

Oracle XML DB

- ⦿ Historia
- ⦿ Almacenamiento
- ⦿ Generación de XML a partir de SQL
- ⦿ Consulta de datos XML

Historia

- ◎ Oracle 8 (1997) / 8i (1999):
 - > Carga de librerías Java para el entorno XDK de Oracle JServer
 - > Creación de paquetes PL/SQL para XML sobre procedimientos almacenados en Java
- ◎ Oracle 9i – Release 1 (2002):
 - > XMLType Tipo nativo para XML. Permite almacenar y consultar datos XML.
- ◎ Oracle 9i – Release 2 (2003):
 - > Almacenamiento de datos a partir del XMLType basados en un XML Schema
 - > Repositorio XML DB: para gestión de documentos (enfoque documentcentric)
 - > Nuevas funciones para manejar XMLType, SQL/XML y paquetes basados en C para procesamiento de XML mediante PL/SQL
- ◎ Oracle 10g (2005):
 - > Soporta evolución del XML Schema
 - > Más funcionalidad XML nativa
 - > Oracle XDK permite conectar BD a través de ODBC para la creación de aplicaciones de capa intermedia

Almacenamiento

⦿ Dos opciones:

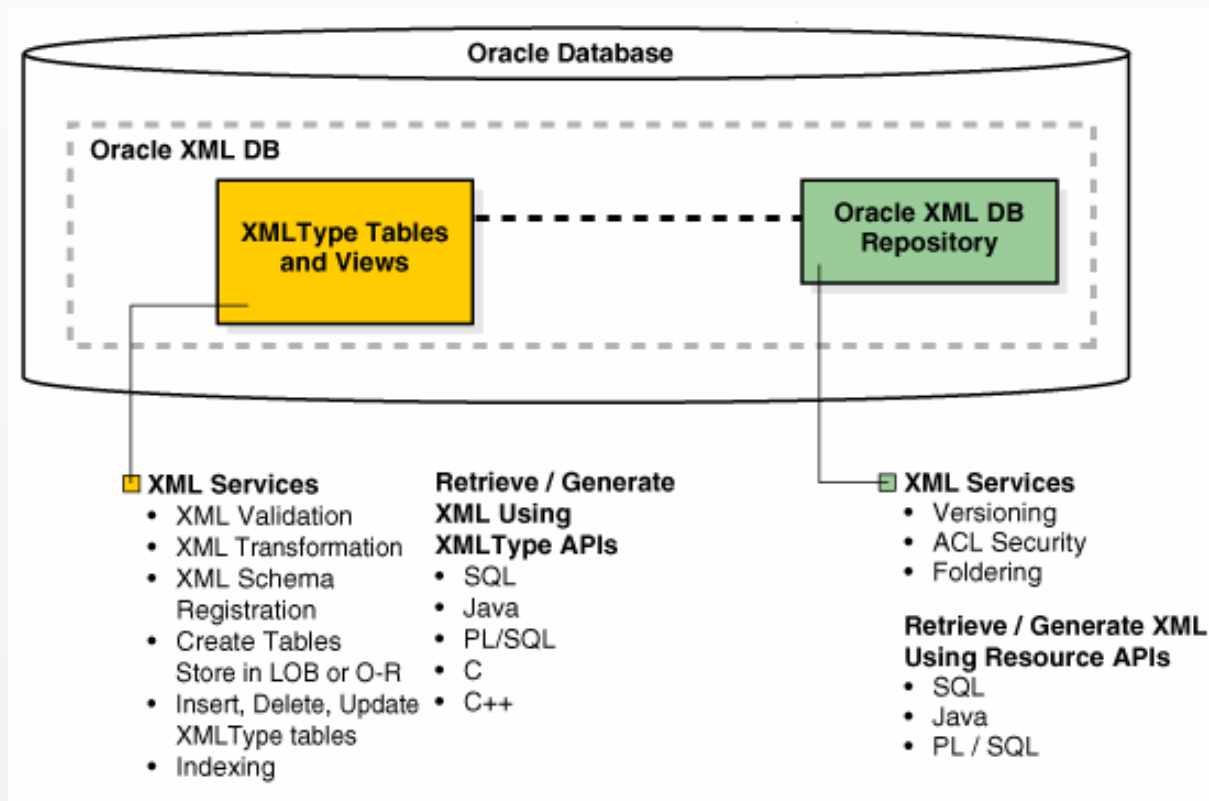
> Repositorio de datos (**Oracle XML DB Repository**):

- Organizado jerárquicamente, consultable
- Almacenamiento y visualización de contenido XML como un directorio jerárquico de carpetas
- Acceso a los documentos y representación de las relaciones entre documentos con:
 - XPath
 - URLs → HTTP/FTP
 - SQL y PL/SQL

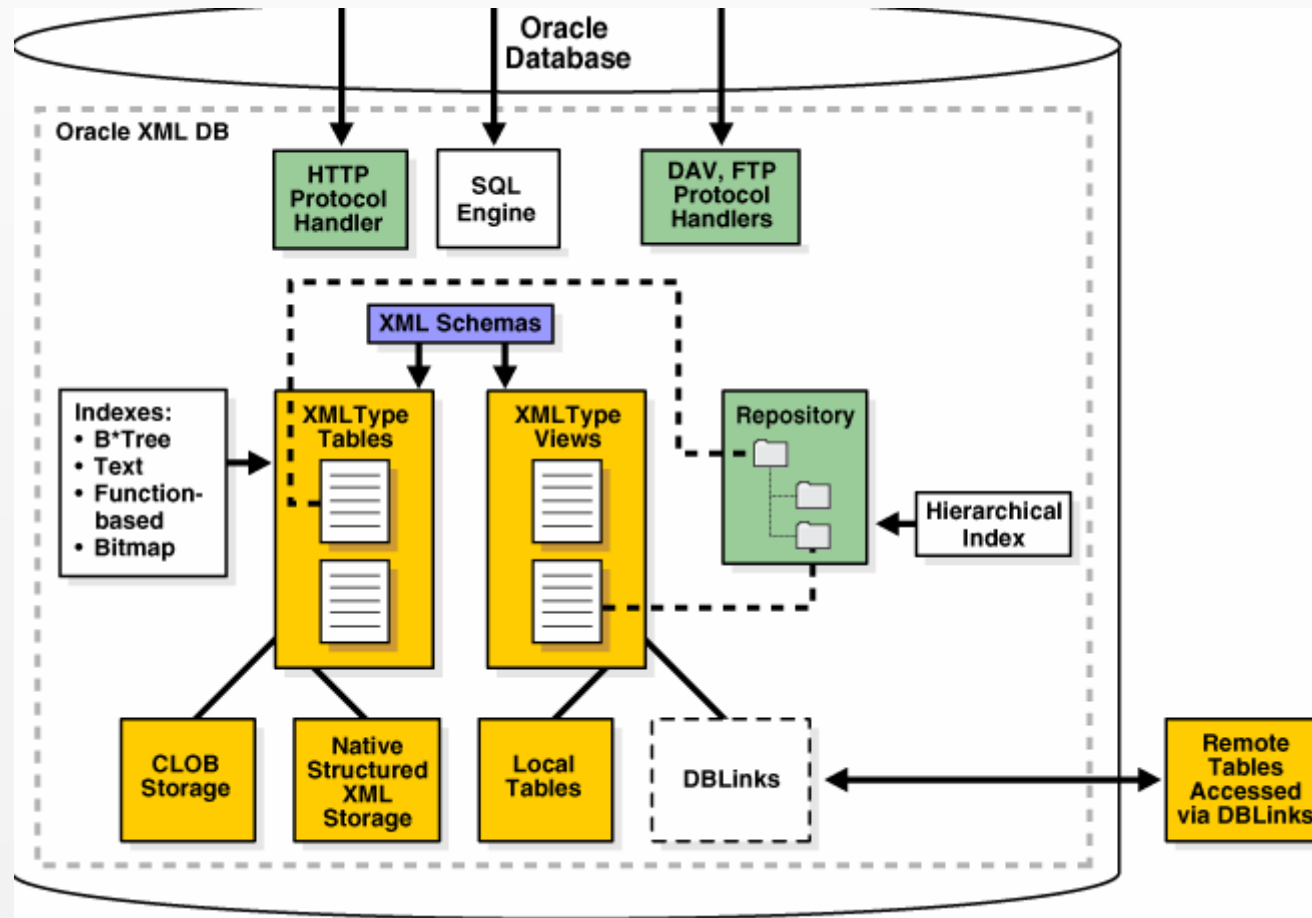
> Tipo de dato nativo (**XMLType**)

- Permite definir tablas, columnas, parámetros, valores devueltos por funciones o variables en procedimientos PL/SQL
- Dispone de Funciones predefinidas para crear instancias XMLType, validar contenidos XML contra XML Schemas, aplicar hojas de estilos XSLT, etc.

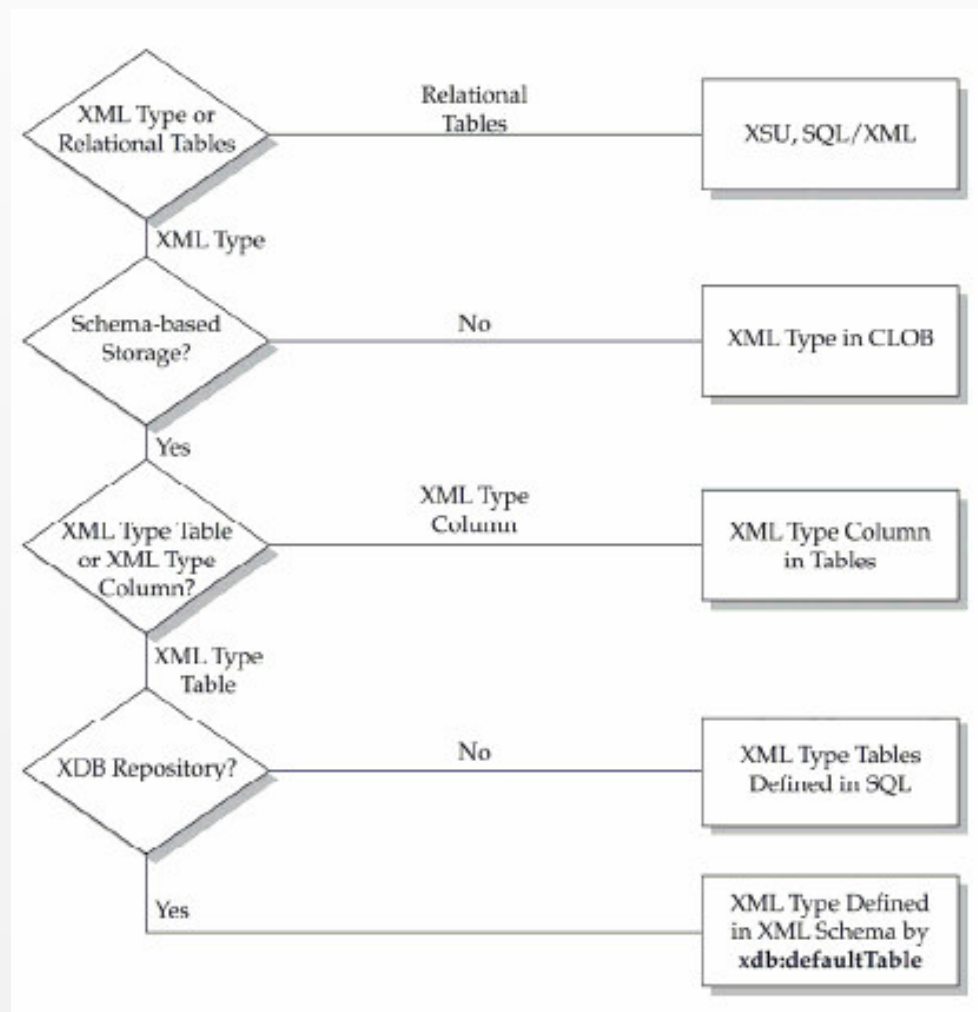
Oracle XML DB Architecture: XMLType Storage and Repository



Oracle XML DB Architecture: XMLType Storage



Oracle XML DB Storage



¿Tablas relacionales o XMLType?

- ⦿ Decisión basada en el formato de los items XML y en la necesidad de conservar fidelidad a nivel DOM para su contenido
- ⦿ **FIDELIDAD DOM:**
 - > En general, cualquier técnica que involucre dividir un documento XML para almacenarlo en una BDR pierde la fidelidad del documento:
 - Espacios en blanco entre los elementos y atributos.
 - El orden de los elementos.
 - Comentarios dentro del documento XML.
 - Procesamiento de instrucciones.
 - Declaraciones de Namespaces
- ⦿ XML DB de Oracle soporta la fidelidad del documento con respecto a su DOM (Document Object Model).

¿Tablas relacionales o XMLType? (y 2)

- ◎ Formato de los items XML
 - > **Data-Centric:** tablas relacionales
 - Estructura regular de los datos
 - Poco contenido o no mezclado
 - Fidelidad DOM no requerida
 - > **Document-Centric:** XMLType
 - Estructura de datos poco regular o irregular
 - Contenido mezclado
 - Muchas consultas sobre el contenido XML

¿XMLType estructurado (esquema) o no estructurado?

- ◎ XMLType no estructurado:
 - > Almacenado en CLOBs (opción por defecto)
 - > Óptimo para documentos XML basados en DTD o en XML Schemas que varían frecuentemente
- ◎ XMLType basado en esquema (estructurado):
 - > Los documentos XML se “dividen” y se almacenan como un conjunto de objetos SQL (tablas, columnas, tipos, etc.)
 - > Mayor rendimiento en consultas y actualizaciones (índices, vistas indexadas,...)
 - > Reduce el espacio de memoria y almacenamiento, aunque tiene mayor overhead la recuperación o actualización de un documento completo
 - > Óptimo para consultas sobre partes del documento XML
 - > Puede validarse el contenido XML con un XML Schema

XMLType: ¿Columna o Tabla?

- En Oracle se puede almacenar datos XML en
 - > Columnas de tipo XMLType

```
CREATE TABLE MiTabla ( id int primary key, ...,  
                        xmlCol xmlType)
```

- > Tablas de objetos a partir del tipo XMLType
 - Crear las tablas utilizando sentencias SQL:

```
CREATE TABLE MiTablaXML OF XMLType
```

- Crear las tablas cuando se registra un Schema XML registrado en el repositorio de Oracle XML DB

```
DBMS_XMLSCHEMA.registerSchema(  
  SCHEMAURL => 'http://xmlns.oracle.com/xdm/MiSchema.xsd',  
  SCHEMADOC => bfilename('XMLDIR','MiSchema.xsd'),  
  CSID => nls_charset_id('AL32UTF8'));
```

Columna XMLType

```
CREATE TABLE Person ( id int primary key, nombre varchar2(20) not null,  
                        direccion xmlType);
```

```
INSERT INTO Person(id,nombre,direccion)
```

```
VALUES(1, 'Peter Smith',
```

```
  XMLTYPE('<MailAddressTo id="1">
```

```
    <Person>Peter Smith</Person> <Street>10 Apple Tree Lane</Street>
```

```
    <City>New York</City> <State>NY</State> <Zipcode>12345</Zipcode>
```

```
  </MailAddressTo>')));
```

```
SELECT extract(direccion, '/MailAddressTo/Street')
```

```
FROM Person;
```

```
UPDATE Person SET direccion = XMLTYPE('<MailAddressTo id="1">
```

```
  <Person>Peter Smith</Person> <Street>10 Downing Street</Street>
```

```
  <City>London</City> <State>England</State>
```

```
  <Zipcode>22334</Zipcode>
```

```
  </MailAddressTo>');
```

Namespaces

- ⦿ Los namespaces se utilizan para describir el conjunto de elementos y atributos que pueden utilizarse en una instancia XML. Una instancia XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML
- ⦿ Hay dos namespaces que se utilizan habitualmente:
 - > <http://www.w3.org/2001/XMLSchema>
 - > <http://xmlns.oracle.com/xdb>
 - Este namespace reescribe algunas funciones SQL, XPath (ej. ora:contains, ceiling, floor, not, string-length, substring, and translate). Los atributos utilizados por XML DB también pertenecen a este namespace.

Trabajando con Schemas

- ① ¿Cómo crear un schema y usarlo sobre documentos XML?
 1. Registrar el schema
 2. Crear una tabla basada en el XMLSchema
 3. Insertar datos en ella

Trabajando con Schemas (y 2)

1. Registrar el schema

```
CREATE DIRECTORY xmldir AS 'c:/temp';
```

```
%Registrar schema
```

```
BEGIN
```

```
  DBMS_XMLSCHEMA.registerSchema(
```

```
    SCHEMAURL => 'http://xmlns.oracle.com/xdb/postbox.xsd',
```

```
    SCHEMADOC => bfilename('XMLDIR','postbox.xsd'),
```

```
    CSID => nls_charset_id('AL32UTF8'));
```

```
END;
```

```
%Borrar schema
```

```
BEGIN
```

```
  DBMS_XMLSCHEMA.deleteSchema(
```

```
    SCHEMAURL => 'http://xmlns.oracle.com/xdb/postbox.xsd',
```

```
    DELETE_OPTION => dbms_xmlschema.DELETE_CASCADE_FORCE);
```

```
END;
```

Trabajando con Schemas (y 3)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0">
<xs:element name="POSTBOX" xdb:defaultTable="POSTBOX">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MailAddressTo">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Person" type="xs:string"/>
            <xs:element name="Street" type="xs:string"/>
            <xs:element name="City" type="xs:string"/>
            <xs:element name="State" type="xs:string"/>
            <xs:element name="Zipcode" type="xs:string"/>
          </xs:sequence>
          <xs:attribute name="id" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

POXBOX.xsd

Trabajando con Schemas (y 4)

2. Como consecuencia de la línea en el Schema

```
<xs:element name="POSTBOX" xdb:defaultTable="POSTBOX">
```

se crea una tabla POSTBOX basada en el XMLSchema

3. Ahora ya podemos insertar datos en la tabla (dos formas)

Insert into POSTBOX values

```
(XMLType(bfilename('XMLDIR', 'datos.txt'),  
          nls_charset_id('AL32UTF8')));
```

Insert into POSTBOX values (xmltype('

```
<POSTBOX><MailAddressTo id="1">  
  <Person>Peter Smith</Person>  
  <Street>10 Downing Street</Street>  
  <City>London</City>  
  <State>England</State>  
  <Zipcode>22334</Zipcode>  
</MailAddressTo>  
</POSTBOX>')');
```

Trabajando con Schemas (y 5)

- Una vez registrado el schema, si se visualiza se observará la existencia de atributos especiales en la definición de los elementos y atributos del XML Schema
- Por ejemplo:
 - > **xdb:defaultTable** – especifica el nombre de la tabla en la que las instancias XML de este esquema serán almacenadas
 - > **xdb:SQLName** – especifica el nombre del objeto SQL que se corresponde con el elemento XML que se está definiendo
 - > **xdb:SQLCollType** – especifica el nombre de la colección SQL que corresponderá al elemento SQL que tiene **maxOccurs > 1**
 - > **xdb:SQLType** – especifica el nombre del tipo SQL correspondiente al elemento XML

Trabajando con Schemas (y 6)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0" xdb:flags="295"
  xdb:schemaURL="http://xmlns.oracle.com/xdb/postbox.xsd"
  xdb:schemaOwner="SYSTEM" xdb:numProps="8">
  <xs:element name="POSTBOX" xdb:defaultTable="POSTBOX" xdb:propNumber="3264"
    xdb:global="true" xdb:SQLName="POSTBOX" xdb:SQLType="POSTBOX228_T"
    xdb:SQLSchema="SYSTEM" xdb:memType="258" xdb:defaultTableSchema="SYSTEM">
    <xs:complexType xdb:SQLType="POSTBOX228_T" xdb:SQLSchema="SYSTEM">
      <xs:sequence>
        <xs:element name="MailAddressTo" xdb:propNumber="3263" xdb:global="false"
          xdb:SQLName="MailAddressTo" xdb:SQLType="MailAddressTo229_T"
          xdb:SQLSchema="SYSTEM" xdb:memType="258" xdb:SQLInline="true"
          xdb:MemInline="false" xdb:JavaInline="false">
          <xs:complexType xdb:SQLType="MailAddressTo229_T" xdb:SQLSchema="SYSTEM">
            <xs:sequence>
              <xs:element name="Person" type="xs:string" xdb:propNumber="3258"
                xdb:global="false" xdb:SQLName="Person" xdb:SQLType="VARCHAR2"
                xdb:memType="1" xdb:SQLInline="true" xdb:MemInline="true" xdb:JavaInline="true"/>
              <xs:element name="Street" type="xs:string" xdb:propNumber="3259"
                xdb:global="false" xdb:SQLName="Street" xdb:SQLType="VARCHAR2"
                xdb:memType="1" xdb:SQLInline="true" xdb:MemInline="true" xdb:JavaInline="true"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Trabajando con Schemas (y 7)

```
create or replace TYPE "MailAddressTo229_T" AS OBJECT
("SYS_XDBPD$" "XDB"."XDB$RAW_LIST_T",
"Id" VARCHAR2(4000 CHAR),
"Person" VARCHAR2(4000 CHAR),
"Street" VARCHAR2(4000 CHAR),
"City" VARCHAR2(4000 CHAR),
"State" VARCHAR2(4000 CHAR),
"Zipcode" VARCHAR2(4000 CHAR))FINAL INSTANTIABLE
```

```
create or replace TYPE "POSTBOX228_T" AS OBJECT
("SYS_XDBPD$" "XDB"."XDB$RAW_LIST_T",
"MailAddressTo" "MailAddressTo229_T")FINAL INSTANTIABLE
```

Columnas basadas en schemas

- Definir una columna como XMLType basado en esquema

```
CREATE TABLE Person2( id int primary key,  
nombre VARCHAR2(100),  
direccion XMLType)
```

```
XMLType COLUMN direccion XMLSCHEMA
```

```
"http://xmlns.oracle.com/xdb/postbox.xsd" ELEMENT "POSTBOX";
```

```
INSERT INTO Person2(id, nombre,direccion)
```

```
VALUES(1,'John Smith',
```

```
XMLTYPE(
```

```
<POSTBOX><MailAddressTo id="1">
```

```
  <Person>Peter Smith</Person> <Street>10 Downing Street</Street>
```

```
  <City>London</City><State>England</State>
```

```
  <Zipcode>22334</Zipcode>
```

```
</MailAddressTo>
```

```
</POSTBOX>').CreateSchemaBasedXML('http://xmlns.oracle.com/xdb/postbox.xsd');
```

Element root del
schema

Tablas basadas en schemas

- Definir una tabla a partir de XMLType basado en esquema

```
CREATE TABLE postbox_xmltype_tbl OF XMLTYPE  
XMLSCHEMA "http://xmlns.oracle.com/xdb/postbox.xsd"  
ELEMENT "POSTBOX";
```



Element root del
schema

SQL/XML: Funciones del estándar suministradas por ORACLE

- ◎ Funciones para generar datos XML con datos procedentes de la BD relacional:
 - > XMLPARSE
 - > XMLSERIALIZE
 - > XMLELEMENT
 - > XMLATTRIBUTES
 - > XMLFOREST
 - > XMLCONCAT
 - > XMLAGG
 - > XMLPI
 - > XMLCOMMENT
 - > XMLROOT

XMLParse y XMLSerialize

- **XMLParse:** Convierte una cadena de caracteres que contiene datos XML en un valor (instancia) de tipo XML
- **XMLSerialize:** Obtiene una representación en string o LOB de un datos de tipo XML

```
INSERT INTO employees VALUES ('123456', 'Smith', ..., XMLPARSE (DOCUMENT '<?xml|
version="1.0"?> <resume xmlns="http://www.res.com/resume"><name> ...
</name><address> ... </address>...</resume>' PRESERVE WHITESPACE) );
```

```
SELECT e.id, XMLSERIALIZE (DOCUMENT e.resume AS VARCHAR (2000)) AS resume
FROM employees AS e
WHERE e.id = '123456';
```

→


ID	RESUME
123456	<?xml version="1.0" encoding ="UTF-8"> <resume xmlns="http://www.res.com/resume"> <name> ... </name> <address> ... </address> ... </resume>

XMLElement

- ① Devuelve un valor XML que es un nodo de elemento Xquery dado,
 - > Un identificador SQL que actúa como su **name**
 - > Una lista opcional de declaraciones namespace
 - > Una lista opcional de nombres y valores de sus atributos
 - > Una lista opcional de expresiones que suministran su contenido
 - > Opciones para contenido NULL
 - Empty element
 - Null
 - Empty element con atributo nil='true'
 - Empty sequence or XQuery document node sin hijos

XMLElement (ejemplo)

```
select e.employee_id,  
       XMLELEMENT(NAME "NombreEmpleado",  
e.first_name) as "resultadoEnXML"  
from hr.employees e
```

 EMPLOYEE_ID	resultadoEnXML
100	<NombreEmpleado>Steven</NombreEmpleado>
101	<NombreEmpleado>Neena</NombreEmpleado>
102	<NombreEmpleado>Lex</NombreEmpleado>
103	<NombreEmpleado>Alexander</NombreEmpleado>

XMLElement (ejemplo anidado)

```
SELECT XMLELEMENT ("Emp",  
    XMLELEMENT ("NombreEmpleado",  
        e.first_name || ' ' || e.last_name ),  
    XMLELEMENT("e-mail", e.email ) )  
FROM hr.employees e;
```

```
<Emp><NombreEmpleado>Steven King</NombreEmpleado><e-mail>SKING</e-mail></Emp>  
<Emp><NombreEmpleado>Neena Kochhar</NombreEmpleado><e-mail>NKOCHHAR</e-mail></Emp>  
<Emp><NombreEmpleado>Lex De Haan</NombreEmpleado><e-mail>LDEHAAN</e-mail></Emp>  
<Emp><NombreEmpleado>Alexander Hunold</NombreEmpleado><e-mail>AHUNOLD</e-mail></Emp>
```

XMLElement (ej. subqueries)

```
SELECT XMLELEMENT (name "Dpto",  
XMLELEMENT ("NombreDpto", d.department_name  
),  
XMLELEMENT("trabajadores",(select count(*) from  
employees e where e.department_id=  
d.department_id)))  
FROM hr.departments d;
```

```
<Dpto><NombreDpto>Administration</NombreDpto><trabajadores>1 </trabajadores></Dpto>  
<Dpto><NombreDpto>Marketing</NombreDpto><trabajadores>2</trabajadores></Dpto>  
<Dpto><NombreDpto>Purchasing</NombreDpto><trabajadores>6</trabajadores></Dpto>  
<Dpto><NombreDpto>Human Resources</NombreDpto><trabajadores>1 </trabajadores></Dpto>
```

XMLATTRIBUTES (dentro XMLELEMENT)

- ⦿ La especificación de atributos debe aparecer directamente después del nombre del elemento y de la declaración del namespace (opcional)
- ⦿ El atributo se puede nombrar implícita o explícitamente

XMLATTRIBUTES (Ej.)

```
SELECT XMLELEMENT ("NombreEmpleado",  
  XMLATTRIBUTES (e.email AS "eMail"),  
  e.first_name || ' ' || e.last_name ) as Resultado  
FROM hr.employees e;
```

```
<NombreEmpleado eMail="SKING">Steven King</NombreEmpleado>  
<NombreEmpleado eMail="NKOCHHAR">Neena Kochhar</NombreEmpleado>  
<NombreEmpleado eMail="LDEHAAN">Lex De Haan</NombreEmpleado>  
<NombreEmpleado eMail="AHUNOLD">Alexander Hunold</NombreEmpleado>
```

XMLCONCAT

- Produce un valor XML dado dos o más expresiones de tipo XML
- Si alguna de las expresiones se evalúa como nulo, es ignorada

SELECT

XMLCONCAT(

 XMLELEMENT("NombreEmpleado", e.first_name),

 XMLELEMENT("Apellido", e.last_name))

FROM hr.employees e;

```
<NombreEmpleado>Ellen</NombreEmpleado><Apellido>Abel</Apellido>
```

```
<NombreEmpleado>Sundar</NombreEmpleado><Apellido>Ande</Apellido>
```

```
<NombreEmpleado>Mozhe</NombreEmpleado><Apellido>Atkinson</Apellido>
```

XMLForest

- Produce una secuencia de elementos XML de los argumentos que se le pasan.
- XMLFOREST permite realizar consultas de forma más compacta, y además tiene como ventaja con respecto a XMLELEMENT que elimina los nulos. Sin embargo, no permite incluir atributos.

XMLForest (ej.)

SELECT

XMLFOREST (e.first_name as "Nombre"
 ,e.email as "Email")

FROM hr.employees e;

```
<Nombre>Steven</Nombre><Email>SKING</Email>  
<Nombre>Neena</Nombre><Email>NKOCHHAR</Email>  
<Nombre>Lex</Nombre><Email>LDEHAAN</Email>
```

XMLAGG

- Función de agregación similar a SUM, AVG, etc.

```
SELECT e.department_id,  
XMLEMENT("DEP", XLAGG (  
    XMLEMENT ("NombreEmpleado", e.first_name )  
    ORDER BY e.first_name))  
FROM hr.employees e  
group by e.department_id;
```

DEP...	XMLEMENT("DEP",XLAGG(XMLEMENT("NOMBREEMPLEADO",E.FIRST_NAME)ORDERBYE.FIRS
10	<DEP><NombreEmpleado>Jennifer</NombreEmpleado></DEP>
20	<DEP><NombreEmpleado>Michael</NombreEmpleado><NombreEmpleado>Pat</NombreEmpleado></DEP>
30	<DEP><NombreEmpleado>Alexander</NombreEmpleado><NombreEmpleado>Den</NombreEmpleado>
40	<DEP><NombreEmpleado>Susan</NombreEmpleado></DEP>

XMLPI

- Permite generar instrucciones de procesamiento

```
SELECT XMLPI(NAME "OrderAnalysisComp",  
  'imported, reconfigured, disassembled')  
AS pi FROM DUAL;
```

XMLComment

- Permite crear un comentario

```
SELECT XMLComment('This is a comment')  
AS cmnt FROM DUAL;
```

XMLRoot

- Función que añade la propiedad versión y opcionalmente la propiedad STANDALONE al item de información root.

```
SELECT  
  XMLRoot(XMLType('<poid>143598</poid>'  
), VERSION '1.0', STANDALONE YES) AS  
xmlroot FROM DUAL;
```

SQL/XML: Funciones propias de ORACLE

- ⦿ Funciones para generar datos XML con datos procedentes de la BD relacional:
 - > SYS_XMLGEN()
 - > XMLSEQUENCE()
 - > XMLCOLATTVAL()
 - > SYS_XMLAGG()

SYS_XMLGEN()

- Similar a la función XMLElement(), pero en este caso, la función recibe un único argumento y devuelve un documento XML bien formado.

```
SELECT SYS_XMLGen(XMLELEMENT ("Empleado",  
  (XMLELEMENT("NombreEmpleado", XMLATTRIBUTES (e.email),  
    e.first_name || ' ' || e.last_name)),  
  XMLELEMENT("Departamento", e.department_id),  
  XMLELEMENT("Telefono", e.phone_number)))  
FROM hr.employees e;
```

```
SYS_XMLGEN(XMLELEMENT("EMPLEADO",(XMLELEMENT("NOMBREEMPLEADO",XMLATTRIBUTES(E.EMAIL),E.FIRST_NAME||' '||E.LAST_NAME)),XMLELEMENT("DEPARTAMENTO",E.DEPARTMENT_ID),XMLELEMENT("TELEFONO",E.PHONE_NUMBER)))  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="SKING">Steven King</NombreEmpleado><Departamento>90</Departamento><Telefono>515.123.4567</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="NKOCHHAR">Neena Kochhar</NombreEmpleado><Departamento>90</Departamento><Telefono>515.123.4568</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="LDEHAAN">Lex De Haan</NombreEmpleado><Departamento>90</Departamento><Telefono>515.123.4569</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="AHUNOLD">Alexander Hunold</NombreEmpleado><Departamento>60</Departamento><Telefono>590.423.4567</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="BERNST">Bruce Ernst</NombreEmpleado><Departamento>60</Departamento><Telefono>590.423.4568</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="DAUSTIN">David Austin</NombreEmpleado><Departamento>60</Departamento><Telefono>590.423.4569</Telefono></Empleado></ROW>  
<?xml version="1.0"?><ROW><Empleado><NombreEmpleado EMAIL="VPATABAL">Valli Pataballa</NombreEmpleado><Departamento>60</Departamento><Telefono>590.423.4560</Telefono></Empleado></ROW>
```

XMLSEQUENCE()

- Realiza la función inversa de SYS_XMLGen. Devuelve un varray de instancias de XMLType. Al devolver una colección, se debe utilizar en el FROM de la consulta

SELECT * FROM

**table(XMLSequence(extract(XMLType('<A>V1V2V3'),
'/A/B')) as tabla;**

COLUMN_VALUE
V1
V2
V3

XMLCOLATTVAL()

- Crea un fragmento XML, con etiqueta COLUMN y un atributo NAME, que lo iguala al nombre de la columna. Podemos cambiar el valor del atributo mediante el alias

```
SELECT XMLCOLATTVAL(e.first_name as nombre)  
FROM hr.employees e;
```

```
XMLCOLATTVAL(E.FIRST_NAMEASNOMBRE)  
<column name = "NOMBRE">Ellen</column>  
<column name = "NOMBRE">Sundar</column>  
<column name = "NOMBRE">Mozhe</column>  
<column name = "NOMBRE">David</column>  
<column name = "NOMBRE">Hermann</column>  
<column name = "NOMBRE">Shelli</column>  
<column name = "NOMBRE">Amit</column>
```

SYS_XMLAGG()

- Agrega todas las instancias XML que se le pasan como parámetro y devuelve un documento XML

```
SELECT SYS_XMLAGG (  
  XMLELEMENT ("NombreEmpleado", e.first_name || '  
    ' || e.last_name))  
FROM hr.employees e;
```

```
<?xml version="1.0"?><ROWSET><NombreEmpleado>Ellen Abel</NombreEmpleado><NombreEmpleado>Sundar Ande</NombreEmpleado><NombreEmpleado>Mozhe Atkinson</NombreEmpleado>
```

SQL/XML: Funciones manipulación de ORACLE

- ◎ Funciones para manipulación de datos XML. Utilizan XPath para localizar items en una instancia XML.
 - > EXTRACT()
 - > EXTRACTVALUE()
 - > EXISTSNODE()
 - > XMLSEQUENCE()
 - > XMLQUERY() (en estándar)
 - > XMLTABLE() (en estándar)
 - > UPDATEXML()
 - > DELETEXML()

EXTRACT()

- Selecciona un nodo individual y sus nodos hoja de una instancia XML.

```
select extract(direccion,  
  '/POSTBOX/MailAddressTo').getStringval()  
from person2;
```

```
<MailAddressTo id="1">  
  <Person>Peter Smith</Person>  
  <Street>10 Downing Street</Street>  
  <City>London</City>  
  <State>England</State>  
  <Zipcode>22334</Zipcode>  
</MailAddressTo>
```

EXTRACTVALUE()

- Extrae el valor de un nodo hoja. Devuelve un valor, no una instancia XML

```
select extractValue(direccion,  
  '/POSTBOX/MailAddressTo/Person') Person  
from person2;
```

PERSON
Peter Smith

EXISTSNODE()

- Busca valores específicos en el nodo hoja, si existe devuelve true.

```
Select count(*) from person2  
where existsNode(direccion,  
    '/POSTBOX/MailAddressTo[Person="Peter  
    Smith"]') = 1;
```

COUNT(*)
1

XMLSEQUENCE()

- A diferencia de `extractValue()`, que sólo extrae un valor de un solo nodo, `XMLSequence` puede buscar en varios nodos o fragmentos de un documento XML.
- Devuelve un varray de instancias de `XMLType`. Al devolver una colección, se debe utilizar en el `FROM` de la consulta

```
select addr.getstringval() from person2 i,  
table(XMLSequence(extract(i.direccion,  
'/POSTBOX/MailAddressTo'))) addr  
where existsNode( i.direccion,  
'/POSTBOX/MailAddressTo[@id="1"]') = 1;
```

XMLQuery()

- ◎ FLWOR (FOR, LET, WHERE, ORDER BY, and RETURN) y XQuery functions doc, count, avg, e integer.
- ◎ Oracle incorpora 5 funciones adicionales: ora:view, ora:matches, ora:replace, ora:sqrt y ora:instanceof

XMLQuery: ej.

```
Select id, nombre, XMLQuery(  
  'for $i in /POSTBOX  
  where $i/MailAddressTo/Zipcode = "22334"  
  return <Details>  
    <Zipcode num="{ $i/MailAddressTo/Zipcode }"/>  
    <CityName char="{ $i/MailAddressTo/City }"/>  
    <City>{if ($i/MailAddressTo/City = "New York")  
      then "Correct City"  
      else "Incorrect City"}  
    </City>  
    <State>{if ($i/MailAddressTo/State = "NY")  
      then "Correct State" else "Incorrect State"}  
    </State>  
  </Details>'  
  )
```

PASSING direccion **RETURNING CONTENT**) Personas
FROM person2;

ID	NOMBRE	PERSONAS
1	John Smith	<Details><Zipcode num="22334"></Zipcode><CityName char="London"></CityName><City>Incorrect City</City><State>Incorrect State</State></Details>

XMLTABLE

- Transforma datos XML en formato tabla
- Evalúa una expresión XQuery o XPath “**el patrón fila**”
 - > Cada item del resultado es una fila
- Los valores de los elementos/atributos se mapean a valores de columna usando expresiones XPath “**el patrón columna**”
- El nombre y tipo de dato tienen que especificarse
- Se puede asignar valores por defecto
- Se puede generar una columna ORDINALITY que contiene un n° secuencial del item XQuery al que corresponde

XMLTABLE (ej.)

Select x.* from person2,
XMLTable('for \$i in /POSTBOX/MailAddressTo
return \$i' passing direccion
columns NumOrden **FOR ORDINALITY**,
Persona varchar2(20) **path** 'Person',
Calle varchar2(20) **path** 'Street',
NumMail varchar2(2) **path** '@id') as x

NUMORDEN	PERSONA	CALLE	NUMMAIL
1	Peter Smith	10 Downing Street	1

UpdateXML()

- Función que permite la actualización parcial de un documento almacenado como un valor XMLType.
- Permite realizar múltiples cambios en una sola operación.
- Cada cambio consiste en una expresión Xpath que identifica el nodo a ser actualizado y el nuevo valor para ese nodo.

UpdateXML(), Ej.

-- actualiza el atributo y el valor de un elemento

```
UPDATE person2
```

```
SET direccion = updateXML(direccion,  
                            '/POSTBOX/MailAddressTo/Person/text()', 'Stephen G. King',  
                            '/POSTBOX/MailAddressTo/@id', '2')
```

```
WHERE existsNode(direccion, '/POSTBOX') = 1;
```

--actualiza el nodo completo

```
UPDATE person2
```

```
SET direccion = updateXML(direccion,  
                            '/POSTBOX/MailAddressTo', XMLType('<MailAddressTo id="3">  
                            <Person>Peter Smith</Person>  
                            <Street>10 Downing Street</Street>  
                            <City>London</City>  
                            <State>England</State>  
                            <Zipcode>22334</Zipcode>  
                            </MailAddressTo>'))
```

```
WHERE existsNode(direccion, '/POSTBOX') = 1;
```

DeleteXML()

- Borra un nodo de cualquier clase

-- elimina el elemento City

```
UPDATE person2
```

```
SET direccion =
```

```
    deleteXML(direccion,
```

```
        '/POSTBOX/MailAddressTo/City')
```

```
WHERE existsNode(direccion,
```

```
    '/POSTBOX/MailAddressTo[@id="3"]') = 1;
```

Otras funciones asociadas al tipo XMLType

- XMLType()
- createSchemaBasedXML()
- createNonSchemaBasedXML()
- getClobVal()
- getNumberVal()
- getStringVal()
- isSchemaBased()
- isSchemaValidate()
- isSchemaValid()
- schemaValidation()
- setSchemaValidate()
- getSchemaURL()
- getRootElement()
- getNamespace()

Validador de schemas XML

<http://www.w3.org/2000/06/webdata/xsv>

XML Schema (REC (20010502) version, as amended) Checking Service - Windows Internet Explorer

http://www.w3.org/2000/06/webdata/xsv?docAddr=&style=text#

Archivo Edición Ver Favoritos Herramientas Ayuda

DAEMON Tools DAEMON Tools Lite AstroBurn Products News [30/30] Weather Radio player IP-lookup Translate Game Database

Google Search Bookmarks Find Check AutoFill Sign In

XML Schema (REC (20010... x C:\Temp\cv3.xml

W3C XML LG

Validator for XML Schema **REC (20010502) version, as amended**

XSV version: XSV 3.1-1 of 2007/12/11 16:20:05

NOTICE: This is a service for a W3C [approved recommendation](#). This version is for schema documents with the namespace URI <http://www.w3.org/2001/XMLSchema> and is being actively developed: see [XSV for XML Schema 20000922 version](http://www.w3.org/2000/10/XMLSchema) for the no longer maintained previous version, for schema documents with the namespace URI <http://www.w3.org/2000/10/XMLSchema>, and [XSV for XML Schema 20000407 version](http://www.w3.org/2000/04/XMLSchema) for the no longer maintained even earlier version, for schema documents with the namespace URI <http://www.w3.org/1999/XMLSchema>.

Use this form for checking a schema which is accessible via the Web, and/or schema-validating an instance with a schema of your own.

Address(es):

Show warnings Keep Going Check as complete schema

Default output is now text/xml with an XSLT stylesheet. Select fallbacks for browsers which don't support <http://www.w3.org/1999/XSL/Transform> stylesheets:

- text/xml + official XSLT (suits IE5 and greater, Mozilla)
- text/xml + early MS XSL support (suits vanilla IE5)
- text/html (styled server-side: suits Netscape, older IE)
- xml, but labelled text/plain (works for any browser, but hard to read)

Listo Internet 100%

Referencias

- ① **Getting into SQL/XML** by *Tim Quinlan*
<http://www.oracle.com/technology/pub/articles/quinlan-xml.html>
- ① **Oracle® XML DB Developer's Guide
10g Release 2 (10.2)**
<http://www.mcs.csueastbay.edu/support/oracle/doc/10.2/appdev.102/b14259/toc.htm>