Dynamische Generierung von XML in der Datenbank

Kunde:	DOAG*News*
Ort, Datum:	Artikel im Heft Q3 / 2005
Thema / Themen:	**Artikel von merlin.zwo**
Projekt:	**Dynamische Generierung von XML In der Datenbank**
Autor:	**Hakon Lugert**

- Oracle & Technologien
- Systementwicklung
- Individuelle Lösungen

In den letzten Versionen wurde die XML-Unterstützung von Oracle stetig vorangetrieben – bis zur aktuellen XMLDB, mit der XML-Dokumente sehr einfach in die Datenbank geladen, dort nach Bedarf verarbeitet und auch wieder herausgeneriert werden können. Daneben gibt es schon seit einigen Versionen die SQL-Erweiterung SQLX (mit den Funktionen xmlelement, xmlforest, …), um XML direkt aus Select-Statements zu erhalten. Eine Voraussetzung muss bei diesen Methoden zur XML-Erzeugung jedoch gegeben sein: Das XML-Schema (XSL), d.h. die Beschreibung der XML-Struktur, muss bekannt sein. Bei der XMLDB wird ein solches Schema als Basis für den XMLType registriert, ein SQLX-Statement ist zwingend so aufgebaut, dass es immer die gleiche hierarchische XML-Struktur zurückliefert.

Doch welche Lösungsansätze gibt es, wenn die zu verarbeitende XML-Struktur nicht bekannt ist? Sofern immer neue XML-Schematas in der Datenbank verarbeitet werden müssen, ohne dass die Datenbankanwendung diese Strukturen kennt bzw. kennen muss, liegt der Ansatz einer dynamischen Verarbeitung eines XML-Dokumentes nahe. Dieser Fall kommt in der Praxis durchaus vor, denn beim Datenaustausch mit externen Tools gibt es solche, bei denen ein Export als XML möglich ist, die aber nicht ein zugehöriges XSL-Sheet liefern. Umfang und Struktur eines solchen XML-Exports hängen mit dem Definitionsumfang in der externen Anwendung zusammen, sind also von Fall zu Fall unterschiedlich. Sollen XML-Files mit unbekannter Struktur in der Datenbank verarbeitet, modifiziert und auch wieder exportiert werden, so steht zur dynamischen Verarbeitung eine umfangreiche API für XML in der Datenbank zur Verfügung. Mit dieser API können XML-Typen flexibel innerhalb PL/SQL behandelt werden.

In einem konkreten Beispiel wird ein XML-File aus der externen Anwendung in die Datenbank geladen und in einer hierarchischen Tabelle gespeichert. Innerhalb der Datenbank-Anwendung kann der Inhalt des XML-Files bearbeitet werden. Schließlich ist auch die Rückgabe der modifizierten Inhalte an das Quellprogramm als XML möglich. Dazu muss das XML in der Datenbank dynamisch generiert werden.

Neben den relevanten Informationen wie Tagname, Attribute und Inhalt eines Tags ist die hierarchische Struktur über die Zuordnung KNOTEN – VATER_KNOTEN in

der Tabelle abgebildet. Jedes importierte XML-File bekommt eine eigene File-ID, so dass die Zuordnung der Tags zu dem XML-File möglich ist.
Um aus den Tabellendaten ein XML-File zu erzeugen, wird zunächst ein Select benötigt, der die zur XML-Erzeugung relevanten Informationen beinhaltet. Dies sind der Tagname, zugehörige Attribute, Inhalt eines Tags und vor allem die hierarchische Zuordnung des Elementes innerhalb der Baum-Struktur.
Dazu wird typischerweise ein Select mit einer CONNECT BY / START WITH Klausel verwendet.

```
((Beginn Programmcode))
SELECT LEVEL, xs.knoten_id, xs.vater_knoten_id,
       xs.xml_tag, xs.xml_attributes,
       xs.xml_tag_value
FROM   xmlstruktur xs
WHERE  xs.file_id = 1
START WITH xs.vater_knoten_id IS NULL --ROOT-Tag
CONNECT BY xs.vater_knoten_id = PRIOR xs.knoten_id;
((Ende Programmcode))
```

Die Ausgabe hat die Eigenschaft, dass die hierarchische Zuordnung aus der Level-Angabe abgeleitet werden kann. Den Anfang macht immer der Datensatz ohne Vaterknoten, denn dieser ist das oberste Element in der XML-Struktur (Root-Tag). Die korrekte Reihenfolge der Elemente ist in diesem Beispiel über die Knoten-ID sichergestellt, kann im Bedarfsfall aber auch über ein anderes Kennzeichen gesteuert werden.

LEVEL	KNOTEN_ID	VATER_KNOTEN_ID	XML_TAG	XML_ATTRIBUTES	XML_TAG_VALUE
1	1		DOAG		
2	2	1	DOAGNEWS	date=“200501“	
3	3	2	BEITRAG	num=“1“	Überschrift 1
3	4	2	BEITRAG	num=“2“	Überschrift 2
3	5	2	BEITRAG	num=“3“	Überschrift 3
2	6	1	DOAGNEWS	date=“200502“	
3	7	6	BEITRAG	num=“1“	Überschrift 1
3	8	6	BEITRAG	num=“2“	Überschrift 2
3	9	6	INSERAT		
4	10	9	TEST	num=“1“	Test 1
4	11	9	TEST	num=“2“ testattrib=“test2“	Test 2

Aus dieser Ausgabe kann jetzt ohne großen Aufwand ein XML-File erstellt werden. Dafür wird die API XMLDOM (dbms_xmldom) verwendet. Bei DOM (Document Object Model) handelt es sich allgemein um eine (W3C-spezifizierte) API, mit der XML-Strukturen und Inhalte beschrieben werden können. Bildlich gesehen ist es ein Container, in dem ein XML-File in einer Baumstruktur hierarchisch abgelegt ist. Dabei sind die einzelnen hierarchischen Knoten (Nodes) als Objekte zu sehen, die immer an einen Vaterknoten gebunden sind, bis zum obersten Knoten. Innerhalb der Oracle XMLDOM-API gibt es unter anderem die Typen DOMDocument,

DOMElement und DOMNode. Im DOMDocument können verschiedenste Elemente und Ihre Verbindung untereinander beschrieben werden. In einer Variable dieses Typs wird in PL/SQL die XML-Struktur und damit auch das XML-File erstellt. Dabei werden die XML-Tags als Elemente (DOMElement) über Knoten (DOMNode) beschrieben. Daneben gibt es noch eine Reihe weiterer Typen, wie z.B. DOMComment, die aber in dem hier vorgestellten Beispiel nicht verwendet werden. Der Umfang der API ist in der Oracle Dokumentation „XMLDB-Developers Guide“ ausführlich beschrieben.

Die wesentlichsten Funktionen zur Erstellung einer XML-Struktur im DOM sind:

- createElement zur Erzeugung eines Elementes
- makeNode zur Erzeugung eines Nodes
- appendChild zur Zuordnung eines Nodes zu einem Vaterknoten
- setAttribute zum Setzen von Attributen.
- getParentNode um innerhalb der Hierarchie zum Vater eines Nodes zu gelangen

Im Programmcode wird zunächst ein neues DOMDocument instanziert und ein initialer Knoten angelegt, mit dem dann die eigentlichen XML-Knoten verknüpft werden:

```
((Beginn Programmcode))
DECLARE
  doc dbms_XMLDom.DOMDocument;
  last_dom_node dbms_XMLDom.DOMNode;
BEGIN
  doc := dbms_XMLDom.newDOMDocument; --DOMDocument anlegen
  last_dom_node := dbms_XMLDom.makeNode(doc); --initialen Vaterknoten anlegen
((Ende Programmcode))
```

Ein Unterknoten wird anschließend wie folgt angehängt:

```
((Beginn Programmcode))
element := dbms_XMLDom.createElement(doc, bezeichnung); --Erzeugen eines Elementes
node := dbms_XMLDom.makeNode(element); --Element in einen Node formen
last_dom_node := dbms_XMLDom.appendChild(last_dom_node, node); --Zuordnung zum
Vaterknoten
((Ende Programmcode))
```

Da das zu erstellende Dokument aufgrund des Basis-Selects je Knoten hierarchisch aufsteigend abgearbeitet wird, benötigt die Funktion einen „Merker“, der einen Verweis auf den zuletzt bearbeiteten Knoten in Abhängigkeit zur Hierarchiestufe beinhaltet. Dies wird über die Variablen „last_level“ (Speicherung, in welchem Level sich die Routine befindet) und last_dom_node (Verweis auf den zuletzt bearbeiteten Node) gesteuert.

Eine Besonderheit der XMLDOM ist, dass ein Element bzw. der daraus erzeugte Node (außer den evtl. zugeordneten Attributen) keinerlei Inhalt hat. Der eigentliche Wert innerhalb eines XML-Tags ist in einem seperaten Node (vom Typ #textnode) gespeichert. Dies hat den Grund, dass ein Tag/Element laut W3C-Spezifikation sowohl einen Value als auch weitere Untertags/Unterelemente besitzen kann. Daher ist der Wert eines Elements als eigener Subnode des Nodes definiert. Ein solcher Textnode wird in der XMLDOM über createTextNode erzeugt. Dieser Node wird anschliessend über appendChildNode an den zugehörigen Knoten angeheftet.

Da im Basisselect für die XML-Erzeugung Tagname und Tagvalue in einer Zeile ausgegeben werden, diese aber innerhalb des DOM als zwei Hierarchiestufen zu behandeln sind, werden einfach beide Stufen im gleichen Programmablauf verarbeitet: Wenn xml_tag_value gefüllt ist, wird gleichzeitig ein Subnode vom Typ #textnode mit dem gegebenen Wert zu diesem Knoten angelegt.

Im gesamten Workflow läuft die Erzeugung des XML-Dokumentes wie folgt ab:

1. Erzeugen eines XMLDOMDocuments.

2. Ausführen des Basis-Selects für die umzusetzende Struktur, innerhalb einer Schleife werden die zu verarbeitenden Zeilen (=Strukturelemente) verarbeitet.

3. Bestimmen der hierarchischen Position innerhalb des Dokumentes, eventuell Rücksprung auf übergeordnete Knoten.

4. Erzeugen des Nodes.

5. Eventuell hinzufügen von Attributen. Da die Attribute als Fließtext (Attribut="Value") in der Datenbank gespeichert sind, muss dieser String zunächst in die erforderlichen Fragmente zerlegt werden.

6. Erzeugung eines Subnodes vom Typ #textnode, sofern xml_tag_value gefüllt ist.

Sind alle Nodes abgearbeitet, müssen im XMLFile noch die Headerinformationen, wie z.B. die Zeichensatzangabe, aufgenommen werden. Soll nur der Zeichensatz gesetzt werden, so wird dafür die Funktion setCharset verwendet. Sollen noch Doctype-Informationen spezifiziert werden, so bleibt nur die „unschöne" Methode (siehe Code): Hinzufügen des statischen Headers als String nach Erzeugung eines CLOBs aus dem DOM. Der Grund: die Methode setDocType ist in der aktuellen Oracle-Version 10g nicht unterstützt.

Dynamische Generierung von XML in der Datenbank

Im Folgenden wird der komplette Code der dynamischen XML-Erzeugung dargestellt. Dieser ist in einer Funktion, die das fertige XML-File als CLOB zurückliefert, implementiert.

```
((Beginn Programmcode))
CREATE OR REPLACE
FUNCTION generate_xml(xfile_id IN xmlstruktur.file_id%TYPE) RETURN CLOB IS

  doc           dbms_XMLDom.DOMDocument;
  last_dom_node dbms_XMLDom.DOMNode;
  text_node     dbms_XMLDom.DOMNode;
  element       dbms_XMLDom.DOMElement;
  text          dbms_xmldom.domtext;
  xmlclob       CLOB;
  hxmlclob      CLOB;
  last_level    NUMBER := 1;
  attribtext    VARCHAR2(500);
  attribname    VARCHAR2(500);
  attribvalue   VARCHAR2(500);
BEGIN
  doc := dbms_XMLDom.newDOMDocument;
  last_dom_node := dbms_XMLDom.makeNode(doc);

  FOR i IN (SELECT LEVEL, xs.knoten_id, xs.vater_knoten_id,
                   xs.xml_tag, xs.xml_attributes, xs.xml_tag_value
            FROM   xmlstruktur xs
            WHERE  xs.file_id = xfile_id
            START WITH xs.vater_knoten_id IS NULL --ROOT-Tag
            CONNECT BY xs.vater_knoten_id = PRIOR xs.knoten_id
            ORDER BY xs.knoten_id)
  LOOP
    --Das alles Entscheidene ist die HierarchieStufe. Hier muss evtl. zum
übergeordneten Node zurückgesprungen werden.
    IF i.level < last_level THEN
      FOR h IN i.level .. last_level LOOP
        last_dom_node := dbms_XMLDom.getparentnode(last_dom_node);
      END LOOP;
    ELSIF last_level = i.level AND last_level > 1 THEN --auf 1 Ebene nicht! Nur
darüber liegende Ebenen. Sonst springen wir aus dem DOM hinaus...
      last_dom_node := dbms_XMLDom.getparentnode(last_dom_node);
    END IF;
    last_level := i.level;
    --Hier sind wir richtig!

    -- 1.Schritt: Tag erstellen
    ELEMENT := dbms_XMLDom.createElement(doc, i.xml_tag);
    last_dom_node := dbms_XMLDom.appendChild(last_dom_node,
dbms_XMLDom.makeNode(ELEMENT));
```

```
    --2.Schritt: Attribute setzen, sofern vorhanden:
    IF i.xml_attributes IS NOT NULL THEN
      attribtext := i.xml_attributes;
      --Attribut ausfiltern:
      WHILE INSTR(attribtext,'=') <> 0 LOOP
        attribname  := LTRIM(RTRIM(SUBSTR(attribtext,1,INSTR(attribtext,'=')-1),'
'),' ');
        attribvalue := LTRIM(RTRIM(SUBSTR(attribtext,
INSTR(attribtext,'"',1,1)+1,INSTR(attribtext,'"',1,2)-INSTR(attribtext,'"',1,1)-
1),' '),' ');
        attribtext  := LTRIM(RTRIM(SUBSTR(attribtext,
INSTR(attribtext,'"',1,2)+1),' '),' ');
        --Setzen des Attributes
        dbms_xmldom.setattribute(ELEMENT,attribname,attribvalue);
      END LOOP;
    END IF;

    -- 3. Schritt: TextElement erstellen als Inhalt von Tag
    IF i.xml_tag_value IS NOT NULL THEN
      text_node := dbms_XMLDom.appendChild(last_dom_node,
dbms_XMLDom.makeNode(dbms_XMLDom.createTextNode(doc,i.xml_tag_value)));
    END IF;
  END LOOP;

  dbms_XMLDom.writetobuffer(doc, xmlclob);
  --Header hinzufügen:
  hxmlclob := '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>'||CHR(10);
  Dbms_lob.append(hxmlclob, xmlclob);

  dbms_xmldom.freedocument(doc);
  RETURN hxmlclob;
EXCEPTION    WHEN OTHERS THEN
  RAISE_APPLICATION_ERROR(-20120, 'Fehler in generate_xml: '||SQLERRM);

END generate_xml;

((Ende Programmcode))
```

Das erzeugte XML-File:

```
((Beginn Programmcode))
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DOAG>
  <DOAGNEWS date="200501">
    <BEITRAG num="1">Überschrift 1</BEITRAG>
    <BEITRAG num="2">Überschrift 2</BEITRAG>
    <BEITRAG num="3">Überschrift 3</BEITRAG>
  </DOAGNEWS>
  <DOAGNEWS date="200502">
    <BEITRAG num="1">Überschrift 1</BEITRAG>
    <BEITRAG num="2">Überschrift 2</BEITRAG>
    <INSERAT>
      <TEST num="1">Test 1</TEST>
      <TEST num="2" testattib="test2">Test 2</TEST>
    </INSERAT>
  </DOAGNEWS>
</DOAG>

((Ende Programmcode))
```

Sollten Fragen zu diesem Artikel bestehen - ich stehe Ihnen gerne zur Verfügung:

Ihr Ansprechpartner:	Kontakt
Hakon Lugert	hakon.lugert@merlin-zwo.de
Systementwickler	Tel.: 0721 / 790 71 71 Fax: 0721 / 790 71 99
merlin.zwo InfoDesign GmbH & Co. KG Taglöhnergärten 43 76228 Karlsruhe	www.merlin-zwo.de
merlin.zwo InfoDesign GmbH & Co. KG Karmelstraße 9 75378 Bad Liebenzell	