Lekker snel XML met SQL-functies

In steeds meer opdrachten kom je XML als requirement tegen. Omdat het makkelijk is of omdat de interface die je moet aanspreken het vereist. Dit is zeker het geval wanneer je webservices moet aanroepen. En ook dat komt steeds meer voor. Zelfs vanuit de database (PL/SQL). Maar hoe maak je nu even snel een XML-bestand van gegevens in de database. Je kunt dat op je gemak uitcoderen. Maar het kan nog eenvoudiger. Vanaf Oracle 9i Release 2 kennen we de XMLtype. En met de bijbehorende SQL functies is het karwei zo gepiept.

Over dit onderwerp is een mooi Technet artikel te lezen ('SQL in, XML out' door Jonathan Gennick, http://www.oracle.com/technology/oramag/oracle/03-may/o33xml.html). Het gaat om een aantal SQL-functies die je helpen om de klus te klaren.

XML Element

De XMLElement functie zorgt voor een xml-element of een node. Dat kan een rij uit de query omvatten maar ook de waarde uit een kolom. Dit ziet er bijvoorbeeld zo uit:

```
select xmlelement
( car
, xmlelement( "car-licence", license)
, xmlelement( "car-category", category)
, xmlelement( "build-year", year)
)
from cars
```

De functie heeft, zoals je ziet, minimaal twee parameters: de eerste is de naam van de tag, en de tweede (en eventueel volgende), is ofwel de output van een andere XML-functie (bijvoorbeeld een xmlelement), een kolom-waarde of constante dan wel functie-output. Hier zie je ook de hiërarchische opbouw terug. De output van bovenstaande query is als volgt:

```
XML
--------------------------------------------------------------------
----------------------------------
<CAR><car-licence>EJ32JEM</car-licence><car-category>2</car-
category><build-year>2001</build-<CAR><car-licence>YJJYR55</car-
licence><car-category>3</car-category><build-year>2003</build-<CAR><car-
licence>454JQA</car-licence><car-category>4</car-category><build-
year>2002</build-<CAR><car-licence>JH5U9471</car-licence><car-catego
ry>5</car-category><build-year>2000</build-
```

XMLAttributes

Naast Elementen kent XML ook attributen. Een attribuut is een eigenschap van een element. In een query ziet dat er zo uit:

```
select xmlelement
( car
, xmlattributes
( license as "license"
, category as "category"
, year as "year"
)
) xml
from cars
```

XMLAttributes geeft je de mogelijkheid om meerdere attribuut waarden van een element op te geven, op de manier als in de query aangegeven. Achter het keyword 'as' wordt de naam van het attribuut opgegeven. De output van de query is dan als volgt:

```
XML
------------------------------------------------------
<CAR license="EJ32JEM" category="2" year="2001"></CAR>
<CAR license="YJJYR55" category="3" year="2003"></CAR>
<CAR license="454JQA" category="4" year="2002"></CAR>
```

XMLForest

XMLForest geeft je de mogelijkheid om een Element te vullen met een lijst van relationele waarden.
Op het eerste gezicht is het een andere manier van opbouwen van jouw XML dan met het gebruik van de XMLElement functie:

```
select xmlelement
( car
, xmlforest
( brand as "brand"
, model as "model"
, city as "city"
, country as "country"
)
) xml
from cars
```

De output hiervan is:

```
XML
```

```
--------------------------------------------------------------------------
-------------------------------------------------
<CAR><brand>Peugeot</brand><model>406</model><city>Saint Louis,
Missouri</city><country>United <CAR><brand>Renault</
brand><model>Megane</model><city>Saint Louis, Missouri</
city><country><CAR><brand>Fiat</brand><model>Stilo</model><city>Saint
Louis, Missouri</city><country>United
```

Eigenlijk lijkt dit hetzelfde als de eerste query. Het verschil hier is dat in het geval van het gebruik van xmlelement een null waarde altijd leidt tot een lege node: </CITY>. Als je dit niet wilt, dan kun je xmlforest gebruiken: leidt een van de parameters tot null dan geeft XMLForest die node niet.

XMLAttributes & XMLForest

Deze twee kun je dan ook weer combineren in een query:

```
select xmlelement
( car
, xmlattributes
( license as "license"
, category as "category"
, year as "year"
)
, xmlforest
( brand as "brand"
, model as "model"
, city as "city"
, country as "country"
)
) xml
from cars
```

Met als output:

```
XML
--------------------------------------------------------------------------
-----------------------------------------------------
<CAR license="EJ32JEM" category="2" year="2001"><brand>Peugeot</
brand><model>406</model><city><CAR license="YJJYR55" category="3"
year="2003"><brand>Renault</brand><model>Megane</model><<CAR
license="454JQA" category="4" year="2002"><brand>Fiat</
brand><model>Stilo</model><city>
```

Merk ook de gelijkenis in notatie op tussen XMLAttributes en XMLForest. Een alternatief is de volgende query:

```
select xmlelement
( "Cars"
, xmlagg
( xmlelement
( car
, xmlattributes
( license as "License"
, category as "Category"
, year as "Year"
)
, CASE WHEN brand IS NULL THEN NULL
ELSE XMLElement("Brand", brand) END
, CASE WHEN model IS NULL THEN NULL
ELSE XMLElement("Model",model) END
, CASE WHEN city IS NULL THEN NULL
ELSE XMLElement("City", city) END
, CASE WHEN city IS NULL THEN NULL
ELSE XMLElement("Country", country)
END
)
)
from cars
```

XMLAgg

In bovengenoemde queries zijn de rijen eigenlijk nog afzonderlijke elementen. Die wil je natuurlijk kunnen samenvoegen tot een xmldocument met een omvattende node. Daarvoor is de functie XMLAgg bedoeld:

```
select xmlelement
( cars
, xmlagg
( xmlelement
( car
, xmlattributes
( license as "license"
, category as "category"
, year as "year"
)
, xmlforest
( brand as "brand"
, model as "model"
, city as "city"
, country as "country"
)
)
)
) XML
from cars
where license in ('79-JF-VP', 'JR8GG1')
```

De notatie is vergelijkbaar met xmlelement: eerst een parameter voor de element naam en dan de xml-waarden die dan worden gegroepeerd binnen het element. De output ziet er dan als volgt uit:

```
XML
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
-----------------------------------
<CARS><CAR license="79-JF-VP" category="1" year="2002"><brand>BMW</
brand><model>320D</year="2003"><brand>Renault</brand><model>Megane</
model><city>London</city><country>United
```

GetClobVal, GetStringval, Extract

Bovengenoemde queries leveren feitelijk een XMLType op, ook al suggereer ik hierboven dat het een tekstuele output is. Wanneer je de queries uitvoert in SQL*Plus, dan herkent SQL*Plus het XMLType en maakt er een tekstuele output van. In PL/SQL zou het er zo uit komen te zien:

```
declare
l_xml xmltype;
l_xml_clob clob;
begin
select xmlelement
( cars
, xmlagg
( xmlelement
( car
, xmlattributes
( license as "license"
, category as "category"
```

```
, year as "year"
)
, xmlforest
( brand as "brand"
, model as "model"
, city as "city"
, country as "country"
)
)
)
) XML
into l_xml
from cars
where license in ('79-JF-VP', 'JR8GG1');
l_xml_clob := l_xml.Getclobval;
dbms_output.put_line(dbms_lob.substr( l_xml_clob, 255));
dbms_output.put_line(l_xml.extract('/CARS/CAR/@license').getstringval);
end;
```

met als output:

```
<CARS><CAR license="79-JF-VP" category="1"
year="2002"><brand>BMW</brand><model>320D</model><city>Amsterdam</
city><country>
The Netherlands</country></CAR><CAR license="JR8GG1" category="3"
year="2003"><brand>Renault</brand><model>Megane</model><city>Londo
79-JF-VPJR8GG1
```

Met de methode GetClobVal van het XMLType object is de CLOB Representatie op te vragen van het xmltype. De methode Extract geeft je de mogelijkheid om met behulp van een XPATH expressie de XMLType uit te vragen. Extract levert feitelijk weer een XMLType op, en met getstringval kun je daar dan weer een Varchar2-representatie van opvragen. Je ziet in bovenstaand voorbeeld dat de output van het Extract-resultaat een concatenatie is van beide kenteken-nummers.

Namespaces

Namespaces maken het vaak een stuk ingewikkelder en zijn vaak onnnodig. Een expressive als '/level1/level2/level3' is immers veel simpeler dan '/ns1:level1/ns2:level2/ns3:level3'. Helemaal als je niet weet hoe je de betreffende namespaces behorende bij de afkortingen (ns1, ns2, ns3) moet opgeven.
Laatst wilde ik de resultaten van een BPEL Proces in de database opslaan om de onderdelen ervan te bevragen in een later proces. Ik wilde de resultaten als zodanig in een XMLtype opslaan zodat het niet nodig was om er een compleet data model voor te ontwerpen. Een simpele xmltype-tabel voldoet. Dus ik creeerde een pl/sql functie met de filenaam als parameter dat de juiste rij in de resultaat table opvroeg met de xmltype kolom met daarin de status van die file.
De xmltype.extract() functie is hier boven uitgelegd. Maar dan? Je kunt de namespaces vermijden door gebruik te maken van de local-name() xpath function, maar dan wordt het wat lastig om de preciese waarde van de betreffende file op te vragen, wanneer er de resultaten van meerdere files in zijn opgeslagen. Gelukkig heeft de extract() function nog een andere parameter, de namespace string:

```
extract(XMLType_instance IN XMLType,
XPath_string IN VARCHAR2,
namespace_string In VARCHAR2 := NULL) RETURN XMLType;
```

Deze namespace string kan de namespace declarations bevatten die kunnen worden gebruikt in een xpath expressie. De namespace declaratives zien er dan als volgt uit: namespace-shortage=uri. Bijvoorbeeld

```
ns1="http://www.example.org/namespace1 "
```

Je kunt meerdere declaraties opgeven, gescheiden door een spatie.

```
declare
   xp_no_data_found exception;
   pragma exception_init(xp_no_data_found, -30625);
   l_xpath varchar2(32767) := '/ns1:level1/ns2:level2/ns3:level3';
   l_nsmap varchar2(32767) := 'ns1="http://www.example.org/namespace1"
ns2="http://www.example.org/namespace2" ns3="http://www.example.org/
namespace3"';
   l_xml xmltype;
   l_clob clob;
begin
   l_xml := function_that_gets_an_xmltype_value();
   l_clob := l_xml.extract(l_xpath, l_nsmap).getclobval();
end;
```

Merk overigens op dat er eigenlijk twee extract-functies zijn: extract() en extractvalue(). Extract() retourneert een xmltype, welke weer ge-'sub-queried' kan worden. De extractvalue() function retourneert het datatype van de variable waaraan de value wordt toegekend. Met de xmltype functions getstringval(), getclobval(), etc. kun je overigens ook de betreffende waarde van een node van het xmltype opvragen. Er is echter een klein verschil tussen het van de getstringval() en corresponderende functions en Extractvalue. En dat is dat Extractvalue een zogenaamd "unescaped" waarde van de node (de encoding entities zijn vervangen), terwijl getstringval() de waarde teruggeeft met de entity encodings intact.

Tenslotte

Dit 'how-to'-artikel is aan de summiere kant, maar geeft je wel een handreiking om aan de slag te gaan. Eigenlijk kun je met bovengenoemde voorbeelden al bijna de hele wereld aan. Wat ik onhandig vind, is dat XMLType geen methodes heeft om procedureel door het document te lopen. Stel dat je op volgorde van jaar en merk door een van bovenstaande XMLType-objecten wilt lopen, dan gaat dat met XMLType lastig. Eigenlijk heb je alleen maar XPATH om informatie uit te vragen. Wil je dat een beetje handig doen, dan moet je toch naar de XML-Dom parser grijpen. Dat betekent dan dat je het XMLType eerst naar een CLOB moet omzetten en vervolgens met de XML-Dom parser verwerken.

Martien van den Akker is Technical Architect bij Darwin IT-Professionals.