Manejando Binary XML en Oracle Database 11gR2

Por Francisco Riccio 

Introducción

XML es un lenguaje diseñado para estructurar documentos con la finalidad de intercambiar información entre diferentes plataformas.

Oracle Database desde la versión 9iR2 nos proporciona un tipo de dato nativo llamado XMLTYPE; el cual nos permite albergar información en formato XML con soporte a XSD (esquema de validación), XSLT (transformaciones de documentos), XPATH, XQUERY, indexamiento y particionamiento a documentos XML. En las versiones Oracle Database 9i y 10g los documentos XML se almacenan como CLOB internamente pero a partir de la versión Oracle Database 11g se ha creado una nueva forma de almacenar los documentos XML, dicho almacenamiento es en formato binario (Binary XML). Este nuevo modo de almacenar es más eficiente en el consumo de espacio y en el tiempo de respuesta en el acceso a los datos. Binary XML es el modo de almacenar por default a partir de la versión 11.2.0.2.

Debemos tener presente que albergar información en formato XML en nuestra base de datos da la posibilidad de romper modelos altamente relacionales en modelos relacionales/jerárquicos disminuyendo una serie de tablas normalizadas en nuestro diseño.

En el ejemplo a implementar se creará una tabla llamada RESERVA que almacenará los pedidos de un cliente, dichos pedidos serán almacenados en un campo XML con almacenamiento binario sobre una base de datos versión 11.2.0.3.

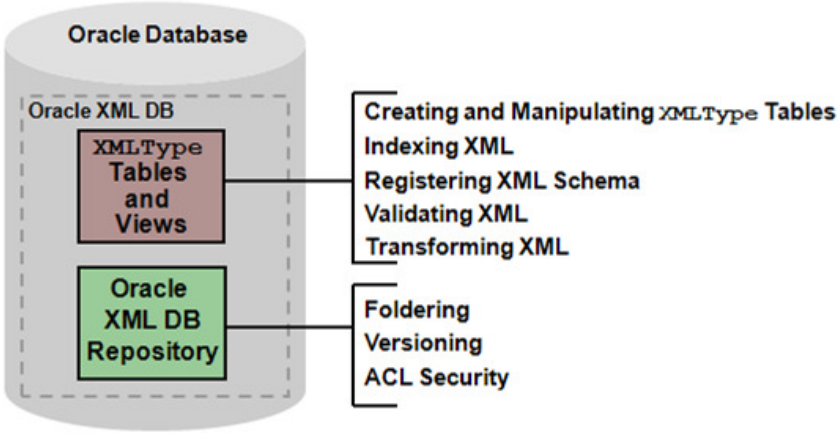
Implementación

a) Validación del componente XDB

El trabajo en XML con Oracle Database se hace a través del componente XDB. Este componente permite albergar información de tipo XMLType (tipo de dato nativo para XML).

Asimismo el componente XDB mantiene un repositorio llamado XML DB Repository que permite organizar y manejar los documentos XML en forma de archivos y carpetas, los cuales son llamados recursos. Debemos pensar en este repositorio como si fuera un filesystem que se encuentra internamente en nuestra base de datos.

Se adjunta un gráfico mostrando los 2 sub-componentes explicados que conforman el componente XDB de la base de datos.



Para validar el status de este componente consultamos el siguiente query:

SQL> select comp_name, status from dba_registry where comp_name='Oracle XML Database';

El status debe devolver el valor de VALID.

Ejemplo:

```
SQL> select comp_name, status from dba_registry where comp_name='Oracle XML Database';

COMP_NAME                                STATUS
---------------------------------------- -----------
Oracle XML Database                      VALID
```

b) Creando el Esquema de Validación

Para dicha implementación, primero crearemos un esquema de validación de documentos XML (XSD) con la finalidad de que todo documento XML ingresado cumpla cierta estructura y condiciones.

Acorde a nuestro ejemplo, necesitaríamos crear un esquema de validación que solo permita el ingreso de pedidos que se compongan de 5 elementos: fecha, precio unitario (pu), cantidad, descripción y tipo de pedido. Donde el elemento Pedido tiene un atributo de tipo entero. Cada elemento tiene su propio tipo de dato y algunas reglas de negocio por ejemplo: el atributo tipo solo puede albergar los valores A, B y C; asimismo el elemento cantidad solo puede albergar valores enteros comprendidos de 0 a 50.

Cualquier documento XML debe cumplir con dicha especificación y será validado al momento de ser insertado en la tabla RESERVA que crearemos más adelante.

Se presenta el documento XSD que registraremos en la base de datos:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="pedidos">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" name="pedido">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="fecha" type="xs:string" />
                            <xs:element name="pu" type="xs:decimal" />
                            <xs:element name="cantidad" default="1">
                                <xs:simpleType>
                                    <xs:restriction base="xs:unsignedByte">
                                        <xs:minInclusive value="0"/>
                                        <xs:maxInclusive value="50"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="descripcion" type="xs:string"/>
                            <xs:element name="tipo">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:enumeration value="A"/>
                                        <xs:enumeration value="B"/>
                                        <xs:enumeration value="C"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="cod" type="xs:integer" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
    </xs:complexType>
    </xs:element>
</xs:schema>
```

Para registrar el documento XSD en la base de datos lo hacemos a través del procedure **DBMS_XMLSCHEMA.REGISTERSCHEMA**:

```
begin

 DBMS_XMLSCHEMA.REGISTERSCHEMA(SCHEMAURL=>'pedidos.xsd', SCHEMADOC=>'<?xml
version="1.0" encoding="utf-8"?>

          <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

           <xs:element name="pedidos">

            <xs:complexType>
```

```
<xs:sequence>
 <xs:element maxOccurs="unbounded" name="pedido">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="fecha" type="xs:string" />
    <xs:element name="pu" type="xs:decimal" />
    <xs:element name="cantidad" default="1">
     <xs:simpleType>
      <xs:restriction base="xs:unsignedByte">
          <xs:minInclusive value="0"/>
        <xs:maxInclusive value="50"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="descripcion" type="xs:string"/>
    <xs:element name="tipo">
     <xs:simpleType>
      <xs:restriction base="xs:string">
            <xs:enumeration value="A"/>
        <xs:enumeration value="B"/>
        <xs:enumeration value="C"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
   </xs:sequence>
```

```
                        <xs:attribute name="cod" type="xs:integer" use="required"/>

                  </xs:complexType>

                 </xs:element>

                </xs:sequence>

               </xs:complexType>

              </xs:element>

            </xs:schema>', LOCAL=>true, GENTYPES=>false, GENBEAN=>false, GENTABLES=>false,
FORCE=>false, OPTIONS=>DBMS_XMLSCHEMA.REGISTER_BINARYXML, OWNER=>USER);

 commit;

end;

/
```

```
SQL> begin
 DBMS_XMLSCHEMA.REGISTERSCHEMA(SCHEMAURL=>'pedidos.xsd', SCHEMADOC=>'<?xml version="1.0" encoding="utf-8"?>
  2    3        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  4        <xs:element name="pedidos">
  5          <xs:complexType>
  6            <xs:sequence>
  7              <xs:element maxOccurs="unbounded" name="pedido">
  8                <xs:complexType>
  9                  <xs:sequence>
 10                    <xs:element name="fecha" type="xs:string" />
 11                           <xs:element name="pu" type="xs:decimal" />
 12                    <xs:element name="cantidad" default="1">
 13                       <xs:simpleType>
 14                                <xs:restriction base="xs:unsignedByte">
 15                             <xs:minInclusive value="0"/>
                                   <xs:maxInclusive value="50"/>
 16   17                                 </xs:restriction>
 18                       </xs:simpleType>
 19                    </xs:element>
 20                    <xs:element name="descripcion" type="xs:string"/>
 21                    <xs:element name="tipo">
 22                       <xs:simpleType>
 23                                <xs:restriction base="xs:string">
 24                               <xs:enumeration value="A"/>
 25                                    <xs:enumeration value="B"/>
 26                                    <xs:enumeration value="C"/>
 27                                </xs:restriction>
 28                       </xs:simpleType>
                       </xs:element>
 29   30                      </xs:sequence>
 31                           <xs:attribute name="cod" type="xs:integer" use="required"/>
 32                 </xs:complexType>
 33               </xs:element>
 34             </xs:sequence>
 35           </xs:complexType>
 36         </xs:element>
 37       </xs:schema>', LOCAL=>true, GENTYPES=>false, GENBEAN=>false, GENTABLES=>false, FORCE=>false, OPTIONS=>DBMS_XMLSCHEMA.REGISTER_BINARYXML, OWNER=>USER);
 38   commit;
 39  end;
 40  /

PL/SQL procedure successfully completed.
```

Se detallan algunos de los parámetros:

a) Local, si el valor es true el documento XSD se guardará dentro del componente XDB en la carpeta /sys/schemas/<owner>/. En caso contrario se guardará en la carpeta /sys/schemas/PUBLIC/.

b) Gentypes, al crear el archivo XSD se creará un tipo de Object Table que se basará en las reglas del esquema cuando se le asigne el valor de true. En nuestro caso posteriormente crearemos la tabla de forma manual, por dicha razón se ha colocado el valor de false.

c) Genbeans, si su valor es true se creará Java Beans al crear el archivo XSD.

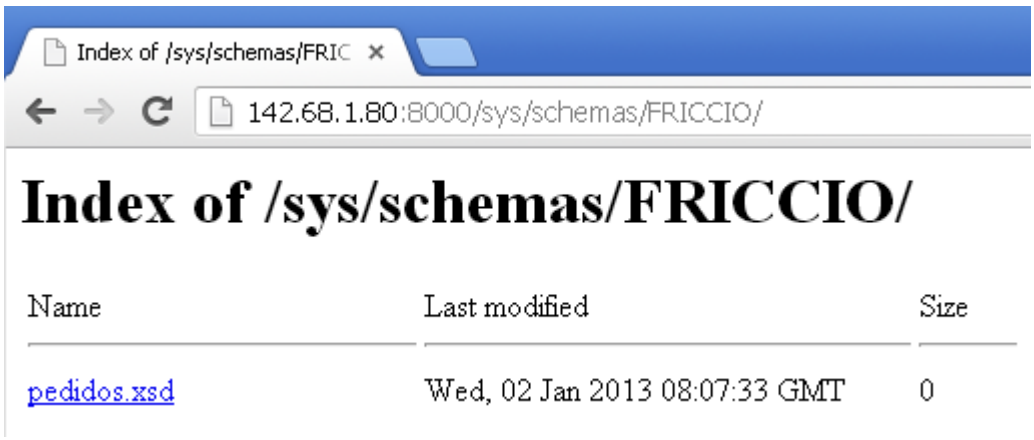
d) Gentables, al crear el archivo XSD se creará una tabla que se basará en las reglas del esquema si se le asigna el valor de true.

e) Force, si se coloca el valor de true no dará error en caso exista problemas en registrar el archivo XSD.

Es importante colocar la opción: DBMS_XMLSCHEMA.REGISTER_BINARYXML si es que pensamos almacenar los documentos en Binary XML. Si no agregamos está opción y queremos crear una tabla que use almacenamiento en Binary XML tendremos el siguiente error:

```
ERROR at line 1:
ORA-44424: BINARY XML storage requires XML Schema registered for BINARY usage
```

Asimismo podemos ver como el archivo XSD (pedidos.xsd) ha sido creado dentro del XML DB Repository después de ser registrado:



Debemos configurar los puertos de XDB para acceso FTP o HTTP si deseamos visualizar o manipular los archivos dentro de XML DB Repository por dichos protocolos.

Esto lo realizamos mediante el paquete dbms_xdb, ejemplo:

```
SQL> connect / as sysdba
Connected.
SQL> execute dbms_xdb.sethttpport(8000);

SQL> execute dbms_xdb.setftpport(9000);

PL/SQL procedure successfully completed.
```

Se adjunta un ejemplo de cómo deben ser los documentos XML que se ingresarán en la tabla RESERVA cumpliendo con el esquema de validación.

```
<?xml version="1.0" ?>
- <pedidos>
  - <pedido cod="1">
      <fecha>01-01-2013</fecha>
      <pu>45.5</pu>
      <cantidad>2</cantidad>
      <descripcion>Botella de Vino</descripcion>
      <tipo>C</tipo>
    </pedido>
  - <pedido cod="2">
      <fecha>31-12-2012</fecha>
      <pu>25</pu>
      <cantidad>1</cantidad>
      <descripcion>Menu Ejecutivo</descripcion>
      <tipo>A</tipo>
    </pedido>
  </pedidos>
```

Nota: Debemos recordar que otra manera de poder validar documentos XML es mediante DTD, pero a diferencia de los esquemas de validación, los DTD no siguen una sintaxis XML (nacieron basado en el ISO 8879 para el lenguaje SGML y no para XML) y además no permite especificar los tipos de datos de los elementos, por dicha razón la recomendación es usar esquemas de validación.

c) Creando la tabla Reserva

SQL> CREATE TABLE FRICCIO.RESERVA(id number, pedido xmltype)

XMLTYPE COLUMN pedido

STORE AS BINARY XML

XMLSCHEMA "http://xmlns.oracle.com/xdb/schemas/FRICCIO/pedidos.xsd"

ELEMENT "pedidos";

```
SQL> CREATE TABLE FRICCIO.RESERVA(id number, pedido xmltype)
XMLTYPE COLUMN pedido
STORE AS BINARY XML
XMLSCHEMA "http://xmlns.oracle.com/xdb/schemas/FRICCIO/pedidos.xsd"
ELEMENT "pedidos";

Table created.
```

Hemos creado una tabla llamada RESERVA que se compone de un campo llamado pedido de tipo XMLTYPE el cual le estamos especificando que será almacenado como Binary XML. Recordemos que a partir de la versión 11.2.0.2 hacia delante, éste será el default. También indicamos que el campo pedido será validado por el esquema pedidos.xsd que previamente lo hemos creado.

Nota: El tipo de dato XMLTYPE almacenado como Binary XML se guarda internamente como un SecureFile LOB automáticamente en versión Oracle Database 11.2.0.2. En caso no pueda crearse de esa manera lo hará como Basic LOB. Recordemos que no será posible la creación de objetos SecureFile LOB cuando el tablespace no está configurado como ASSM ó el parámetro DB_SECUREFILE está en FALSE.

Ingresaremos un documento XML que no cumpla la especificación del esquema XSD definido para ese campo:

```
SQL> insert into friccio.reserva values (1,
'<?xml version="1.0"?>
<pedidos>
  2    3    4   <pedido>
  5             <fecha>01-01-2013</fecha>
  6             <pu>45.5</pu>
  7             <cantidad>2</cantidad>
  8             <descripcion>Botella de Vino</descripcion>
  9             <tipo>C</tipo>
 10     </pedido>
 11     <pedido>
 12             <fecha>31-12-2012</fecha>
 13             <pu>25</pu>
 14             <cantidad>1</cantidad>
                <descripcion>Menu Ejecutivo</descripcion>
 15   16               <tipo>A</tipo>
 17     </pedido>
 18  </pedidos>');
insert into friccio.reserva values (1,
                    *
ERROR at line 1:
ORA-31061: XDB error: XML event error
ORA-19202: Error occurred in XML processing
LSX-00266: missing required attribute "cod"
```

En este ejemplo vemos que la operación de INSERT falla porque el documento XML no cumple con la especificación definida en el archivo XSD. En este caso faltó el atributo "cod" de cada elemento Pedido, donde el atributo lo hemos configurado como requerido.

Lo corregimos y veremos que ahora si se registra:

```
SQL> insert into friccio.reserva values (1,
'<?xml version="1.0"?>
          <pedidos>
        <pedido cod="1">
                <fecha>01-01-2013</fecha>
                <pu>45.5</pu>
                <cantidad>2</cantidad>
                <descripcion>Botella de Vino</descripcion>
                <tipo>C</tipo>
        </pedido>
        <pedido cod="2">
                <fecha>31-12-2012</fecha>
                <pu>25</pu>
                <cantidad>1</cantidad>
                <descripcion>Menu Ejecutivo</descripcion>
                <tipo>A</tipo>
        </pedido>
          </pedidos>');

1 row created.
```

Nota: Es posible insertar un documento a partir de un archivo XML existente en el Sistema Operativo o en el XML DB Repository, ejemplo:

```
SQL> INSERT INTO <tabla>

VALUES (XMLType(bfilename('<DIR>','<archivo.xml>'),nls_charset_id('AL32UTF8')));
```

d) Creación de Índices

Para indexar columnas cuyo almacenamiento es Binary XMLType lo podemos hacer mediante: índices basado en funciones ó índices de tipo XML Index.

En **My Oracle Support (MOS) Nota: 742192.1 (Indexing Binary XML Columns)**, se específica que no está asegurado el uso de índices basado en funciones en campos almacenados como Binary XML, por lo cual la recomendación es crearlo como XML Index.

XML Index es un nuevo tipo de índice a partir de la versión Oracle Database 11g.

Ejemplo de su creación:

```
SQL> create index IDX_RESERVA on RESERVA(pedido) INDEXTYPE is XDB.XMLINDEX;
```

El problema con este tipo de creación por default es que creará índices por cada elemento que tenga el documento XML de modo que podría perjudicarnos en espacio.

Para nuestro ejemplo solo indexaré el elemento pu (precio unitario).

SQL> create index idx_reserva ON friccio.reserva(pedido) INDEXTYPE IS XDB.XMLINDEX PARAMETERS

('PATHS (INCLUDE (/pedidos/pedido/pu))');

```
SQL> create index idx_reserva ON friccio.reserva(pedido)
INDEXTYPE IS XDB.XMLINDEX
PARAMETERS ('PATHS (INCLUDE (/pedidos/pedido/pu))');

Index created.
```

Podemos apreciar que para crear el índice de manera más específica debemos apoyarnos del uso de XPATH para seleccionar el elemento que queremos indexar.

Validando:

Si ejecutamos una consulta mediante XPATH podemos apreciar que el índice es utilizado:

```
SQL> select extract(pedido,'/pedidos/pedido/pu/text()') from reserva r where id=1;

EXTRACT(PEDIDO,'/PEDIDOS/PEDIDO/PU/TEXT()')
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
45.525


Execution Plan
----------------------------------------------------------
Plan hash value: 3188923056

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                               |     1 |    15 |     3   (0)| 00:00:01 |
|   1 |  SORT GROUP BY                |                               |     1 |    31 |            |          |
|*  2 |   TABLE ACCESS BY INDEX ROWID| SYS77430_IDX_RESERV_PATH_TABLE |     1 |    31 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN           | SYS77430_IDX_RESERV_PIKEY_IX  |     2 |       |     1   (0)| 00:00:01 |
|*  4 |  TABLE ACCESS FULL            | RESERVA                       |     1 |    15 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------
```

e) Actualizaciones sobre el campo Binary XML

Existen algunas funciones que nos ayudan a dar mantenimiento a los elementos de un documento XML ya registrado. Haremos la demostración de tres de ellos.

e.1) Agregaremos un nuevo elemento pedido sobre el documento XML.

```
SQL> UPDATE reserva

set pedido=appendchildxml(pedido,'/pedidos',

'       <pedido cod="3">

                <fecha>31-12-2012</fecha>

                <pu>30</pu>

                <cantidad>1</cantidad>

                <descripcion>xxx</descripcion>

                <tipo>B</tipo>

        </pedido>')

where id=1;
```

```
SQL> UPDATE reserva
set pedido=APPENDCHILDXML(pedido,'/pedidos',
'       <pedido cod="3">
                <fecha>31-12-2012</fecha>
                <pu>30</pu>
                <cantidad>1</cantidad>
                <descripcion>xxx</descripcion>
                <tipo>B</tipo>
        </pedido>')
where id=1;

1 row updated.
```

e.2) Deseamos modificar el pu (precio unitario) del nuevo elemento pedido ingresado del valor de 30 a 20.

```
SQL> update reserva set pedido=updatexml(pedido,'/pedidos[1]/pedido[3]/pu/text()',20) where id=1;
```

```
SQL> update reserva
set pedido=updatexml(pedido,'/pedidos[1]/pedido[3]/pu/text()',20)
where id=1;

1 row updated.
```

e.3) Deseamos eliminar el último elemento ingresado.

SQL> UPDATE reserva set pedido=**deletexml**(pedido,'/pedidos[1]/pedido[3]');

```
SQL> UPDATE reserva
set pedido=deletexml(pedido,'/pedidos[1]/pedido[3]');

1 row updated.
```

f) Funciones Útiles

f.1) Si deseamos obtener el documento XML como String.

select id,r.PEDIDO.**getStringVal()** from reserva r;

f.2) Si deseamos obtener el documento XML como CLOB.

select id,r.PEDIDO. **getClobVal()** from reserva r;

f.3) Crear un String o CLOB a partir de un contenido.

select **xmlserialize(DOCUMENT|CONTENT** r.PEDIDO **as CLOB|VARCHAR|VARCHAR2)** from reserva r;

g) XPath & XQuery

f.1) XPath

XPath es un lenguaje que nos permite construir expresiones con la finalidad de recorrer un documento XML y entregarnos los nodos del documento que contienen la información que deseamos.

Ejemplos:

Ejemplo 1, deseamos obtener todos los pu (precios unitarios) de la reserva con id=1.

SQL> select extract(pedido,'/pedidos/pedido/pu') from reserva r where id=1;

```
SQL> select extract(pedido,'/pedidos/pedido/pu')
     from reserva r where id=1;

EXTRACT(PEDIDO,'/PEDIDOS/PEDIDO/PU')
--------------------------------------------------
<pu>45.5</pu>
<pu>25</pu>
```

Ejemplo 2, deseamos obtener aquellos pedidos que han superado un precio unitario de 48 de tipo A.

SQL> select id,r.pedido from reserva r where xmlexists('/pedidos/pedido[pu>48 and tipo="A"]/descripcion' passing pedido);

```
SQL> select id,r.pedido
from reserva r
where xmlexists('/pedidos/pedido[pu>30 and tipo="A"]/descripcion' passing pedido);

no rows selected
```

En nuestro caso no existe filas devueltas porque no tenemos ningún pedido que tenga un precio unitario superior a 48 de tipo A.

Ejemplo 3, deseamos obtener aquellos pedidos cuyo atributo cod sea diferente del valor de 3.

SQL> select id,r.pedido.extract('/pedidos/pedido[@cod!=3]/descripcion') from reserva r;

```
SQL> select id,r.pedido.extract('/pedidos/pedido[@cod!=3]/descripcion')
     from reserva r;

        ID
----------
R.PEDIDO.EXTRACT('/PEDIDOS/PEDIDO[@COD!=3]/DESCRIPCION')
--------------------------------------------------------------------------------

<descripcion>Botella de Vino</descripcion>
<descripcion>Menu Ejecutivo</descripc
```

f.2) XQuery

XQuery es un lenguaje de consulta diseñado para trabajar con colecciones de datos XML, el cual proporciona los medios para extraer y manipular información de documentos XML ó de cualquier fuente de datos que pueda ser representada mediante XML.

Ejemplos:

Ejemplo 1, se desea obtener aquellos pu (precios unitarios), si uno de ellos sobrepasa el valor de 40 debe aumentarse un costo de 18%.

```
SQL> select id,xmlquery('for $i in /pedidos/pedido
                         let $pu:=$i/pu/text()
                         where $pu>0
                        order by $pu
                         return <pu valor="{$pu}">
                          {
                          if ($pu >= 40) then
                            $pu*1.18
                          else
                            $pu
                          }
                        </pu>'
                        PASSING pedido RETURNING CONTENT) "ejemplo_xquery"
```

from reserva r;

```
SQL> select id,xmlquery('for $i in /pedidos/pedido
                  let $pu:=$i/pu/text()
  2    3                 where $pu>0
  4                  order by $pu
  5                  return <pu valor="{$pu}">
  6                    {
  7                    if ($pu >= 40) then
  8                      $pu*1.18
  9                    else
 10                      $pu
                      }
 11   12                 </pu>'
 13                  PASSING pedido RETURNING CONTENT) "ejemplo_xquery"
 14  from reserva r;

        ID
----------
ejemplo_xquery
--------------------------------------------------------------------------------
         1
<pu valor="25">25</pu><pu valor="45.5">53.69</pu>
```

Explicación:

a) La función XMLQUERY recibe de parámetro una consulta XQuery y el documento XML el cual lo pasamos a través del comando PASSING y asimismo solicitamos el retorno del resultado del script XQuery.

b) El comando FOR nos permite recorrer todos los nodos a partir de una ruta, en este caso estamos recorriendo todos los elementos Pedido de la raíz (pedidos).

c) El comando LET nos permite crear variables y asignarles un valor específico, en este caso se ha creado la variable $pu.

Ejemplo 2, a partir de una tabla o vista generar una salida XML.

SQL> select xmlquery('ora:view("HR","DEPARTMENTS")' returning content) from dual;

```
SQL> select xmlquery('ora:view("HR","DEPARTMENTS")' returning content) from dual;

XMLQUERY('ORA:VIEW("HR","DEPARTMENTS")'RETURNINGCONTENT)
--------------------------------------------------------------------------------
<ROW><DEPARTMENT_ID>10</DEPARTMENT_ID><DEPARTMENT_NAME>Administration</DEPARTMEN
```

También le podemos colocar comandos de for, let, etc como se vio en el ejemplo 1.

SQL> select xmlquery('for $i in ora:view("HR","DEPARTMENTS") return $i' returning content) FROM DUAL;

```
SQL> select xmlquery('for $i in ora:view("HR","DEPARTMENTS") return $i' returning content) FROM DUAL;

XMLQUERY('FOR$IINORA:VIEW("HR","DEPARTMENTS")RETURN$I'RETURNINGCONTENT)
--------------------------------------------------------------------------------
<ROW><DEPARTMENT_ID>10</DEPARTMENT_ID><DEPARTMENT_NAME>Administration</DEPARTMEN
```

La opción "ora:view" crea una vista en tiempo de runtime, esta opción está disponible desde la versión Oracle Database 11g.

g) Vistas del Diccionario de Datos

Oracle Database nos ofrece algunas vistas útiles referente a nuestros objetos XML.

Listar todos los schemas de la base de datos:

DBA_XML_SCHEMAS

Ejemplo:

```
SQL> select SCHEMA_URL,LOCAL,BINARY from DBA_XML_SCHEMAS where owner='FRICCIO';

SCHEMA_URL                     LOC BIN
------------------------------ --- ---
pedidos.xsd                    YES YES
```

Listar todos los Object Tables XML:

DBA_XML_TABLES

```
SQL> select TABLE_NAME from DBA_XML_TABLES where owner='FRICCIO';

no rows selected
```

En nuestro caso nos devuelve filas ya que hemos creado tablas que contienen columnas XML no Object Tables XML.

Listar todas las columnas que son de tipo XMLTYPE o XML INDEX:

DBA_XML_TAB_COLS

```
SQL> select TABLE_NAME,COLUMN_NAME,STORAGE_TYPE
     from dba_xml_tab_cols where owner='FRICCIO';

TABLE_NAME                     COLUMN_NAME                    STORAGE_TYPE
------------------------------ ------------------------------ ------------
RESERVA                        PEDIDO                         BINARY
```

i) XML DOM

XML DOM es una interfaz de programación (API) que proporciona un conjunto de objetos para representar documentos XML y asimismo acceder y modificar el contenido, estructura y estilo de un documento XML. La implementación de DOM sobre Oracle Database está dado sobre el paquete XMLDOM.

Sobre nuestro escenario se recorrerá cada atributo y elemento de cada documento XML.

Ejemplo:

```
set serveroutv_parseut on
declare
 v_parse          xmlparser.Parser;
 v_doc            xmldom.DOMdocument;
 v_nodo           xmldom.DOMNode;
 childnode        dbms_xmldom.DOMNode;
 v_atributo       xmldom.DOMNode;
 v_nodos          xmldom.DOMNodeList;
 v_nodos_hijo     xmldom.DOMNodeList;
 v_elemento       xmldom.DOMNode;
 v_atributos      xmldom.DOMNamedNodeMap;
 v_xml            clob;
begin
  /*Obtenemos el documento xml de la base de datos*/
  select r.pedido.getClobVal()
  into v_xml
  from reserva r
  where id=1;
  /*Se parsea el contenido del documento xml*/
  v_parse:= xmlparser.newparser();
  xmlparser.parseclob(v_parse,v_xml);
  /*Obtenemos el documento xml cargado en un objeto DOM*/
  v_doc:=xmlparser.getdocument(v_parse);
  xmlparser.freeparser(v_parse);
  /*Obtenemos un arreglo de nodos que tiene la raiz del documento*/
  v_nodos:=xmldom.getChildrenByTagName(xmldom.getdocumentelement(v_doc),'*');
  dbms_output.put_line('Total de nodos: '||xmldom.getlength(v_nodos));
  /*Recorremos cada nodo del arreglo, el arrego inicia en el item 0*/
  for j in 1..xmldom.getlength(v_nodos) loop
   v_nodo:=xmldom.item(v_nodos,j-1);
   /*Obtenemos la lista de elementos que tiene el nodo actual*/
   v_nodos_hijo:=xmldom.getchildnodes(v_nodo);
   --Recorremos los atributos del nodo--
   /*Obtenemos la lista de atributos que tiene el nodo*/
   v_atributos:=xmldom.getattributes(v_nodo);
   /*Recorrdemos cada atributo del nodo*/
   for i in 1..xmldom.getlength(v_atributos) loop
    /*Obtenemos un atributo*/
    v_atributo:=xmldom.item(v_atributos,i-1);
    dbms_output.put_line(xmldom.getnodename(v_atributo)||'='||xmldom.getnodevalue(v_atributo));
   end loop;
   --Recorremos los elementos del nodo--
   /*Recorremos cada elemento del nodo*/
   for z in 1..xmldom.getlength(v_nodos_hijo) loop
     /*Obtenemos un elemento*/
    v_elemento:=xmldom.item(v_nodos_hijo,z-1);
    dbms_output.put_line(xmldom.getnodename(v_elemento)||'='||xmldom.getNodeValue(xmldom.getfirstchild(v_elemento)));
   end loop;
  end loop;
end;
/
```

Conclusión

Concluimos que Oracle Database nos provee todos un soporte completo y herramientas para poder trabajar con documentos XML y sus tecnologías relacionadas como XPATH, XQuery, Esquemas de validación, etc. Podemos asimismo aprovechar XML en ventaja de generar modelos relacionales/jerárquicos en pro de eliminar un nivel detallado de normalizaciones en nuestro diseño y así eliminar una serie de joins en nuestras consultas SQL generando un mejor tiempo de respuesta en nuestros programas.

Publicado por Ing. Francisco Riccio. Es un IT Specialist en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: francisco@friccio.com

web: www.friccio.com