# SQL/XML: extensies voor genereren data

## *XML ondersteuning in de Oracle database (1)*

***Dit is het eerste artikel in een reeks van artikelen over de XML ondersteuning in de Oracle database. In deze reeks zal Erwin Groenendal in detail, en aan de hand van veel voorbeelden, de mogelijkheden en toepassingen van XML technologie op het Oracle database platform introduceren. Aangenomen wordt dat de lezer basiskennis van XML heeft.***

De eerste drie artikelen in deze reeks zullen de onderstaande onderwerpen behandelen:

- SQL/XML: SQL extensies voor het genereren van XML data
- XMLType: Native XML datatype in de database
- XDB Repository: File system in de database

Daarna zullen nog aan de orde komen: editing van XML documenten, de PL/SQL DOM (Document Object Model) API (DBMS_DOM), werken vanuit Java met XML en XQuery.

### Historie

De geschiedenis van de XML ondersteuning op het Oracle platform gaat terug tot de eerste PL/SQL utility's voor het genereren van XML-data in 1999. Met de introductie van Java ondersteuning in Oracle8i werden deze utilities samen met andere componenten gebundeld in de XML Developer's Kit (XDK). Er kwamen aparte XDK's voor PL/SQL, Java, C en C++. Hierbij maakte de XDK voor PL/SQL gebruik van Java Stored Procedures. Deze moest apart geïnstalleerd worden en kon het best beschouwd worden als "een technology preview". Vanaf Oracle9i release 2 (9.2) bevindt de functionaliteit van de eerdere XDK voor PL/SQL zich standaard in de database. Deze is gebaseerd op "natively compiled" C code (en dus sneller). Deze versie (9.2) introduceerde de Oracle XML Database (XDB) en biedt "native" XML ondersteuning met de introductie van het XMLType datatype en de XDB Repository. De XDK voor PL/SQL bestaat vanaf deze versie van de database dan ook niet meer (de XDK's voor C++, C en Java nog wel).

### Oracle 10g

In Oracle 10g (10.0) is de XML ondersteuning in beperkte mate uitgebreid ten opzichte van Oracle9i release 2 (9.2). Wel zijn er veel verbeteringen in de technische implementatie doorgevoerd. Drie belangrijke functionele uitbreidingen zijn de ondersteuning van wijzigingen in XML Schema definities (dit zal in het tweede artikel van deze reeks aan de orde komen), verbeteringen in het omgaan met verschillende character sets en de introductie van de DBMS_XMLSTORE package voor het manipuleren van relationele data via XML. Deze package vervangt de "oude" XDK package DBMS_XMLSAVE.

### SQL/XML

SQL/XML is de naam voor een groep van functies waarmee SQL wordt uitgebreid met mogelijkheden om XML data te genereren met "normale" SQL query's. Deze functionaliteit vormt een extensie op de ANSI/ISO SQL 2003 standaard. In de implementatie in de Oracle database (vanaf 9.2) retourneren de SQL/XML functies een XMLType waarde (het native XML datatype en onderwerp van het tweede artikel in deze reeks). Opvallend is dat in de Oracle9i (9.2) XDB Developer's Guide de functies nog aangeduid werden als onderdeel van de *SQLX* standaard. De website van het SQLX standaardisatie "initiatief" is inmiddels, in ieder geval op het moment van het schrijven van dit artikel, niet meer beschikbaar. Blijkbaar is SQLX inmiddels opgegaan in SQL/XML.

***In Oracle 10g is de XML-ondersteuning in beperkte mate uitgebreid ten opzichte van Oracle 9i release 2***

De Oracle 10g XDB Developer's Guide waarschuwt, net als de overeenkomende Oracle9i (9.2) manual dit deed voor SQLX,

dat de SQL/XML functies aan wijzigingen onderhevig zijn omdat zij onderdeel zijn van een "emerging standard". Wat grasduinen op Internet geeft inzicht in andere implementaties van SQL/XML, onder andere in IBM DB2. Hierbij komt ook aan het licht dat er SQL/XML functies zijn die (nog) niet in de Oracle database geïmplementeerd zijn en dat de XMLElement functie in Oracle verschilt (in volgorde van parameters) van andere implementaties. De waarschuwing in de genoemde manual lijkt dus op zijn plaats.
Naast de SQL/XML functies ondersteunt de Oracle database nog andere manieren om XML data te genereren: de "oude" XDK package DBMS_XMLGEN en de Oracle specifieke functies (in tegenstelling tot de standaard SQL/XML functies) SYS_XMLGEN en SYS_XMLAGG en de bekende, en zeer gewaardeerde, XML SQL Utility (XSU). De laatste wordt ook gebruikt in het XSQL (met de "X" vóór "SQL") raamwerk voor het genereren van webpagina's in of via XML. Al deze methoden kunnen echter als 'achterhaald' gezien worden door de SQL/XML functionaliteit, die eenvoudig, handig, krachtig en goed werkt.

## SQL/XML functies

De onderstaande vijf SQL/XML functies worden in dit artikel behandeld.

- XMLElement
- XMLAttributes
- XMLAgg
- XMLForest
- XMLConcat

In de Oracle 10g XDB Developer's Guide worden ook XMLColAttVal en XMLSequence genoemd als SQL/XML functies. De eerste is een uitbreiding van Oracle op de SQL/XML standaard en komt ook aan de orde in dit artikel. XMLSequence heeft eigenlijk niets te maken met de generatie van XML data, maar is een essentiële functie bij het uitvoeren van query's op XMLType waarden. Deze functie wordt daarom beschreven in het tweede artikel in deze reeks.

## Voorbeelden

In de voorbeelden in dit artikel wordt gebruik gemaakt van het datamodel uit tabel 1. De tabel ACME_OFFICES bevat adresgegevens van kantoren (vestigingen) van een fictief bedrijf. Lijsten van kantoren worden opgeslagen in tabel ACME_OFFICES_LISTS, waarbij de kantoren in de lijst worden bijgehouden in ACME_OFFICES_LIST_ENTRIES.

```
create table ACME_OFFICES
( ID            NUMBER(10,0)   NOT NULL
, NAME          VARCHAR2(50)   NOT NULL
, ADDRESS       VARCHAR2(50)   NOT NULL
, SUITE         VARCHAR2(50)       NULL
, CITY          VARCHAR2(50)   NOT NULL
, ZIP           VARCHAR2(10)   NOT NULL
, MAIN_PHONE    VARCHAR2(15)   NOT NULL
, MAIN_FAX      VARCHAR2(15)   NOT NULL
)
/
create table ACME_OFFICES_LISTS
( ID            NUMBER(10,0)   NOT NULL
, DESCRIPTION   VARCHAR2(50)   NOT NULL
)
/
create table ACME_OFFICES_LIST_ENTRIES
( OFFICES_LIST_ID   NUMBER(10,0)   NOT NULL
, OFFICE_ID         NUMBER(10,0)   NOT NULL
)
/
```

*Tabel 1*

## XMLElement

De functie XMLElement wordt gebruikt om een XML element te genereren op basis van een kolomwaarde. In tabel 2 wordt dit geïllustreerd.

```
SQL> select xmlelement("CITY", city) xmldata
  2  from   acme_offices
  3 /

XMLDATA
--------------------------------------------------
<CITY>Costa Mesa</CITY>
<CITY>El Segundo</CITY>
<CITY>San Rafael</CITY>
<CITY>Rocklin</CITY>

4 rijen zijn geselecteerd
```

*Tabel 2*

De query in tabel 2 levert vier rijen op. In SQL*Plus worden de XMLType waarden –
alle SQL/XML functies leveren een XMLType waarde op – getoond als een string. Aanroepen van XMLElement kunnen genest worden. Met de query in tabel 3 wordt binnen het parent element 'OFFICE' een child element 'CITY' gegenereerd.

```
SQL> select xmlelement("OFFICE"
  2        ,         xmlelement("CITY", city)
  3        ) xmldata
  4  from   acme_offices
  5  /

XMLDATA
--------------------------------------------------
<OFFICE><CITY>Costa Mesa</CITY></OFFICE>
<OFFICE><CITY>El Segundo</CITY></OFFICE>
<OFFICE><CITY>San Rafael</CITY></OFFICE>
<OFFICE><CITY>Rocklin</CITY></OFFICE>

4 rijen zijn geselecteerd.
```

*Tabel 3*

Uiteraard kunnen er meer geneste XMLElement aanroepen gedaan worden en kan er dieper genest worden. Belangrijk om op te merken is nog dat op de plek waar "OFFICE" en "CITY" staan in de query in tabel 3 *geen* expressie gebruikt kan worden. Op deze plek is het in PL/SQL *niet* mogelijk om in een cursor een bind variabele neer te zetten – een poging om dit te doen leverde een XML element op met de naam 'b_element_name' … De documentatie bevestigt deze beperking. Het is dus geen parameter maar een alias zoals we die ook kunnen gebruiken van kolomnamen en tabelnamen in SQL query's.
Zoals al eerder opgemerkt verschilt de Oracle implementatie van XMLElement in syntax ook van andere implementaties. Bij deze implementaties is de syntax om eerst de kolomnaam te noemen en dan, zonder een komma te gebruiken, de alias (elementnaam) neer te zetten: `xmlelement(city "CITY")`. Deze syntax lijkt de correcte syntax te zijn en is ook logischer (en geeft geen ruimte om de alias te verwarren met een parameter).

## XMLAttributes

XMLAttributes is een functie die als optionele eerste parameter in XMLElement aangeroepen kan worden en waarmee attributen gegenereerd kunnen worden binnen een XML element. Een voorbeeld hiervan wordt gegeven in tabel 4.

```
SQL> select xmlelement("OFFICE"
  2        ,         xmlattributes(id as "ID")
  3        ,         xmlelement("CITY", city)
  4        ) xmldata
  5  from   acme_offices
  6  /

XMLDATA
--------------------------------------------------
<OFFICE ID="1"><CITY>Costa Mesa</CITY></OFFICE>
<OFFICE ID="2"><CITY>El Segundo</CITY></OFFICE>
<OFFICE ID="3"><CITY>San Rafael</CITY></OFFICE>
<OFFICE ID="4"><CITY>Rocklin</CITY></OFFICE>

4 rijen zijn geselecteerd.
```

*Tabel 4*

Uiteraard kunnen er meerdere attributen gegenereerd worden binnen hetzelfde element en kunnen er binnen het child element ook weer attributen gegenereerd worden met XMLAttributes.

## XMLAgg

Met XMLAgg kunnen aggregaties (collecties) van XML elementen gegenereerd worden. Deze functie wordt altijd gebruikt in combinatie met XMLElement. In tabel 5 wordt het typische gebruik van XMLAgg getoond: de eerste aanroep van XMLElement genereert een zogenaamde container-element ('OFFICES') en daarbinnen wordt een aggregatie (collectie) van 'OFFICE' elementen gegenereerd.

```
SQL> select xmlelement("OFFICES"
  2        ,         xmlagg(xmlelement("OFFICE"
  3                     ,         xmlelement("CITY", city)
  4                  )
  5             )
  6        ) xmldata
  7  from   acme_offices
  8  /

XMLDATA
--------------------------------------------------
<OFFICES><OFFICE><CITY>Costa Mesa</CITY></OFFICE><OFFICE><CITY>El Seg
undo</CITY></OFFICE><OFFICE><CITY>San Rafael</CITY></OFFICE><OFFICE><
CITY>Rocklin</CITY></OFFICE></OFFICES>

1 rij is geselecteerd.
```

*Tabel 5*

In SQL*Plus wordt standaard slechts een deel van een XMLType waarde getoond. Om de gehele gegenereerde waarde te zien moet de lengte van getoonde long-waarden hoger gezet worden, bijvoorbeeld op 10.000, met het commando: `set long 10000`. De query in tabel 5 resulteert in één rij door het gebruik van XMLAgg die als een group-functie werkt, net als min, max, count, et cetera. Door een group by clausule te gebruiken in een query kunnen er meerdere rijen geselecteerd worden. De volgorde van de elementen in de aggregatie kan worden aangegeven met een optionele tweede parameter in XMLAgg (de eerste parameter is een XMLType waarde). Door een bug in 9.2.0.4 kan deze parameter echter niet gebruikt worden in PL/SQL. In een SQL query, uitgevoerd in SQL*Plus, kan deze parameter wel gebruikt worden, maar dezelfde SQL query in een cursor in PL/SQL geeft een compileerfout.

## XMLForest

De hierboven behandelde functies XMLElement, XMLAttributes en XMLAgg zijn de belangrijkste en verreweg meest gebruikte SQL/XML functies. Voor de volledigheid worden hieronder de standaard SQL/XML functies XMLForest en XMLConcat en de Oracle specifieke functie XMLColAttVal nog

beschreven. XMLForest wordt gebruikt om een "forest" van XML elementen te genereren, dat wil zeggen een aantal XML elementen achter elkaar. Dit wordt geïllustreerd in tabel 6.

```
SQL> select xmlforest(city as "CITY"
  2       ,       zip as "ZIP"
  3       ) xmldata
  4  from   acme_offices
  5  /

XMLDATA
--------------------------------------------------
<CITY>Costa Mesa</CITY><ZIP>CA 92626</ZIP>
<CITY>El Segundo</CITY><ZIP>CA 90245</ZIP>
<CITY>San Rafael</CITY><ZIP>CA 94903</ZIP>
<CITY>Rocklin</CITY><ZIP>CA 95765</ZIP>

4 rijen zijn geselecteerd.
```

*Tabel 6*

De gegenereerde XMLType waarde is geen geldig XML document, dat immers een root element moet hebben. In feite wordt een zogenaamd XML fragment gegenereerd. XMLForest zal dus vooral genest gebruikt worden binnen XMLElement aanroep.

## XMLConcat

Met XMLConcat kunnen meerdere XMLType waarden aan elkaar "geplakt" worden. Net als bij XMLForest genereert deze functie een XML fragment. De query in tabel 7 selecteert (genereert) dezelfde rijen als de query in tabel 6.

```
SQL> select xmlconcat(xmlelement("CITY", city)
  2       ,       xmlelement("ZIP", zip)
  3       ) xmldata
  4  from   acme_offices
  5  /

XMLDATA
--------------------------------------------------
<CITY>Costa Mesa</CITY><ZIP>CA 92626</ZIP>
<CITY>El Segundo</CITY><ZIP>CA 90245</ZIP>
<CITY>San Rafael</CITY><ZIP>CA 94903</ZIP>
<CITY>Rocklin</CITY><ZIP>CA 95765</ZIP>

4 rijen zijn geselecteerd.
```

*Tabel 7*

## XMLColAttVal

Deze specifieke Oracle SQL/XML functie maakt het gemakkelijk om in een generiek XML-formaat gegevens te "exporteren". In de gegenereerde XML komen de kolomnamen en -waarden terecht van de kolommen die gespecificeerd worden als argumenten in de aanroep van XMLColAttVal. Een voorbeeld van het gebruik van deze functie wordt gegeven in tabel 8.

```
SQL> select xmlcolattval(city as "CITY"
  2       ,          zip as "ZIP"
  3       ) xmldata
  4  from   acme_offices
  5  /

XMLDATA
--------------------------------------------------
<column name = "CITY">Costa Mesa</column>
<column name = "ZIP">CA 92626</column>

<column name = "CITY">El Segundo</column>
<column name = "ZIP">CA 90245</column>

<column name = "CITY">San Rafael</column>
<column name = "ZIP">CA 94903</column>

<column name = "CITY">Rocklin</column>
<column name = "ZIP">CA 95765</column>

4 rijen zijn geselecteerd.
```

*Tabel 8*

## Gebruik in functies en views

Uiteraard kunnen er views gemaakt worden waarin SQL/XML functies gebruikt worden om XMLType kolommen in de view te "verkrijgen". Een goede aanpak is het ontwikkelen van functies die van SQL/XML gebruik maken om "stukjes" XML te genereren en deze functies weer te gebruiken in soortgelijke functies. Op deze manier wordt de XML generatie mooi modulair opgezet en worden de query's niet te complex. De 'acme' package in tabel 9a (specification) en 9b (body) illustreert deze aanpak.

```
create or replace
package acme is

function get_office_xml
(       p_office_id   in   number
)
return xmltype;

function get_offices_list_xml
(       p_offices_list_id   in   number
)
return xmltype;

end;
```

*Tabel 9a*

```
create or replace
package body acme is

function get_office_xml
(       p_office_id   in   number
)
```

```
return xmltype
is

  cursor c_office
  is
  select xmlelement("office"
       , xmlelement("name", name)
       , xmlelement("address", address )
       , xmlelement("suite", suite)
       , xmlelement("city", city )
       , xmlelement("zip", zip)
       , xmlelement("mainPhone", main_phone)
       , xmlelement("mainFax", main_fax)
       )
  from   acme_offices
  where  id = p_office_id;

  l_office   xmltype;

begin

  if p_office_id is null
  then
    return xmltype('<office/>');
  end if;

  open c_office;
  fetch c_office into l_office;
  close c_office;

  return l_office;


end get_office_xml;

function get_offices_list_xml
(       p_offices_list_id   in   number
)
return xmltype
is

  cursor c_offices_list
  is
  select xmlelement("officesList"
       , xmlagg(get_office_xml(office_id))
       )
  from   acme_offices_list_entries
  where  offices_list_id = p_offices_list_id;

  l_offices_list   xmltype;

begin

  open c_offices_list;
  fetch c_offices_list into l_offices_list;
  close c_offices_list;

  return l_offices_list;

end get_offices_list_xml;

end;
```

*Tabel 9b*

In de cursor 'c_offices_list' in functie 'get_offices_list_xml' wordt de functie 'get_office_xml' gebruikt. Om de functie aan te kunnen roepen in de cursor moet deze wel public zijn (door deze in de package specification op te nemen). De functies uit de 'acme' package kunnen als volgt gebruikt worden (tabel 10):

```
SQL> select acme.get_offices_list_xml(1) xmldata
  2  from   dual
  3  /

XMLDATA
--------------------------------------------------------------------------------
<officesList><office><name>Costa Mesa</name><address>600 Anton
Blvd.</address><s
uite>1400</suite><city>Costa Mesa</city><zip>CA
92626</zip><mainPhone>714.444.83
00</mainPhone><mainFax>714.444.8400</mainFax></office><office><name>El
Segundo</
name><address>222 N. Sepulveda Blvd.</address><suite>2300</suite><city>El
Segund
o</city><zip>CA
90245</zip><mainPhone>310.322.1771</mainPhone><mainFax>310.364.0
059</mainFax></office><office><name>San Rafael</name><address>1 Thorndale
Drive<
/address><suite>275</suite><city>San Rafael</city><zip>CA
94903</zip><mainPhone>
925.251.4702</mainPhone><mainFax>415.507.2099</mainFax></office><office><na
me>Ro
cklin</name><address>1001 Sunset Boulevard</address><suite></suite><city>Rocklin
</city><zip>CA
95765</zip><mainPhone>916.315.3500</mainPhone><mainFax>916.315.30
00</mainFax></office></officesList>
```

*Tabel 10*

## Toepassingen SQL/XML

SQL/XML kan toegepast worden in uiteenlopende oplossingen waar XML data gegenereerd moet worden op basis van relationele data. Hierbij kan gedacht worden aan het aanroepen van PL/SQL (packaged) stored functions, zoals in de 'acme' package uit tabel 9a en 9b, vanuit Java of "direct" gebruik van SQL queries met SQL/XML functies in JDBC. De XML generatie kan bijvoorbeeld benut worden voor de implementatie van web services of dynamische HTML web pagina's. Hierbij kan de gegenereerde XML naar HTML getransformeerd worden in Java, of in de database zelf (wat in het tweede artikel in deze reeks aan de orde komt). Ook kan de gegenereerde XML naar een file geschreven worden (wat beschreven wordt in het derde artikel in deze reeks).

**Erwin Groenendal** is oprichter en technisch directeur van Cumquat Information Technology en kan bereikt worden via erwin@cumquat.nl.