnie.

Validation totale



Pour que la validation soit complète (*full compliant*), il faut ajouter à une table de stockage de type objet-relationnelle une contrainte de type `CHECK` ou programmer un déclencheur de type `BEFORE INSERT` (qui réalise la même fonctionnalité).

Les performances d'insertion sont affectées par ce type de validation.

La fonction `XMLISVALID` permet de programmer la contrainte de validité de l'objet en paramètre. Assurez-vous que la table ne contienne pas des enregistrements ne respectant pas la grammaire sinon Oracle retournera l'erreur `ORA-02293: impossible de valider (…) - violation d'une contrainte de contrôle`. Pour supprimer les documents invalides : `DELETE FROM … c WHERE c.ISSCHEMAVALID()=0;`
Une fois cette contrainte existante, seuls les documents vraiment compatibles avec la grammaire pourront être stockés en base. L'erreur retournée, le cas échéant, sera invariablement `ORA-02290: violation de contraintes (…) de vérification`.

Tableau- Mécanismes de validation

Contrainte de vérification	Déclencheur

`ALTER TABLE compagnie_OR_xmlschema`
`ADD CONSTRAINT valide_compagnie`
`CHECK (`**`XMLISVALID`**`(OBJECT_VALUE) = 1);`
	`CREATE TRIGGER valide_compagnie`
	`BEFORE INSERT ON compagnie_OR_xmlschema`
	`FOR EACH ROW`
	`BEGIN`
	`IF (:NEW.OBJECT_VALUE IS NOT NULL) THEN`
	`:NEW.OBJECT_VALUE.`**`SCHEMAVALIDATE`**`();`
	`END IF;`
	`END;`

Le déclencheur de type `BEFORE INSERT` aura l'avantage de renseigner davantage l'erreur. Son code est simple et la fonction `SCHEMAVALIDATE` retourne une exception fournissant des informations précises sur l'erreur comme le montre le code suivant.

Tableau-Insertion de contenu invalide

Code SQL	Commentaires
`INSERT INTO compagnie_OR_xmlschema` `VALUES(XMLTYPE.CREATEXML('<?xml`	`ERREUR ORA-31154: document XML non valide` `ORA-19202: Une erreur s'est produite lors`

version="1.0" encoding="ISO-8859-1"?> <compagnie> <comp>NoPil</comp> <pilotes></pilotes> <nomComp>No pilot</nomComp> </compagnie>'));	du traitement la fonction XML (LSX-00213: Seulement 0 occurrences de l'élément "pilote" ; minimum : 1)… ORA-04088: erreur lors d'exécution du déclencheur 'SOUTOU.VALIDE_COMPAGNIE'

Contraintes

Bien que le mécanisme d'une grammaire *XML Schema* soit puissant, il n'est pas possible de créer une contrainte d'unicité ou de référence. De plus, alors que les contraintes XML concernent individuellement chaque document, une contrainte SQL va permettre d'étendre une restriction à plusieurs documents (peuplant une table ou une colonne).
Deux mécanismes doivent être mis en œuvre : la pseudo-colonne `XMLDATA` qui permet d'adresser une colonne au niveau d'un type XMLType et la table de stockage définie dans la directive `VARRAY` pour opérer sur des collections.

Unicité

Le tableau suivant décrit l'ajout de différentes contraintes. Assurez-vous que les contenus XML respectent chaque contrainte. Si ce n'était pas le cas par exemple pour la première contrainte, Oracle retourne l'erreur `ORA-00001: violation de contrainte unique`. Pour les contraintes de vérification, il sera retourné `ORA-02290: violation de contraintes`.

Tableau- Ajout de contraintes

Code SQL	Contrainte
ALTER TABLE compagnie_OR_xmlschema ADD CONSTRAINT unique_nomcomp **UNIQUE** (**XMLDATA**."NOMCOMP");	Unicité du nom de la compagnie.
ALTER TABLE compagnie_OR_xmlschema ADD CONSTRAINT pk_compagnie_OR **PRIMARY KEY**(**XMLDATA**."COMP");	Clé primaire sur le code compagnie.
ALTER TABLE compagnie_OR_xmlschema ADD CONSTRAINT nn_nomcomp **CHECK** ((**XMLDATA**."NOMCOMP") IS **NOT NULL**);	Non nullité du nom de la compagnie.
ALTER TABLE compagnie_OR_xmlschema ADD CONSTRAINT debut_comp **CHECK** ((**XMLDATA**."COMP") LIKE 'A%' OR (**XMLDATA**."COMP") LIKE 'E%');	Le code de la compagnie débute par la lettre 'A' ou la lettre 'E'.

Intégrité référentielle

Le tableau suivant décrit la mise en œuvre d'une contrainte de référence vers une table relationnelle. Assurez-vous que les données relationnelles recensent toutes les compagnies du contenu XML sinon Oracle retourne l'erreur classique « `ORA-02291: violation de contrainte d'intégrité - clé parent introuvable` ».

Tableau- Intégrité référentielle

Table et données relationnelles	Contrainte
CREATE TABLE compagnie_R1 (codecomp VARCHAR2(6) PRIMARY KEY, budget NUMBER); INSERT INTO compagnie_R1 VALUES ('AB',30090); INSERT INTO compagnie_R1 VALUES ('AC',500000); INSERT INTO compagnie_R1 VALUES ('EA',900000); INSERT INTO compagnie_R1 VALUES ('EF',700000); INSERT INTO compagnie_R1 VALUES ('EG',40000);	ALTER TABLE compagnie_OR_xmlschema ADD CONSTRAINT fk_comp **FOREIGN KEY** (**XMLDATA**."COMP") **REFERENCES** compagnie_R1(codecomp);

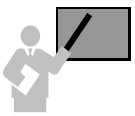
Eléments composés

La directive XMLDATA est nécessaire pour de définir une contrainte sur un élément précis. Le tableau suivant décrit les mécanismes à mettre en œuvre pour rendre l'élément nomAv unique parmi touts les documents qui seront stockés dans la table.

Tableau- Contrainte d'élément composé

Exemple d'un document et partie de sa grammaire	Table associée et contrainte
<avion> <immat>F-GOWW</immat> <**typav**> <**nomAv**>A320</nomAv> <prixAv>40000</prixAv> </typav> </avion>	CREATE TABLE avion_OR_xmlschema OF XMLType XMLTYPE STORE AS OBJECT RELATIONAL XMLSCHEMA "http://www.soutou.net/avions.xsd" ELEMENT "avion";
… <xsd:complexType name="avionType" xdb:SQLType="AVION_TYPE"> <xsd:sequence> <xsd:element name="immat" type="immattype" xdb:SQLName="IMMAT"/> <xsd:element name="**typav**" type="typavType" xdb:SQLName="**TYPEAV_T**"/> </xsd:sequence> </xsd:complexType> <xsd:complexType name="typavType"> <xsd:sequence> <xsd:element name="**nomAv**" type="nomAvtype" xdb:SQLName="**NOMAV**" xdb:SQLType="VARCHAR2"/> …	ALTER TABLE avion_OR_xmlschema ADD CONSTRAINT unique_nomav **UNIQUE** (**XMLDATA**."**TYPEAV_T**"."**NOMAV**");

Collections



Il est possible de définir une contrainte sur un élément ou un attribut d'une collection en utilisant le nom de la table de stockage définie dans la clause STORE AS de la directive VARRAY.

Le code suivant ajoute une contrainte d'unicité sur le numéro de brevet du pilote au niveau de chaque document XML. Le nom de la table de stockage du *varray* est utilisé, de même que la directive NESTED_TABLE_ID qui identifie chaque ligne de la collection.

```
ALTER TABLE pilote_table
      ADD CONSTRAINT un_brevet UNIQUE (NESTED_TABLE_ID, "BREVET");
```

Pour étendre cette contrainte au niveau de tous les documents XML (toutes les compagnies), il ne faut pas mentionner la directive NESTED_TABLE_ID et se placer au niveau de la table de stockage elle-même (qui constitue l'union de toutes les collections).

Tableau- Contrainte sur une collection

Code SQL	Commentaire
ALTER TABLE **pilote_table** ADD **CONSTRAINT** un_brevet_tous **UNIQUE** (BREVET); ALTER TABLE **pilote_table** ADD **CONSTRAINT** ck_salaire **CHECK** (SALAIRE IS NULL OR SALAIRE > 600);	Un pilote n'est embauché que par une seule compagnie. Les salaires sont soit non renseignés soit supérieurs à 600 €.

Extractions

Oracle fournit plusieurs fonctions SQL utiles à l'extraction de types XMLType, citons XMLQUERY, XMLTABLE, XMLEXISTS, et XMLCAST. D'autres fonctions doivent y être associées, citons EXTRACT, EXTRACTVALUE, OBJECT_VALUE et EXISTSNODE.

Éléments (nœuds)

Le tableau suivant présente quelques extractions (code des compagnies, nom des pilotes et détails de tous les deuxièmes pilotes). La fonction EXTRACT admet un paramètre ciblant un nœud et retourne une instance XMLType. La directive OBJECT_VALUE extrait la valeur d'une colonne XMLType (remplace la fonction VALUE utilisée dans les versions antérieures à 10g et associée à la pseudo-colonne SYS_NC_ROWINFO$).

Tableau- ?? Extractions d'éléments XML

Code SQL	Résultats

`SELECT EXTRACT(OBJECT_VALUE,`
`  'compagnie/comp')`
`  FROM compagnie_OR_xmlschema;`
 	
`SELECT EXTRACT(OBJECT_VALUE,`
`      'compagnie/pilotes/pilote/nom')`
`      "Nom de tous les pilotes"`
`       FROM compagnie_OR_xmlschema;`
 	
`SELECT EXTRACT(OBJECT_VALUE,`
`      'compagnie/pilotes/pilote[2]')`
`      "Deuxièmes pilotes"`
`       FROM compagnie_OR_xmlschema;`
	`EXTRACT(OBJECT_VALUE,'compagnie/comp')`
	`------------------------------------`
	`<comp>AB</comp>`
	`<comp>AC</comp>`
	
	`Nom de tous les pilotes`
	`------------------------`
	`<nom>C. Sigaudes</nom>`
	`<nom>P. Filloux</nom>`
	`<nom>G. Diffis</nom>`
	`<nom>S. Lacombe</nom>`
	`Deuxièmes pilotes`
	`---------------------------------`
	`<pilote brevet="PL-2">`
	`  <nom>P. Filloux</nom>`
	`  <salaire>5000</salaire>`
	`</pilote>`
	`<pilote brevet="PL-4">`
	`  <nom>S. Lacombe</nom>`
	`</pilote>`

Attributs et textes

Le tableau suivant présente l'utilisation de la fonction XMLQUERY qui admet en premier paramètre une expression (XMLType ou scalaire SQL) et retourne le résultat de l'évaluation *XQuery* en reliant d'éventuelles variables. La directive PASSING BY VALUE permet de programmer la substitution dans l'expression *XQuery*. La clause AS précise la nature du résultat extrait (ici une chaîne de 9 caractères). Si le résultat est un ensemble vide, la fonction retourne NULL.

La deuxième requête extrait les numéros de brevet mais l'état de sortie n'est pas vraiment satisfaisant. Il faut utiliser la fonction EXTRACTVALUE en la couplant à la directive XMLTABLE (qui construit une table virtuelle à partir de contenu XML). La dernière requête utilise une fonction d'agrégat pour calculer la somme des salaires.

Tableau- Extractions d'attributs et textes XML

Code SQL	Résultats

`SELECT XMLCAST(XMLQUERY(`
`  '$obj/compagnie/comp/text()'`
`    PASSING BY VALUE OBJECT_VALUE AS "obj"`
`  RETURNING CONTENT) AS VARCHAR2(9)) "Code comp"`
`  FROM compagnie_OR_xmlschema;`
 	
`SELECT XMLCAST(XMLQUERY(`
`   '$obj/compagnie/pilotes/pilote/@brevet'`
`  PASSING BY VALUE OBJECT_VALUE AS "obj"`
`  RETURNING CONTENT) AS VARCHAR2(9)) "Brevet"`
`  FROM compagnie_OR_xmlschema;`
	`Code comp`
	`---------`
	`AB`
	`AC`
	
	`Brevet`
	`---------`
	`PL-1PL-2`
	`PL-3PL-4`

```
SELECT EXTRACTVALUE
  (acoll.COLUMN_VALUE,'/pilote/@brevet')
  "Brev" FROM compagnie_OR_xmlschema c,
   XMLTABLE('/compagnie/pilotes/pilote'
            PASSING c.OBJECT_VALUE) acoll;

SELECT SUM(TO_NUMBER(EXTRACTVALUE
  (acoll.COLUMN_VALUE,'/pilote/salaire')))
  "Masse salariale"
  FROM compagnie_OR_xmlschema c,
  XMLTABLE('/compagnie/pilotes/pilote'
         PASSING c.OBJECT_VALUE) acoll;
```

```
Brev
----
PL-1
PL-2
PL-3
PL-4

Masse salariale
---------------
          14000
```

Prédicats d'extraction

Le tableau suivant présente quelques prédicats. La première requête extrait le nom du pilote de numéro 'PL-2'. La fonction `XMLEXISTS` évalue une expression *XQuery* passée en paramètre (compagnies qui s'acquittent d'un salaire valant 5000). La fonction booléenne `EXISTSNODE` vérifie l'expression *XPath* en paramètre (pilotes ayant un salaire inférieur à 5000). La dernière requête compose (à l'aide d'alias, `COLUMNS` et des variables de liens dans la clause `PASSING`) des tables virtuelles utiles à des fonctions d'agrégations (masse salariale de chaque compagnie).

Tableau- Prédicats d'extraction

Code SQL	Résultats

`SELECT`
`  XMLCAST(XMLQUERY('$obj/compagnie/pilotes/pilote`
`  [@brevet="PL-2"]/nom/text()'`
`  PASSING BY VALUE OBJECT_VALUE AS "obj"`
`  RETURNING CONTENT) AS VARCHAR2(15))`
`  "Nom de PL-2" FROM compagnie_OR_xmlschema;`
	`Nom de PL-2`
	`---------------`
	`P. Filloux`

`SELECT COUNT(*) "Nombre de compagnies"`
`  FROM compagnie_OR_xmlschema WHERE`
`  XMLEXISTS('$obj/compagnie/pilotes/pilote/salaire`
`  [number()=5000]'`
`  PASSING OBJECT_VALUE AS "obj");`
	`Nombre de compagnies`
	`--------------------`
	`                   2`

`SELECT`
`  EXTRACT(OBJECT_VALUE,'compagnie/comp') "comp",`
`  EXTRACT(OBJECT_VALUE,'compagnie/nomComp') "nom"`
`  FROM compagnie_OR_xmlschema`
`  WHERE`
`  XMLEXISTS('$obj/compagnie/pilotes/pilote/salaire`
`  [number()<5000]'`
`  PASSING OBJECT_VALUE AS "obj");`
	`comp`
	`----------------------------`
	`nom`
	`----------------------------`
	`<comp>AB</comp>`
	`<nomComp>Air Blagnac</nomComp>`

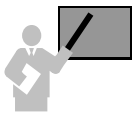
`SELECT`
`  EXTRACT(OBJECT_VALUE,'compagnie/comp') "comp"`
`  FROM compagnie_OR_xmlschema`
`  WHERE EXISTSNODE(OBJECT_VALUE,`
`  'compagnie/pilotes/pilote/salaire`
`  [number()<5000]')=1;`
	`comp`
	`----------------------------`
	`<comp>AB</comp>`

`SELECT acomp.colcmp,SUM(TO_NUMBER(li.sal))`
`  FROM compagnie_OR_xmlschema c,`
`  XMLTABLE('/compagnie' PASSING c.OBJECT_VALUE`
`  COLUMNS`
`  colcmp VARCHAR2(6) PATH 'comp',`
`  pils  XMLType  PATH 'pilotes/pilote') acomp,`
`  XMLTABLE('pilote' PASSING acomp.pils`
`  COLUMNS sal NUMBER PATH 'salaire') li`
`  GROUP BY acomp.colcmp;`
	`COLCMP SUM(TO_NUMBER(LI.SAL))`
	`------ ----------------------`
	`AC                         5000`
	`AB                         9000`

Mises à jour

Il est possible de modifier tout le contenu technique (simple UPDATE) ou seulement une partie (fragment XML). Le code suivant remplace la compagnie de code 'AB' par le contenu du fichier passé en paramètre (sous réserve qu'il soit bien formé, respecte la grammaire et les éventuelles contraintes d'intégrité).

```
UPDATE compagnie_OR_xmlschema
    SET OBJECT_VALUE =
    XMLTYPE(BFILENAME('REPXML','autrecompagnie.xml'),NLS_CHARSET_ID('AL32UTF8'))
    WHERE EXISTSNODE(OBJECT_VALUE, 'compagnie/comp[text()="AB"]') = 1;
```



Le mode de stockage structuré permet l'utilisation de fonctions adaptées à la mise à jour de fragments XML, citons principalement UPDATEXML, INSERTCHILDXML, INSERTXMLBEFORE, APPENDCHILDXML et DELETEXML.

La fonction UPDATEXML(*XMLType_instance, expressionXPath, expression*) met à jour le contenu désigné par le premier paramètre et ciblé par le deuxième paramètre par l'expression du troisième paramètre. Le premier exemple du tableau suivant modifie le salaire d'un pilote.
La fonction INSERTCHILDXML(*XMLType_instance*, *expressionXPath*, *expressionFils*, *expression*) insère un nœud fils (troisième paramètre) à l'emplacement désigné par le deuxième paramètre et valué par le dernier paramètre. La deuxième modification ajoute un pilote pour une compagnie.
La fonction APPENDCHILDXML(*XMLType_instance*, *expressionXPath*, *expression*) insère un nouveau nœud après l'élément désigné par le deuxième paramètre. La troisième modification ajoute l'élément salaire au pilote qui en était démuni.
La fonction DELETEXML(*XMLType_instance*, *expressionXPath*) supprime un ou plusieurs nœuds à l'emplacement désigné par le deuxième paramètre. La dernière modification élimine le premier pilote de la compagnie de code 'AC'.

Tableau- Mises à jour

Code SQL	Résultats (collection pilotes)
UPDATE compagnie_OR_xmlschema SET OBJECT_VALUE = **UPDATEXML**(OBJECT_VALUE, '/compagnie/pilotes/pilote/salaire/text()','6000') WHERE EXISTSNODE(OBJECT_VALUE, '/compagnie/pilotes/pilote/nom[text() = "G. Diffis"]')=1;	<pilote brevet="PL-3"> <nom>G. Diffis</nom> <salaire>**6000**</salaire> </pilote> …
UPDATE compagnie_OR_xmlschema SET OBJECT_VALUE = **INSERTCHILDXML**(OBJECT_VALUE,'/compagnie/pilotes','pilote', XMLType('<pilote brevet="PL-5"> <nom>L. Schneider</nom> <salaire>8000</salaire> </pilote>')) WHERE EXISTSNODE(OBJECT_VALUE, '/compagnie[comp="AC"]') = 1;	<pilote brevet="PL-3"> <nom>G. Diffis</nom> <salaire>6000</salaire> </pilote> <pilote brevet="PL-4"> <nom>S. Lacombe</nom> </pilote> **<pilote brevet="PL-5"> <nom>L. Schneider</nom> <salaire>8000</salaire> </pilote>**
UPDATE compagnie_OR_xmlschema SET OBJECT_VALUE = **APPENDCHILDXML**(OBJECT_VALUE, '/compagnie/pilotes/pilote[2]', XMLType('<salaire>4000</salaire>')) WHERE EXISTSNODE(OBJECT_VALUE, '/compagnie[comp="AC"]') = 1;	… <pilote brevet="PL-4"> <nom>S. Lacombe</nom> **<salaire>4000</salaire>** </pilote> …
UPDATE compagnie_OR_xmlschema SET OBJECT_VALUE = **DELETEXML**(OBJECT_VALUE, '/compagnie/pilotes/pilote[1]') WHERE EXISTSNODE(OBJECT_VALUE, '/compagnie[comp="AC"]') = 1;	<pilote brevet="PL-4"> <nom>S. Lacombe</nom> <salaire>4000</salaire> </pilote> <pilote brevet="PL-5"> <nom>L. Schneider</nom> <salaire>8000</salaire> </pilote>

Vues relationnelles

Les vues relationnelles fournissent un accès classique à du contenu XML. Ce mécanisme intéressera les adeptes de SQL ne maîtrisant pas forcément XML. De plus, ces vues permettent d'« aplatir » les collections pour les réfractaires du modèle objet (il y en a hélas tellement encore !). Les fonctions `EXTRACTVALUE` et `XMLTABLE` qui combinent des expressions *XPath* avec SQL, permettent de définir des vues (*mapping* entre colonnes de la vue et les éléments XML).

Le code suivant déclare une vue relationnelle du contenu XML qui décrit les compagnies. La clause `FROM` contient trois tables : la première héberge les documents XML, la deuxième (virtuelle) compose les colonnes du premier niveau (code et nom de chaque compagnie), la troisième table (virtuelle) définit les éléments de chaque collection (pilotes).

Tableau- Vue relationnelle

Code SQL	Commentaires

`CREATE VIEW compagnie_detail_vue AS`
`  SELECT acomp.colcmp, acomp.nomcmp, ligne.*`
`  FROM compagnie_OR_xmlschema c,`
`  XMLTABLE('/compagnie' PASSING c.OBJECT_VALUE`
`  COLUMNS colcmp VARCHAR2(6)  PATH 'comp',`
`  nomcmp VARCHAR2(20) PATH 'nomComp',`
`  pils XMLType PATH 'pilotes/pilote') acomp,`
`  XMLTABLE('pilote' PASSING acomp.pils`
`  COLUMNS brevetpil VARCHAR2(4) PATH '@brevet',`
`  nompil  VARCHAR2(20) PATH 'nom',`
`  salpil  NUMBER PATH 'salaire') ligne;`
	Colonnes de la vues
	
	Définition de la table virtuelle « maître »
	
	Définition de la table virtuelle « fils »

Une fois cette vue créée, les connaisseurs de SQL y trouveront leur compte quelque soit l'extraction voulue. Le tableau suivant présente quelques extractions classiques (je suppose que la table contient le contenu XML initial).

Tableau- Extraction de contenu XML par vue

Requête	Code SQL

Liste des compagnies.	`SELECT DISTINCT v.colcmp, v.nomcmp`
	`  FROM compagnie_detail_vue v;`
	`COLCMP NOMCMP`
	`------ --------------------`
	`AB  Air Blagnac`
	`AC  Castanet Lines`

Détails des pilotes de la compagnie de code 'AB'.	`SELECT v.brevetpil, v.nompil, v.salpil`
	`  FROM compagnie_detail_vue v`
	`  WHERE v.colcmp = 'AB';`
	`BREV NOMPIL  SALPIL`
	`---- -------------------- ----------`
	`PL-1 C. Sigaudes  4000`
	`PL-2 P. Filloux  5000`

Code des compagnies et détails de chaque pilote.	`SELECT v.colcmp, v.brevetpil, v.nompil, v.salpil`
	`  FROM compagnie_detail_vue v`
	`  ORDER BY 1,2;`
	`COLCMP BREV NOMPIL  SALPIL`
	`------ ---- -------------------- ----------`
	`AB  PL-1 C. Sigaudes  4000`
	`AB  PL-2 P. Filloux  5000`
	`AC  PL-3 G. Diffis  5000`
	`AC  PL-4 S. Lacombe`

Nombre de pilotes salariés par compagnie.	`SELECT v.colcmp, COUNT(v.brevetpil)`
	`  "Nombre de pilotes salariés"`
	`  FROM compagnie_detail_vue v`
	`  WHERE v.salpil IS NOT NULL`
	`  GROUP BY v.colcmp`
	`  ORDER BY 1;`
	`COLCMP Nombre de pilotes salariés`
	`------ --------------------------`
	`AB  2`
	`AC  1`

Création d'une table par annotations

Le code suivant décrit la modification qu'il faudrait faire à la grammaire annotée qui décrit les compagnies pour générer automatiquement la table `compagniedefaut` analogue à la précédente en ce qui concerne les contraintes de clés et la définition de la collection. Seul l'élément racine est à modifier. Pensez à positionner l'option `GENTABLES` à `TRUE` de sorte de générer la table et ses contraintes.

```
<xsd:element name="compagnie" type="compagnieType"
 xdb:defaultTable="COMPAGNIEDEFAUT"
 xdb:columnProps="CONSTRAINT compagniedefault_pk
                            PRIMARY KEY (XMLDATA.COMP),
                  CONSTRAINT fk_compdef FOREIGN KEY (XMLDATA.COMP)
                            REFERENCES compagnie_R1(codecomp)"
 xdb:tableProps="VARRAY XMLDATA.PILOTES.PILOTE
                  STORE AS TABLE pilote_table ((PRIMARY KEY
                            (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))"/>
```

Stockage non structuré (CLOB)

La table suivante contient une colonne de stockage non structurée associée à une grammaire semblable à celle précédemment définie. La spécification du *LOB* est également précisée.

```
CREATE TABLE compagnie_col_xmlschema_CLOB
 (cle VARCHAR2(10) PRIMARY KEY, col_clob_xml XMLType)
  XMLTYPE col_clob_xml STORE AS CLOB
 (TABLESPACE users STORAGE (INITIAL 100K NEXT 50K) CHUNK 800 CACHE LOGGING)
  XMLSCHEMA "http://www.soutou.net/compagnies2.xsd"
  ELEMENT "compagnie";
```

Validation

Le mécanisme de validation est similaire à celui du mode de stockage objet. Le contenu XML doit seulement respecter initialement en partie la grammaire. La validation totale se traite aussi par une contrainte ou par un déclencheur (remplacez la directive `OBJECT_VALUE` par le nom de la colonne `col_clob_xml`).

Contraintes

> Aucune contrainte supplémentaire sur le contenu XML ne peut être mise en œuvre avec SQL.

Extractions

Les requêtes incluant des fonctions `EXTRACT` et `EXISTSNODE` sont bien plus coûteuses que si elles interrogeaient une structure objet (du fait de la construction d'un arbre DOM en mémoire à chaque exécution). Toutes les interrogations précédentes s'adaptent facilement en remplaçant pour chaque requête le nom de la table par `compagnie_col_xmlschema_CLOB c` (avec un alias) et la directive `OBJECT_VALUE` par la colonne `c.col_clob_xml`.

Mises à jour

Le mode de stockage non structuré ne permet la mise à jour qu'au niveau du document entier (`UPDATE` classique).

Ainsi, la modification suivante substitue le document XML relatif à la compagnie de code 'AB' par le contenu XML du fichier passé en paramètre.

```
UPDATE compagnie_col_xmlschema_CLOB c
       SET c.col_clob_xml =
       XMLTYPE(BFILENAME('REPXML','autrecompagnie.xml'),
                         NLS_CHARSET_ID('AL32UTF8'))
       WHERE EXISTSNODE(c.col_clob_xml, 'compagnie/comp[text()="AB"]') = 1;
```

Vues relationnelles

Pour des raisons de performances, il est déconseillé de composer des vues relationnelles de contenu XML stocké en `CLOB`.

Néanmoins, il est possible de définir des vues relationnelles de la même manière que pour le mode de stockage structuré (remplacez la directive `OBJECT_VALUE` par la colonne `col_clob_xml` dans l'exemple).

Stockage non structuré (binary XML)

Le mode de stockage *binary XML* est le plus avancé en ce qui concerne l'encodage du contenu en fonction des grammaires. Considérons dans un premier temps ce type de stockage sans grammaire associée et pour lequel il est nécessaire de définir une ou plusieurs colonnes virtuelles afin de pouvoir mettre en place des contraintes. En effet, le contenu XML n'est pas directement transposé (au niveau de la structure) comme pour le mode de stockage objet-relationnel.

Une colonne virtuelle est basée sur une expression *XPath* qui doit retourner une valeur scalaire par contenu XML (élément ou attribut). La table suivante n'est rattachée à aucune grammaire mais déclare deux colonnes virtuelles. L'encodage de tous les documents utilise le mode neutre (*non-schema-based encoding*).

```
CREATE TABLE compagnie_binaryXML OF XMLType
  XMLTYPE STORE AS BINARY XML
  VIRTUAL COLUMNS
  (vircolcomp    AS (EXTRACTVALUE(OBJECT_VALUE,'/compagnie/comp')),
   vircolnomcomp AS (EXTRACTVALUE(OBJECT_VALUE,'/compagnie/nomComp')));
```

Contraintes

Seules des contraintes fonctionnelles (unique et clé étrangère) sont opérationnelles. Attention, bien qu'Oracle permette de déclarer tout type de contrainte, elles n'auront pas l'effet escompté.

Les contraintes sont définies naturellement à l'aide des colonnes virtuelles comme le montre le tableau suivant.

Tableau- Ajout de contraintes (mode *binary XML*)

Table et données relationnelles	Contrainte
`ALTER TABLE compagnie_binaryXML` `ADD CONSTRAINT unnomcomp_compagnie_binaryXML`	Unicité du nom de la compagnie.

`UNIQUE(vircolnomcomp);`
 	
`ALTER TABLE compagnie_binaryXML`
`ADD CONSTRAINT pk_compagnie_binaryXML`
`PRIMARY KEY(vircolcomp);`
 	
`ALTER TABLE compagnie_binaryXML`
`ADD CONSTRAINT fk_comp_binary`
`FOREIGN KEY (vircolcomp)`
`REFERENCES compagnie_R1(codecomp);`
	Clé primaire sur le code compagnie.
	
	Intégrité référentielle vers une table relationnelle.

Il n'est pas possible d'ajouter une colonne virtuelle à l'aide de l'instruction `ALTER TABLE`.

Extractions

Le mode *binary XML* est le plus performant pour l'extraction de contenus XML. Toutes les requêtes étudiées sur la table structurée (objet-relationnelle) s'écrivent à l'identique en remplaçant simplement pour chaque requête le nom de la table.

Mises à jour

Le mode *binary XML* est moins performant pour la mise à jour de contenus XML que le mode structuré. Néanmoins toutes les mises à jour décrites sur la table objet-relationnelle s'écrivent à l'identique (modifiez seulement le nom de la table).

Vues relationnelles

Les vues relationnelles se définissent de la même manière que pour le mode de stockage structuré.

Options de grammaire

Le mode de stockage *binary XML* est le plus avancé en ce qui concerne l'association de grammaires *XML Schema*.

Une grammaire associée à une table (ou colonne) *binary XML* ne peut être utilisée par une autre table (ou colonne) adoptant le mode de stockage structuré ou `CLOB`. L'inverse est également vrai : l'utilisation d'un autre mode de stockage que *binary XML* pour une grammaire *XML Schema*, impose de n'utiliser cette grammaire que pour le mode de stockage structuré ou `CLOB`.

Le mode de stockage *binary XML* encode le contenu XML en fonction de la grammaire associée (plusieurs grammaires peuvent être associées à une table ou colonne). Possibilité est également donnée d'encoder ou pas du contenu même s'il est associé à une grammaire. Trois choix s'offrent alors à vous (l'option `ALLOW ANYSCHEMA` n'est pas recommandée si votre grammaire est susceptible d'évoluer dans le temps.).

- Encoder le contenu sans tenir compte d'une grammaire (*non-schema-based*). Le contenu XML pourra néanmoins être valide avec la grammaire, sans y être contraint. Toute grammaire associée sera ignorée à propos de l'encodage et le contenu pas automatiquement validé lors d'une insertion ou d'une mise à jour. Il sera également possible de valider explicitement du contenu.

- Encoder le contenu en tenant compte d'une seule grammaire. Tous les documents devront être valides (*full compliant*). Il est possible de préciser que tous les documents ne respectant pas la grammaire (*non-schema-based documents*) peuvent être stockés dans la même colonne.
- Encoder le contenu en tenant compte d'une grammaire parmi plusieurs. Dans ce cas la même grammaire peut être dérivée en plusieurs versions de telle sorte à stocker le contenu en accord avec une version particulière. Les documents ne respectant pas la grammaire peuvent être stockés dans la même colonne.

Tableau- Options de grammaire

Clause SQL	Résultat
`STORE AS BINARY XML`	Encode tous les documents en utilisant le mode neutre (*non-schema-based encoding*).

`STORE AS BINARY XML`
` XMLSCHEMA ...`
`[DISALLOW NONSCHEMA]`
	Encode tous les documents en utilisant la grammaire référencée. L'insertion ou la mise à jour de contenu non valide déclenche une erreur (`ORA-31011 : Echec d'analyse XML`).

`STORE AS BINARY XML`
` XMLSCHEMA ...`
` ALLOW NONSCHEMA`
	Encode tous les documents associés à une grammaire en utilisant cette même grammaire. Encode tous les documents non associés à une grammaire en utilisant le mode neutre.
	L'insertion ou la mise à jour de contenu non valide à sa grammaire déclenche une erreur.

`STORE AS BINARY XML` ` ALLOW ANYSCHEMA`	Chaque grammaire peut être utilisée pour encoder le contenu XML à insérer ou mettre à jour. L'insertion ou la mise à jour de contenu non valide à sa grammaire ou qui ne référence aucune grammaire déclenche une erreur.

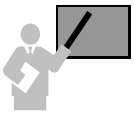
`STORE AS BINARY XML`
` ALLOW ANYSCHEMA`
` ALLOW NONSCHEMA`
	Chaque grammaire peut être utilisée pour encoder le contenu XML à insérer ou mettre à jour. Encode tous les documents non associés à une grammaire en utilisant le mode neutre.
	L'insertion ou la mise à jour de contenu non valide à sa grammaire ou qui ne référence aucune grammaire déclenche une erreur.

Lors de l'enregistrement de la grammaire, positionnez `GENTYPES` à `FALSE` et renseigner l'option `REGISTER_BINARYXML`. La grammaire suivant est sensiblement identique à la première (enlevez les annotations `SQLTYPE` et les types scalaires Oracle comme `VARCHAR2`, etc.).

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    SCHEMAURL => 'http://www.soutou.net/compagnies3.xsd',
    SCHEMADOC => BFILENAME('REPXML','compagniesannotebinXML.xsd'),
    LOCAL => TRUE, GENTYPES => FALSE, GENTABLES => FALSE,
    OPTIONS => DBMS_XMLSCHEMA.REGISTER_BINARYXML);
```

La table suivante est rattachée à une grammaire, déclare une colonne virtuelle et encodera le contenu selon la grammaire.

```
CREATE TABLE compagnie_binaryXML_grammaire OF XMLType
  XMLTYPE STORE AS BINARY XML
   XMLSCHEMA "http://www.soutou.net/compagnies3.xsd" ELEMENT "compagnie"
   DISALLOW NONSCHEMA
  VIRTUAL COLUMNS
          (vircolcomp AS (EXTRACTVALUE(OBJECT_VALUE,'/compagnie/comp')));
```



Le mode de stockage *binary XML* associé à une grammaire fournit une validation complète (*full compliant*).

Autres fonctionnalités

Cette section décrit quelques fonctionnalités qu'il est intéressant de connaître.

Génération de contenus

Plusieurs mécanismes permettent de générer du contenu XML à partir de données relationnelles (tables). Les fonctions les plus intéressantes sont celles de la norme ANSI. Citons `XMLELEMENT` (créé un élément), `XMLATTRIBUTES` (ajoute un attribut à un élément), `XMLFOREST` (créé une arborescence) et `XMLAGG` (peuple une collection).
Considérons les données suivantes (un avion appartient à une compagnie et peut être affrété par plusieurs).

```
Table compagnie_R
CODEC  NOMCOMPA
------ --------------------
AF     Air France
EJ     Easy Jet
AB     Air Blagnac

Table avion_R
NA     TYPAV    CAPACITE PROPRIO
------ ------ ---------- -------
F-GODF A320          170 AB
F-PROG A318          140 AF
F-WOWW A380          490 EJ
```

```
Table affreter_R
NA     CODEC  DATE_A     NB_PASSAGERS
------ ------ ---------- ------------
F-GODF EJ     08-12-2007          120
F-GODF AF     08-12-2007          150
F-PROG AF     08-12-2007          130
F-PROG AF     09-12-2007          110
```

Fig.- Données relationnelles

Le tableau suivant décrit la génération d'une arborescence décrivant les affrètements ordonnés par compagnie.

Tableau 11- Génération de contenus XML

Code SQL	Résultat

`SELECT XMLElement ("Affretement", XMLAttributes(c.codec AS "comp"), XMLForest(c.nomCompa AS "nomComp"), XMLElement("Vols", (SELECT XMLAgg(XMLElement ("Vol", XMLForest(TO_CHAR(af.date_a,'DD/MM/YYYY') AS "date", av.typav AS "avion", af.nb_passagers AS "passagers"))) FROM affreter_R af, avion_R av WHERE af.codec = c.codec AND av.na = af.na))) AS "Contenu_XML" FROM compagnie_R c ORDER BY nomCompa;`	`Contenu_XML`
	`--------------------------------`
	`<Affretement comp="AB">`
	`<nomComp>Air Blagnac</nomComp>`
	`<Vols></Vols>`
	`</Affretement>`
	`<Affretement comp="AF">`
	`<nomComp>Air France</nomComp>`
	`<Vols>`
	`<Vol>`
	`<date>08/12/2007</date>`
	`<avion>A320</avion>`
	`<passagers>150</passagers>`
	`</Vol>`
	`<Vol>`
	`<date>08/12/2007</date>`
	`<avion>A318</avion>`
	`<passagers>130</passagers>`
	`</Vol>`
	`<Vol>`
	`<date>09/12/2007</date>`
	`<avion>A318</avion>`
	`<passagers>110</passagers>`
	`</Vol>`
	`</Vols>`
	`</Affretement>`

	<Affretement comp="EJ"> <nomComp>Easy Jet</nomComp> <Vols> <Vol> <date>08/12/2007</date> <avion>A320</avion> <passagers>120</passagers> </Vol> </Vols> </Affretement>

Vues XMLType

Concernant vos données qui sont stockées dans des tables relationnelles ou objet-relationnelles, les vues `XMLType` permettent de composer du contenu XML contraint ou pas par une grammaire préalablement enregistrée.

Sans grammaire

Le tableau suivant présente la déclaration et l'interrogation de la vue `XMLType` qui fusionne des données de trois tables relationnelles précédentes. La requête de définition inclut en plus un identifiant objet (ici par exemple le nom de la compagnie). Les extractions retournent le nombre de d'affrètements stockés puis le détail d'un affrètement.

Tableau- Vue XMLType

Création de la vue	Interrogations de la vue
CREATE VIEW compagnie_xml OF **XMLType** **WITH OBJECT ID** (SUBSTR(EXTRACTVALUE(OBJECT_VALUE,'/Affretement/nomComp'),1,30)) AS SELECT XMLElement("Affretement", XMLAttributes(c.codec AS "comp"), XMLForest(c.nomCompa AS "nomComp"), XMLElement("Vols", (SELECT XMLAgg(XMLElement("Vol", XMLForest(TO_CHAR(af.date_a,'DD/MM/YYYY') AS "date", av.typav AS "avion", af.nb_passagers AS "passagers"))) FROM affreter_R af, avion_R av WHERE af.codec = c.codec AND av.na = af.na)) AS "Contenu_XML" FROM compagnie_R c;	SELECT COUNT(OBJECT_VALUE) FROM compagnie_xml ; COUNT(OBJECT_VALUE) ------------------ 3 SELECT OBJECT_VALUE FROM compagnie_xml WHERE EXISTSNODE(OBJECT_VALUE, '/Affretement[@comp="EJ"]')= 1; OBJECT_VALUE -------------------------------- <Affretement comp="EJ"> <nomComp>Easy Jet</nomComp> <Vols> <Vol> <date>08/12/2007</date> <avion>A320</avion> <passagers>120</passagers> </Vol> </Vols> </Affretement>

A partir d'un type objet

La fonction `SYS_XMLGEN` génère une instance `XMLType` à partir d'un type objet. Le tableau suivant décrit la création d'un type objet décrivant les affrètements (en utilisant un attribut). La requête de définition de la vue `XMLType` (tous les affrètements de plus de 110 passagers dans un format XML) fait intervenir la fonction `XMLFORMAT` qui compose l'élément racine.

Tableau- Vue XMLType à partir d'un type

Création du type et de la vue	Interrogation de la vue
CREATE TYPE affreter_type AS OBJECT ("@na" CHAR(6), codec VARCHAR(6), date_a DATE, nb_passagers NUMBER); /	SELECT OBJECT_VALUE FROM affreter_view_xml WHERE EXISTSNODE(OBJECT_VALUE, 'Affretement/CODEC[text()="EJ"]')=1;

```
CREATE VIEW affreter_view_xml OF XMLType
  WITH OBJECT ID (EXTRACT(OBJECT_VALUE,
'/Affetement/@na').GETNUMBERVAL()) AS
  SELECT SYS_XMLGEN(affreter_type(a.na,
          a.codec, a.date_a, a.nb_passagers),
        XMLFORMAT('Affretement'))
  FROM affreter_R a
  WHERE a.nb_passagers > 110;
```

```
OBJECT_VALUE
-----------------------------------
<?xml version="1.0"?>
<Affretement na="F-GODF">
 <CODEC>EJ</CODEC>
 <DATE_A>08/12/07</DATE_A>
 <NB_PASSAGERS>120</NB_PASSAGERS>
</Affretement>
```

Association d'une grammaire

Une vue XMLType peut être associée à une grammaire pour contraindre davantage les données extraites. Considérons la simple grammaire caractérisant l'élément avioncomp et définissons une vue XMLType peuplée à partir des données (tous les avions) des tables relationnelles.

Tableau- Structure et grammaire de la vue

Structure XML	Grammaire
<?xml version="1.0" encoding="ISO-8859-1"?> <avioncomp na ="…"> <nomAv> … </nomAv> <capacite> … </capacite> <compav> <comp> … </comp> <nomcomp> … </nomcomp> </compav> </avioncomp>	<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified" version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <xsd:element name="avioncomp" type="avionType"/> <xsd:complexType name="avionType"> <xsd:sequence> <xsd:element name="nomAv" type="xsd:string"/> <xsd:element name="capacite" type="xsd:int"/> <xsd:element name="compav" type="compavType"/> </xsd:sequence> <xsd:attribute name="na" type="xsd:string" /> </xsd:complexType> <xsd:complexType name="compavType"> <xsd:sequence> <xsd:element name="comp" type="xsd:string"/> <xsd:element name="nomcomp" type="xsd:string"/> </xsd:sequence> </xsd:complexType> </xsd:schema>

Le tableau suivant décrit la définition et l'extraction complète de la vue. La fonction GETNUMBERVAL permet d'affecter une valeur numérique à chaque enregistrement extrait et définir ainsi avec WITH OBJECT ID l'identifiant de la vue. On retrouve les fonctions de la norme ANSI qui génèrent du contenu XML.

Tableau- Vue XMLType associée à une grammaire

Création de la vue	Interrogation de la vue
CREATE VIEW avicomp_view_xml OF XMLType XMLSCHEMA "http://www.soutou.net/Avioncomps.xsd" ELEMENT "avioncomp" **WITH OBJECT ID** (EXTRACT (OBJECT_VALUE, '/AvionComp/@immat').GETNUMBERVAL()) AS SELECT XMLElement ("AvionComp", XMLAttributes(av.na AS "immat"), XMLForest(av.typav AS "nomav", av.capacite AS "nbplaces"), XMLElement ("compav", XMLForest(c.codec AS "comp", c.nomcompa AS "nomcomp"))) FROM avion_R av, compagnie_R c WHERE av.proprio = c.codec;	SELECT OBJECT_VALUE FROM avicomp_view_xml; OBJECT_VALUE ------------------------------ <AvionComp immat="F-GODF"> <nomav>A320</nomav> <nbplaces>170</nbplaces> <compav> <comp>AB</comp> <nomcomp>Air Blagnac</nomcomp> </compav> </AvionComp> <AvionComp immat="F-PROG"> <nomav>A318</nomav> <nbplaces>140</nbplaces> <compav> <comp>AF</comp> <nomcomp>Air France</nomcomp> </compav> </AvionComp> <AvionComp immat="F-WOWW"> <nomav>A380</nomav> <nbplaces>490</nbplaces>

	<compav> <comp>EJ</comp> <nomcomp>Easy Jet</nomcomp> </compav> </AvionComp>

Génération de grammaires annotées

La fonction GENERATESCHEMA du paquetage DBMS_XMLSCHEMA permet de générer une grammaire annotée *XML Schema*. Les paramètres sont à inscrire en majuscules. Ils décrivent le nom du schéma d'Oracle qui contient le type objet-relationnel et le nom du type lui même. Générons la grammaire annotée décrivant le type societe_type.

Tableau- Génération d'une grammaire annotée

Code SQL	Résultats (en partie)
CREATE TYPE adresse_type AS OBJECT (nrue CHAR(3), nomrue VARCHAR2(20), ville VARCHAR2(20)) / CREATE TYPE societe_type AS OBJECT (siret VARCHAR2(15), creation DATE, adresse_t adresse_type, effectif NUMBER) / SELECT DBMS_XMLSCHEMA.**GENERATESCHEMA**('SOUTOU','SOCIETE_TYPE') FROM DUAL;	<?xml version="1.0"?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xdb="http://xmlns.oracle.com/xdb" xsi:schemaLocation="http://xmlns.oracle.com/xdb http://xmlns.oracle.com/xdb/XDBSchema.xsd"> <xsd:element name="SOCIETE_TYPE" type="SOCIETE_TYPEType" xdb:SQLType="SOCIETE_TYPE" xdb:SQLSchema="SOUTOU"/> <xsd:complexType name="SOCIETE_TYPEType" xdb:SQLType="SOCIETE_TYPE" xdb:SQLSchema="SOUTOU" xdb:maintainDOM="false"> <xsd:sequence> <xsd:element name="SIRET" xdb:SQLName="SIRET" xdb:SQLType="VARCHAR2"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="15"/> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="CREATION" type="xsd:date" xdb:SQLName="CREATION" xdb:SQLType="DATE"/> <xsd:element name="ADRESSE_T" type="ADRESSE_TYPEType" xdb:SQLName="ADRESSE_T" xdb:SQLSchema="SOUTOU" xdb:SQLType="ADRESSE_TYPE"/> <xsd:element name="EFFECTIF" type="xsd:double" xdb:SQLName="EFFECTIF" xdb:SQLType="NUMBER"/> </xsd:sequence> </xsd:complexType> <xsd:complexType name="ADRESSE_TYPEType" xdb:SQLType="ADRESSE_TYPE" xdb:SQLSchema="SOUTOU" xdb:maintainDOM="false"> <xsd:sequence> <xsd:element name="NRUE" xdb:SQLName="NRUE" xdb:SQLType="CHAR"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="3"/> </xsd:restriction> </xsd:simpleType> </xsd:element> … </xsd:schema>

Il restera à ajouter d'éventuelles annotations qui contraindront ou préciseront le stockage des sociétés. Ce mécanisme fonctionne également pour les collections objet (AS TABLE OF).

Dictionnaire des données

Le dictionnaire des données propose un certains nombre de vues (préfixées par USER pour les objets du schéma courant, ALL pour les objets sur lesquels on a des privilèges et DBA pour tous les objets quelque soit le schéma) qui intéresseront les utilisateurs de XML DB.

Tables XMLType

Au niveau d'un utilisateur, la vue `USER_XML_TABLES` décrit les tables `XMLType` en ce qui concerne le type de stockage et les options de grammaire.

Tableau- Nature des tables XMLType

Code SQL	Résultats

```
SELECT TABLE_NAME,
  XMLSCHEMA, STORAGE_TYPE
  FROM USER_XML_TABLES;
```

```
TABLE_NAME
 --------------------------
XMLSCHEMA                                STORAGE_TYPE
---------------------------------------  --------------
AVION_OR_XMLSCHEMA
http://www.soutou.net/avions.xsd         OBJECT-RELATIONAL

COMPAGNIE_BINARYXML_GRAMMAIRE
http://www.soutou.net/compagnies3.xsd BINARY

COMPAGNIE_OR_XMLSCHEMA
http://www.soutou.net/compagnies.xsd  OBJECT-RELATIONAL
```

```
SELECT TABLE_NAME,
 ELEMENT_NAME,
 ANYSCHEMA, NONSCHEMA
 FROM USER_XML_TABLES;
```

```
TABLE_NAME                     ELEMENT_NAME     ANY NON
------------------------------ ---------------- --- ---
COMPAGNIE_BINARYXML                             NO  YES
AVION_OR_XMLSCHEMA             avion
COMPAGNIE_BINARYXML_GRAMMAIRE compagnie         NO  NO
COMPAGNIE_OR_XMLSCHEMA         compagnie
```

Colonnes XMLType

Sur le même principe, la vue `USER_XML_TAB_COLS` décrit les colonnes `XMLType` en ce qui concerne le type de stockage et la grammaire.

Tableau- Nature des colonnes XMLType

Code SQL	Résultats

```
SELECT COLUMN_NAME,
 XMLSCHEMA, ELEMENT_NAME,
 STORAGE_TYPE
 FROM USER_XML_TAB_COLS;
```

```
COLUMN_NAME       XMLSCHEMA
----------------  -------------------------
ELEMENT_NAME          STORAGE_TYPE
----------------      -----------------
SYS_NC_ROWINFO$   http://www.soutou.net/compagnies3.xsd
compagnie             BINARY
SYS_NC_ROWINFO$   http://www.soutou.net/compagnies.xsd
compagnie              OBJECT-RELATIONAL
SYS_NC_ROWINFO$   http://www.soutou.net/avions.xsd
avion                  OBJECT-RELATIONAL
COL_CLOB_XML      http://www.soutou.net/compagnies2.xsd
Compagnie               CLOB
```

Grammaires *XML Schema*

Sur le même principe, `USER_XML_SCHEMAS` renseigne à propos des grammaires *XML Schema*. La colonne `XMLSCHEMA` de cette vue contient le code complet de la grammaire.

Tableau- Nature des grammaires *XML Schema*

Code SQL	Résultats

```
SELECT SCHEMA_URL, LOCAL,
  BINARY
  FROM USER_XML_SCHEMAS;
```

```
SCHEMA_URL                                        LOC BIN
------------------------------------------------- --- ---
http://www.soutou.net/avions.xsd                  YES NO
http://www.soutou.net/compagnies.xsd              YES NO
http://www.soutou.net/compagnies2.xsd             YES NO
http://www.soutou.net/compagnies3.xsd             YES YES
```

Vues XMLType

Les vues XMLType sont renseignées via USER_XML_VIEWS. La requête suivant extrait les caractéristiques des vues XMLType du schéma courant.

Tableau- Nature des vues XMLType

Code SQL	Résultats
SELECT VIEW_NAME, XMLSCHEMA, ELEMENT_NAME FROM **USER_XML_VIEWS**;	VIEW_NAME XMLSCHEMA ---------- ---------- ELEMENT_NAME ---------- COMPAGNIE_XML AFFRETER_VIEW_XML AVICOMP_VIEW_XML http://www.soutou.net/Avioncomps.xsd avioncomp

XML DB Repository

XML DB Repository est un environnement partagé de contenus (XML ou autre) basé sur le concept de système de gestion de fichiers (répertoires). Les contenus non XML sont stockés en tant que CLOB. L'environnement est compatible avec la norme DAV *(Distributed Authoring and Versioning*) et utilise un serveur WebDAV.
Toutes les informations de cet environnement sont stockées dans le schéma de l'utilisateur XDB (initialement verrouillé à l'installation de la base). Une action dans la console ou l'exécution du script catqm.sql situé dans *ORACLE_HOME*\rdbms\admin rend opérationnel cet utilisateur.

Interfaces

Les moyens de travailler avec *XML DB Repository* sont divers :

- protocoles http(S), WebDAV ou FTP pour les ajouts, mises à jour et suppressions de contenus ;
- directement via les tables en utilisant SQL et le paquetage XML_XDB ;
- gestion de versions en utilisant le paquetage XML_XDB_VERSION ;
- par l'API Java (*Content Connector*).

Configuration

Pour configurer cet environnement, contrôlez que votre instance est bien associée à un service (La commande lsnrctl status doit retourner « Le service "*instanceOracle*XDB" comporte 1 instance(s)… ». Affecter un numéro de port (ici 8080) dans l'élément <http-port> du fichier xdbconfig.xml.11.0 du répertoire ORACLE_HOME\xml.
L'écran suivant illustre un accès http (accès réglementé par un compte et un mot de passe Oracle) à l'arborescence principale de *XML DB Repository*.

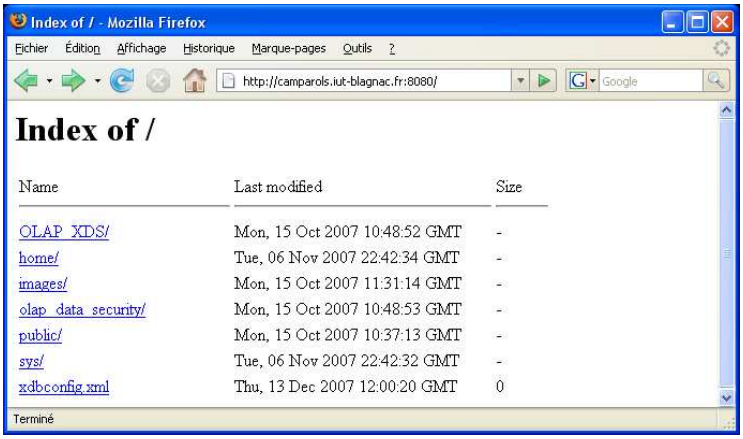


Fig.-5 Accès par http

L'écran suivant illustre le même accès par l'explorateur Windows. Vous pouvez également définir cet accès au niveau de vos favoris réseaux. Une fois connecté, vous pouvez créer des répertoires et déposer vos documents par glisser-déposer.

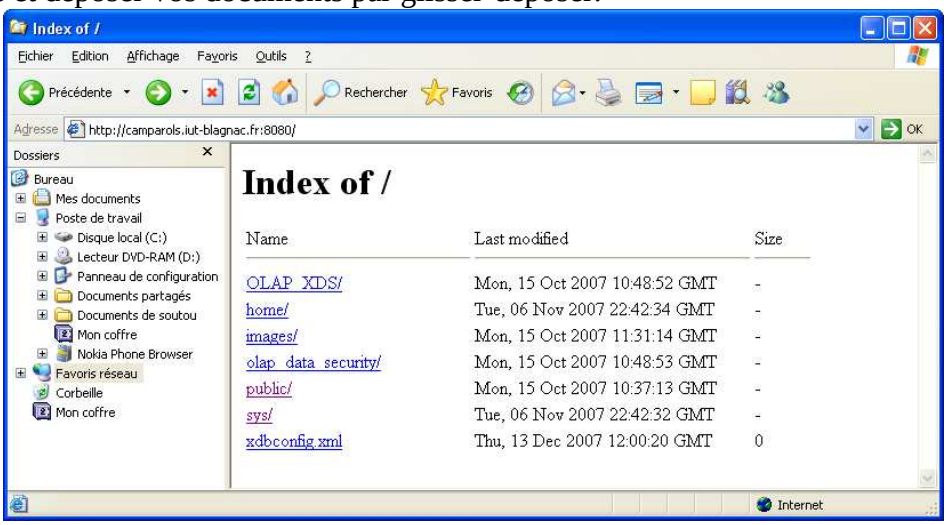


Fig.-6 Accès par l'explorateur Windows

La figure suivante décrit l'arborescence qu'Oracle préconise afin d'utiliser correctement le système de gestion de fichiers de *XML DB Repository*.

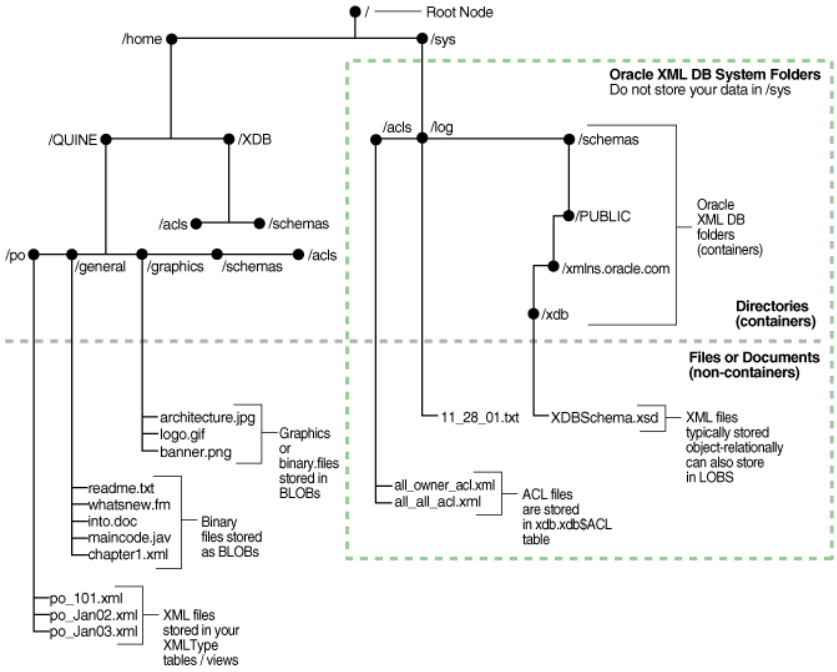


Fig.-7 Arborescence du système de gestion de fichiers

Les fichiers grammaire qui ont été créés par l'instruction `REGISTERSCHEMA` se trouvent dans l'arborescence `sys/schemas` suivi du nom d'utilisateur et de l'URL logique de chaque grammaire.

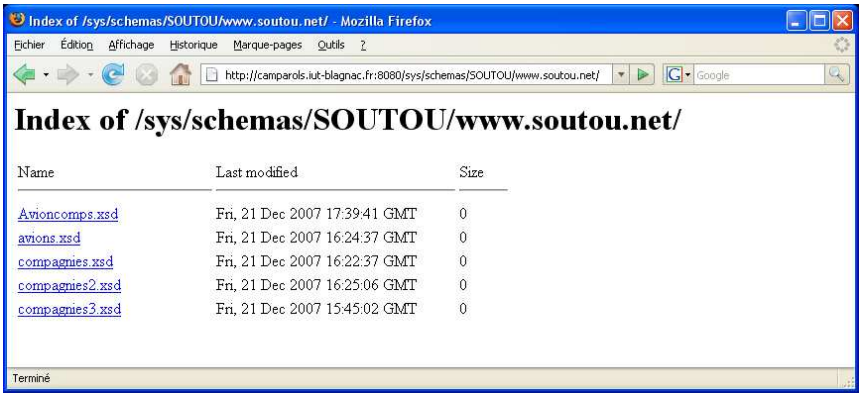


Fig.-8 Liste des grammaires

Paquetage XML_XDB

Le paquetage `XML_XDB` propose de nombreuses fonctions pour manipuler le système de gestion de fichiers, citons :

- `CREATEFOLDER` pour créer un répertoire ;
- `DELETERESOURCE` supprime une ressource (document ou répertoire) ;
- `EXISTSRESOURCE` qui teste l'existence d'un répertoire ;
- `CREATERESOURCE` pour créer une ressource (document ou répertoire).

Le code suivant dépose la ressource (document `A-Faire.txt` contenant trois lignes de texte et situé dans le répertoire référencé par le nom logique `REPXML`) dans l'arborescence `/home/SOUTOU/general`. Bon, c'est vrai, quelques clics auraient pu faire aussi l'affaire !

Tableau- Dépose d'une ressource

Code SQL	Commentaires

DECLARE
  v_resultat BOOLEAN;
BEGIN
 IF (NOT DBMS_XDB.**EXISTSRESOURCE**('/home/SOUTOU')) THEN
  v_resultat := DBMS_XDB.**CREATEFOLDER**('/home/SOUTOU');
 END IF;
IF (NOT DBMS_XDB.**EXISTSRESOURCE**('/home/SOUTOU/general')) THEN
  v_resultat:=DBMS_XDB.**CREATEFOLDER**('/home/SOUTOU/general');
 END IF;
IF (DBMS_XDB.**EXISTSRESOURCE**('/home/SOUTOU/general/note.txt')) THEN
  DBMS_XDB.**DELETERESOURCE**('/home/SOUTOU/general/note.txt',4);
 END IF;
v_resultat := DBMS_XDB.**CREATERESOURCE**(
  '/home/SOUTOU/general/note.txt',
  BFILENAME('REPXML','A-Faire.txt'),NLS_CHARSET_ID('AL32UTF8'));
 COMMIT;
END;
	Création de `/home/SOUTOU` si nécessaire.
	Création du sous répertoire `general` si nécessaire.
	Suppression de la ressource.
	Transfert de la ressource.

Accès par SQL

Deux vues permettent d'accéder aux ressources contenues dans *XML DB Repository* : `RESOURCE_VIEW` et `PATH_VIEW`. Toutes deux possèdent une colonne virtuelle nommée `RES` de type XMLType permettant d'extraire et de mettre à jour des contenus en utilisant la notation pointée (alias SQL). Chaque ligne de la vue `RESOURCE_VIEW` concerne un unique chemin dans l'arborescence. Chaque ligne de la vue `PATH_VIEW` concerne une unique ressource. Une ressource peut être associée à plusieurs chemins de répertoires (liens).

Vue `RESOURCE_VIEW`

Cette vue est composée de trois colonnes :

- `RES` (`XMLType`) : décrit une ressource d'un répertoire ;
- `ANY_PATH` (`VARCHAR2`) : indique un chemin (absolu) d'une ressource
- `RESID` (`RAW`) contient l'identifiant d'une ressource.

La grammaire de la colonne `RES` (`XDBResource.xsd`) de la vue `RESOURCE_VIEW` se situe dans l'arborescence `/sys/schemas/PUBLIC/xmlns.oracle.com/xdb/`. En considérant certains éléments de cette grammaire, des requêtes peuvent être composées pour extraire tout ou partie du contenu stocké dans les ressources. Le tableau suivant décrit les principaux éléments définis dans la grammaire `XDBResource.xsd`.

Tableau- Parties de la grammaire de la vue `RESOURCE_VIEW`

Requêtes SQL	Commentaires

<Resource
  xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
  Container="…"…>
<**CreationDate**> … </CreationDate>
<**ModificationDate**> … </ModificationDate>
<**DisplayName**> … </DisplayName>
<**Language**> … </Language>
<**CharacterSet**> … </CharacterSet>
<**ContentType**> … </ContentType
<**ACL**> … </ACL>
 	
<**Owner**> … </Owner>
<**Creator**> … </Creator>
<**LastModifier**> … </LastModifier>
 	
<**SchemaElement**> … </SchemaElement>
<Contents>
<**text**> … </text> </Contents>
	Répertoire ou fichier.
	Dates de création et de modification de la ressource.
	Nom du fichier.
	Langage, jeu de caractères et type du contenu.
	Autorisations (*Acces Control Lists*)
	Compte Oracle propriétaire de la ressource, créateur et dernier utilisateur ayant modifié la ressource.
	Element de la ressource.
	Contenu de la ressource.

```
</Resource>
```

Les fonctions EQUALS_PATH (trouve une ressource située dans un répertoire) et UNDER_PATH (retourne les sous répertoires d'un répertoire donné) doivent également être utilisées. Le tableau suivant présente quelques extractions. La fonction CONVERT du paquetage DBMS_XMLGEN convertit un contenu XML selon un format donné et retourne un CLOB.

Tableau- Extractions SQL

Requêtes SQL	Commentaires et résultats

`SELECT XDBURIType`
`  ('/home/SOUTOU/general/note.txt').GETCLOB()`
`  "A faire..." FROM DUAL;`
	Extraction sous la forme d'un CLOB du contenu d'une ressource.
	`A faire...`
	`----------`
	`- Finir article Programmez`
	`- Contacter l'éditeur Vuibert`
	`- Acheter 3 baguettes`

`SELECT COUNT(*) FROM RESOURCE_VIEW rv`
`  WHERE UNDER_PATH`
`  (rv.RES,'/home/SOUTOU/general') = 1;`
	Nombre de ressources dans un répertoire donné.
	`COUNT(*)`
	`----------`
	`2`

`SELECT ANY_PATH FROM RESOURCE_VIEW`
`  WHERE EXTRACTVALUE(RES,'/Resource/DisplayName')`
`  LIKE '%.txt';`
	Chemins contenant une ressource d'extension 'txt'.
	`ANY_PATH`
	`----------`
	`/home/SOUTOU/general/note.txt`

`SELECT EXTRACT(rv.RES,'/Resource/CreationDate')`
`  "Date création"`
`  FROM RESOURCE_VIEW rv`
`  WHERE EQUALS_PATH`
`  (rv.RES,'/home/SOUTOU/general/note.txt')=1;`
	Date de création de la ressource note.txt.
	`Date création`
	`----------`
	`<CreationDate xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd">2007-12-22T17:48:09.015000</CreationDate>`

`SELECT ANY_PATH FROM RESOURCE_VIEW`
`  WHERE UNDER_PATH(RES, 2,'/home/SOUTOU') = 1`
`  AND EXISTSNODE(RES,`
`  '/Resource[@Container="true"]')=1;`
	Répertoires seuls (à partir de /home/SOUTOU, profondeur maxi 2).
	`ANY_PATH`
	`----------`
	`/home/SOUTOU/general`
	`/home/SOUTOU/xsd`

`SELECT DBMS_XMLGEN.CONVERT(EXTRACT`
`(rv.RES,'/Resource/Contents/text/text()','xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"').GETCLOBVAL(),1) "Contenu"`
`FROM RESOURCE_VIEW rv`
`WHERE EQUALS_PATH(rv.RES,`
`  '/home/SOUTOU/general/note.txt')=1;`
	Extraction du contenu d'une ressource en utilisant la grammaire.
	`Contenu`
	`----------`
	`- Finir article Programmez`
	`- Contacter l'éditeur Vuibert`
	`- Acheter 3 baguettes`

`SELECT alias.nom`
`  FROM RESOURCE_VIEW rv, XMLTABLE(XMLNAMESPACES`
`('http://xmlns.oracle.com/xdb/XDBResource.xsd'`
`  AS "r"),`
`'/r:Resource/r:Contents/compagnie/pilotes/pilote'`
`PASSING rv.RES`
`COLUMNS nom VARCHAR2(15) PATH 'nom') alias`
`  WHERE EQUALS_PATH(`
`  rv.RES,'/home/SOUTOU/general/compagnie.xml')=1;`
	Extraction du nom des pilotes.
	`NOM`
	`----------`
	`C. Sigaudes`
	`P. Filloux`

Contenus XML basés sur une grammaire

L'accès au contenu de documents associés à une grammaire (*schema-based XML documents*) peut se programmer de deux manières :

- Par la vue `RESOURCE_VIEW` (comme précédemment même si le contenu n'est contraint par aucune grammaire ;
- Par jointure entre la table créée par défaut au niveau de la grammaire et la vue `RESOURCE_VIEW`.

Annotez votre grammaire avec les attributs `xdb:schemaURL = "votreURL.xsd"` et `xdb:defaultTable="NOM_TABLE_EN_MAJUSCULES"` avant de la stocker dans un répertoire de *XML DB Repository*.

Enregistrez ensuite votre grammaire avec la version `CLOB` de la fonction `REGISTERSCHEMA` en utilisant la directive `XDBURITYPE`.

Le tableau suivant présente l'annotation de la grammaire (fichier `compagniesRepository.xsd`) puis le stockage et l'enregistrement de cette grammaire dans *XML DB Repository* (on suppose que le répertoire `/home/SOUTOU/xsd/` existe).

Tableau- Gestion de la grammaire pour *XML DB Repository*

Début de la grammaire	Stockage puis enregistrement

`<xsd:schema`
`  xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
`  xmlns:xdb="http://xmlns.oracle.com/xdb"`
`  xdb:storeVarrayAsTable="true"`
`  **xdb:schemaURL**="http://www.soutou.net/`
`  compRepository.xsd">`
`  <xsd:element name="compagnie"`
`  type="compagnieType"`
`  **xdb:defaultTable**="TABCOMPREPO"/>`
`  <xsd:complexType name="compagnieType">`
…
	`DECLARE`
	`  v_resultat BOOLEAN;`
	`BEGIN`
	`-- stockage`
	`  v_resultat :=`
	`  DBMS_XDB.CREATERESOURCE`
	`  ('/home/SOUTOU/xsd/compagniesRepository.xsd',`
	`  BFILENAME('REPXML',`
	`  'compagniesRepository.xsd'),`
	`  NLS_CHARSET_ID('AL32UTF8'));`
	`-- enregistrement`
	`  DBMS_XMLSCHEMA.**REGISTERSCHEMA**`
	`  (SCHEMAURL`
	`=>'http://www.soutou.net/compRepository.xsd',`
	`  SCHEMADOC => **XDBURITYPE**(`
	`'/home/SOUTOU/xsd/compagniesRepository.xsd').G`
	`ETCLOB(),`
	`  LOCAL => TRUE, GENTYPES =>TRUE,`
	`  GENTABLES=>TRUE, FORCE=>TRUE);`
	`END;`
	`/`

Chaque contenu XML (ressource au vocable *XML DB Repository*) doit renseigner au niveau de l'élément racine l'espace de nom associé, ici `xsi` et l'attribut `noNamespaceSchemaLocation`, dans mon cas :

- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` et
- `xsi:noNamespaceSchemaLocation="http://www.soutou.net/compRepository.xsd"`.

Il est possible enfin d'extraire les emplacements (chemins de répertoires) qui contiennent une ressource dont l'élément racine est précisément défini dans cette grammaire (on suppose les documents `comp1.xml` et `comp2.xml` respectent la grammaire et les conditions précédemment énoncées).

Tableau- Localisation d'emplacements

Requête SQL	Résultat

`SELECT ANY_PATH FROM RESOURCE_VIEW WHERE`
`EXISTSNODE(RES,`
`'/**Resource**[**SchemaElement**="http://www.soutou.net/compRep`
	`ANY_PATH`
	`----------`
	`/home/SOUTOU/comp2.xml`

ository.xsd#compagnie"]')=1;	/home/SOUTOU/general/comp1.xml

Le stockage des documents basés sur une grammaire alimente automatiquement la table par défaut définie dans l'élément racine de la grammaire. Chaque ligne de cette table est accessible par une référence physique (élément `XMLRef` de la vue `RESOURCE_VIEW`). Cet élément doit être utilisé dans une jointure avec la table `XMLType`. Le tableau suivant présente les possibilités de cette technique. La première requête extrait, sous la forme de `CLOB`, les deux premiers contenus associés à cette grammaire (avec leur emplacement). La seconde requête extrait la référence associée au contenu d'une ressource donnée (pas exploitable sans jointure). La dernière extrait une partie d'un contenu donné (ici le nom des pilotes).

Tableau- Extractions par jointure avec la vue `RESOURCE_VIEW`

Requêtes SQL	Résultats

`SELECT ANY_PATH, EXTRACT(VALUE(c), '/compagnie/pilotes').GETCLOBVAL() "CLOB" FROM RESOURCE_VIEW r, TABCOMPREPO c WHERE EXTRACTVALUE(r.res,'/Resource/XMLRef') = REF(c) AND ROWNUM < 3;`	`ANY_PATH`
	`----------`
	`CLOB`
	`----------`
	`/home/SOUTOU/general/comp1.xml`
	`<pilotes>`
	`  <pilote brevet="PL-9">`
	`  <nom>J. Nouveau</nom>`
	`  <salaire>9000</salaire>`
	`  </pilote>`
	`</pilotes>`
	
	`/home/SOUTOU/comp2.xml`
	`<pilotes>`
	`  <pilote brevet="P-10">`
	`  <nom>G. Diffis</nom>`
	`  <salaire>19000</salaire>`
	`  </pilote>`
	`  <pilote brevet="P-11">`
	`  <nom>S. Lacombe</nom>`
	`  <salaire>23000</salaire>`
	`  </pilote>`
	`</pilotes>`

`SELECT EXTRACTVALUE(res,'Resource/XMLRef') FROM RESOURCE_VIEW WHERE ANY_PATH = '/home/SOUTOU/general/comp1.xml';`	`EXTRACTVALUE(RES,'RESOURCE/XMLREF')`
	`----------`
	`00002802093996158 23EE346498E0533F293D703E3417C1AFC651D49F892B9646B6F171956010001B40000`

`SELECT alias.NOM FROM RESOURCE_VIEW rv, TABCOMPREPO c, XMLTable('/compagnie/pilotes/pilote' PASSING c.OBJECT_VALUE COLUMNS nom VARCHAR2(20) PATH 'nom') alias WHERE EQUALS_PATH (rv.RES,'/home/SOUTOU/comp2.xml') = 1 AND REF(c)=EXTRACTVALUE (rv.RES,'/Resource/XMLRef');`	`NOM`
	`----------`
	`G. Diffis`
	`S. Lacombe`

Vue `PATH_VIEW`

Cette vue est composée de cinq colonnes :

- `PATH` (`VARCHAR2`) : indique un chemin (absolu) d'une ressource
- `RES` (`XMLType`) : décrit une ressource du répertoire décrit dans `PATH` ;
- `LINK` (`XMLType`) : décrit un lien vers la ressource ;
- `RESID` (`RAW`) contient l'identifiant d'une ressource.

L'interrogation de cette vue associe les fonctions suivantes :

- DEPTH(*correlation*) retourne la profondeur (niveau du répertoire) relativement à un répertoire donné (de départ). L'entier en paramètre permet de corréler cette fonction à UNDER_PATH. La fonction PATH fournit aussi un tel paramétrage.
- UNDER_PATH(*col_resource*, *profondeurMax*, *chemindepart*, *correlation*) détermine si une ressource est présente sous un chemin passé en paramètre sous la contrainte d'un nombre de niveaux et en corrélation avec les fonctions DEPTH ou PATH.

Tableau- Interrogation de la vue PATH_VIEW

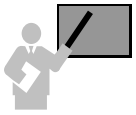
Requête SQL	Résultat

```
SELECT PATH, DEPTH(1),
DEPTH(2)
 FROM PATH_VIEW
  WHERE (UNDER_PATH(RES,3,
  '/sys/schemas/SOUTOU',1) = 1
      OR UNDER_PATH(RES,3,
  '/home/SOUTOU',2) = 1);
```

```
PATH                                     DEPTH(1) DEPTH(2)
---------------------------------------- -------- --------
/home/SOUTOU/general/comp1.xml                           2
/home/SOUTOU/comp2.xml                                   1
/home/SOUTOU/xsd/compagniesRepository.xsd                2
/sys/schemas/SOUTOU/www.soutou.net              1
/sys/schemas/SOUTOU/www.soutou.net/compRe       2
pository.xsd
/home/SOUTOU/general                                     1
/home/SOUTOU/general/note.txt                            2
/home/SOUTOU/xsd                                         1
```

Mises à jour

Les mises à jour de ressources ou de contenus (modifications et suppressions) peuvent également se programmer par UPDATE et DELETE à l'aide de ces vues. Le tableau suivant présente quelques mises à jour (toutes effectives après COMMIT).



La suppression d'une ressource par DELETE (deuxième instruction) n'est possible que si la ressource est finale (document ou répertoire vide). Dans le cas inverse, il faudra supprimer d'abord tous les contenus d'un répertoire pour enfin supprimer le répertoire.

Tableau- Mises à jour de ressources et de contenus

Code SQL	Commentaire

```
UPDATE RESOURCE_VIEW r
    SET r.RES = UPDATEXML(r.RES,
          '/Resource/DisplayName/text()','Mom_memo')
    WHERE EQUALS_PATH(r.RES,
          '/home/SOUTOU/general/note.txt') = 1;
```

Modification du nom logique d'une ressource.

```
DELETE FROM RESOURCE_VIEW
       WHERE EQUALS_PATH(RES,
       '/home/SOUTOU/general/note.txt') = 1;
```

Suppression de la ressource note.txt.

```
UPDATE PATH_VIEW
       SET PATH   = '/home/SOUTOU/comp1.xml'
       WHERE PATH = '/home/SOUTOU/general/comp1.xml';
```

Déplacement de la ressource comp1.xml vers le répertoire /home/SOUTOU.

```
UPDATE RESOURCE_VIEW
  SET res = UPDATEXML(RES,
  '/r:Resource/r:Contents/compagnie/nomComp/text()',
  'Compagnie Euralair',
'xmlns:r="http://xmlns.oracle.com/xdb/XDBResource.xsd"')
  WHERE EQUALS_PATH(RES,'/home/SOUTOU/comp1.xml')=1;
```

Modification du nom de la compagnie du document comp1.xml.

```
UPDATE TABCOMPREPO c
  SET c.OBJECT_VALUE = UPDATEXML(
            c.OBJECT_VALUE,'/compagnie/nomComp/text()',
           'Nouveau nom : Euralair')
  WHERE REF(c) =
    (SELECT EXTRACTVALUE(rv.RES,'/Resource/XMLRef')
     FROM RESOURCE_VIEW rv
     WHERE EQUALS_PATH(rv.RES,'/home/SOUTOU/comp1.xml')=1);
```

Modification du nom de la compagnie du document comp1.xml en utilisant la table de stockage définie par défaut au niveau de la grammaire.

Bibliographie

Sites web :

Oracle Goes XML par By Timothy Dyck :
http://www.eweek.com/c/a/Enterprise-Applications/Oracle-Goes-XML/

Oracle XML DB overview :
http://www2002.org/CDROM/alternate/V5.pdf

FAQ Oracle et XML :
http://www.orafaq.com/faqxml.htm

Travaux pratique Oracle avec XML par Emmanuel Coquery :
TP 7 : XPath : http://www710.univ-lyon1.fr/~ecoquery/enseignement/sibd/tp/xpath.html
TP 8 : XQuery : http://www710.univ-lyon1.fr/~ecoquery/enseignement/sibd/tp/xquery.html
TP 10 : XML, Oracle et Java : http://www710.univ-lyon1.fr/~ecoquery/enseignement/sibd/tp/xml_oracle_java.html

Livres :

Oracle Database 10g XML & SQL: Design, Build, & Manage XML Applications in Java, C, C++, & PL/SQL
Par Mark Scardina, Ben Chang, Jinyu Wang
McGraw-Hill Osborne Media - Première édition 31 mai 2004 - ISBN-13: 978-0072229523

Building Oracle XML Applications
Par Steve Muench
O'Reilly Media, Inc. 2 octobre 2000 - ISBN-13: 978-1565926912

Querying Xml: Xquery, Xpath, And Sql/xml in Context
de Jim Melton et Stephen Buxton
Morgan Kaufmann Publishers - 10 avril 2006 - ISBN-13: 978-1558607118