XMLType

Native XML datatype in de database

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

In het eerste artikel is het gebruik van SQL/XML functies voor het genereren van XML data vanuit SQL behandeld. Met SQL/XML worden XML data verkregen op basis van relationele data in de database. In dit artikel komt het opslaan, queryen en manipuleren van XML data in de database zelf aan de orde. In het volgende artikel wordt het beheren van XML files (en andere files) in de XDB Repository (file system) beschreven. Daarna zullen nog het editen van XML documenten, de PL/SQL DOM (document Object Model) API, werken vanuit Java met XML en XQuery aan de orde komen.

XMLType

Het datatype XMLType is het belangrijkste component van de native XML ondersteuning in de Oracle XML database (Oracle XDB). Dit datatype dient voor de opslag van XML data en kan tevens gebruikt worden voor variabelen, parameters en return-waarden in PL/SQL. Het is een object type dat, naast de opslag van de XML data zelf, constructors en een aantal operaties (member functions en static functions) biedt die uitgevoerd kunnen worden op de XML data, zoals het uitvoeren van XPath expressies en het doen van XML Schema validaties en XSLT transformaties.

Constructors

Een XMLType waarde kan "gemaakt" worden met behulp van één van de vier constructors die geboden worden. Deze constructors maken een XMLType waarde op basis van een VARCHAR2 of CLOB waarde, een REF CURSOR of een abstract datatype (ADT). In het onderstaande voorbeeld wordt een XMLType waarde in PL/SQL "geconstrueerd" op basis van een (literal) VARCHAR2 waarde.

```
declare

   l_xmldoc   xmltype;

begin

   l_xmldoc := xmltype('<office id="1"><city>Costa
Mesa</city></office>');

end;
```

Lezers van het eerste artikel uit deze reeks hebben gezien hoe je met behulp van SQL/XML XML data kunt genereren met SQL queries. De met SQL/XML gegenereerde XML data zijn van het type XMLType. Hieronder wordt getoond hoe een XMLType waarde verkregen wordt op basis van een REF CURSOR.

```
declare

   type t_ref_cursor is ref cursor;

   c_offices   t_ref_cursor;

   l_xmldoc    xmltype;

begin

   open c_offices for 'select id, city from acme_offices';

   l_xmldoc := xmltype(c_offices);

   close c_offices;

   dbms_output.put_line(substr(l_xmldoc.getstringval(), 1, 255));

end;
```

In feite worden in bovenstaande voorbeeld XML data ook gegenereerd op basis van een query. Het resultaat is:

```
<ROWSET>
  <ROW>
    <ID>1</ID>
    <CITY>Costa Mesa</CITY>
  </ROW>
  <ROW>
    <ID>2</ID>
    <CITY>El Segundo</CITY>
  </ROW>
  <ROW>
    <ID>3</ID>
    <CITY>San Rafael</CITY>
  </ROW>
  <ROW>
    <ID>4</ID>
    <CITY>Rocklin</CITY>
  </ROW>
</ROWSET>
```

De gegenereerde XML is in het zogenaamde canonical formaat dat ook gebruikt wordt bij de XML SQL Utility (XSU, onderdeel van de XDK). In dit formaat wordt een <ROWSET> tag (element) gegenereerd "om" het hele resultaat en een <ROW> tag voor iedere rij die de query oplevert. De naam van de kolom (of alias uit de query) wordt gebruikt als naam voor de elementen die voor iedere waarde in de rij gegenereerd worden (ID en CITY). In tegenstelling tot SQL/XML heb je hier dus heel weinig controle over het XML formaat. Ook bij de constructie van een XMLType waarde op basis van een abstract datatype wordt XML in een canonical formaat gegenereerd. De CLOB constructor moet gebruikt worden voor XML documenten groter dan 32K.

XML dataopslag

XMLType kan op een "normale" (native) manier gebruikt worden bij het aanmaken van een tabel, zoals in onderstaand voorbeeld geïllustreerd wordt:

```
create table ACME_OFFICES_XML
( id       number(10,0)   not null
, xmldoc   xmltype        not null
)
```

Het is ook mogelijk om een tabel van XMLType te maken zónder kolommen. De meeste voorbeelden in de documentatie zijn hierop gebaseerd, en dit kan verwarrend zijn omdat de syntax van diverse acties anders is dan bij een tabel met een XMLType kolom. In de meeste toepassingen wordt gebruik gemaakt van XMLType kolommen omdat het handig is om naast de XML data nog relationele data te hebben die los van de XML data gebruikt en gemanipuleerd kan worden. Bij de XDB Repository is het gebruik van XMLType tabellen juist wel aan de orde (dit wordt behandeld in het volgende artikel).

Nu de tabel gecreëerd is kan er XML data opgeslagen worden. Hieronder wordt dit gedaan door in de insert statements gebruik te maken van de VARCHAR2 constructor.

```
insert into acme_offices_xml values
(1, xmltype('<office id="1"><city>Costa Mesa</city></office>'))
/
insert into acme_offices_xml values
(2, xmltype('<office id="2"><city>El Segundo</city></office>'))
/
insert into acme_offices_xml values
(3, xmltype('<office id="3"><city>San Rafael</city></office>'))
/
insert into acme_offices_xml values
(4, xmltype('<office id="4"><city>Rocklin</city></office>'))
/
```

Query's

Op de opgeslagen XML data kunnen met behulp van een drietal SQL functies queries uitgevoerd worden: EXTRACT, EXTRACTVALUE en EXISTSNODE. Met EXTRACT kan een XPath expressie uitgevoerd worden op de XMLType kolom:

```
SQL> select extract(xmldoc, '/office/@id').getnumberval()            id
  2  ,      extract(xmldoc, '/office/city/text()').getstringval()
city
  3  from   acme_offices_xml
  4  /

  ID CITY
---- --------------------------------------------------
   1 Costa Mesa
   2 El Segundo
   3 San Rafael
   4 Rocklin
```

EXTRACT levert een XMLType waarde op. Hoewel deze in SQL*Plus als een string getoond zal worden, worden in het bovenstaande voorbeeld de XMLType waarden met (de member functions) 'getnumberval()' en 'getstringval()' omgezet naar betreffende scalaire waarden. Merk verder nog op dat om de waarde van het element 'city' op te halen expliciet de text node geselecteerd wordt met 'text()'. Een verkorte manier om scalaire waarden te selecteren is met behulp van EXTRACTVALUE:

```
SQL> select extractvalue(xmldoc, '/office/@id')    id
  2  ,      extractvalue(xmldoc, '/office/city')   city
  3  from   acme_offices_xml
  4  /

  ID CITY
---- --------------------------------------------------
   1 Costa Mesa
```

```
   2 El Segundo
   3 San Rafael
   4 Rocklin
```

EXTRACTVALUE zet het (XMLType) resultaat van de XPath expressie om in een scalaire waarde. Voor een element hoeft niet meer expliciet de text node geselecteerd te worden. Het element mag echter geen child elementen hebben en mag dus alleen een text node als content bevatten. Bovenstaande queries selecteren alle vier de rijen in de tabel. Om te "zoeken" op de XML data kan EXISTSNODE gebruikt worden. EXISTSNODE is een functie die 0 of 1 oplevert afhankelijk van het feit of de XPath expressie een resultaat oplevert. In het onderstaande voorbeeld worden de rijen geselecteerd waarvoor geldt dat het 'city' element de waarde 'El Segundo' bevat.

```
SQL> select id
  2  from   acme_offices_xml
  3  where  existsnode(xmldoc, '/office[city = "El Segundo"]') = 1
  4  /

  ID
----
   2
```

Hetzelfde resultaat kan overigens net zo goed geselecteerd worden door gebruik te maken van EXTRACTVALUE in de where clause:

```
select id
from   acme_offices_xml
where  extractvalue(xmldoc, '/office/city') = 'El Segundo'
```

Of zelfs door de member function 'extract' te gebruiken, waarbij het gebruik van een alias voor de tabel verplicht is (!):

```
select id
from   acme_offices_xml t
where  t.xmldoc.extract('/office/city/text()').getstringval() = 'El
Segundo'
```

Als de XML data een collection bevatten, dan kan XMLSEQUENCE in combinatie met de TABLE function gebruikt worden om een pseudo tabel te maken met de onderdelen van de collection als aparte records. Stel dat de XML data er als volgt uit ziet:

```
<office id="1">
   <city>Costa Mesa</city>
   <employees>
      <employee id="101" name="Smith"/>
      <employee id="102" name="Doe"/>
   </employees>
</office>
```

De volgende query levert dan één record op. De XPath expressie levert weliswaar twee elementen op, maar deze bevinden zich binnen één record.

```
SQL> select t.xmldoc.extract('/office/employees/employee')
  2  from   acme_offices_xml t
  3  where  id = 1
  4  /

T.XMLDOC.EXTRACT('/OFFICE/EMPLOYEES/EMPLOYEE')
--------------------------------------------------------
<employee id="101" name="Smith"/>
<employee id="102" name="Doe"/>


1 row selected.
```

Om de elementen in aparte records te selecteren (hetgeen handig kan zijn in PL/SQL) kan de onderstaande query gebruikt worden:

```
SQL> select value(p).extract('/')
  2  from   acme_offices_xml t
  3  ,      TABLE(XMLSEQUENCE(EXTRACT(xmldoc,
'/office/employees/employee'))) p
  4  where  id = 1
  5  /

VALUE(P).EXTRACT('/')
--------------------------------------------------------------------------
-------
<employee id="101" name="Smith"/>
<employee id="102" name="Doe"/>

2 rows selected.

SQL>
```

XMLSEQUENCE zet de elementen die de EXTRACT van 'office/employees/employee' oplevert om zodat deze met TABLE naar een pseudo tabel omgevormd kunnen worden, met een apart record voor iedere onderdeel van de collection. In de select list kan vervolgens de member function 'extract' worden uitgevoerd op 'value(p)', waarmee een XMLType waarde in een record uit de pseudo tabel wordt aangeduid.

Object-relationele opslag

In bovenstaande voorbeelden worden de XML data intern als een CLOB opgeslagen. Voor toepassingen waarbij XML data

alleen moeten worden opgeslagen en als geheel weer opgehaald wordt, voldoet dit prima. Ook kunnen er bij opslag als CLOB indexen aangemaakt worden om het zoeken op de XML data te optimaliseren. Desondanks is voor veel toepassingen object-relationele opslag van XML data te prefereren. Dit is waarschijnlijk de meest krachtige en aansprekende functionaliteit in Oracle XDB.
Om XML data object-relationeel op te kunnen slaan moet een XML Schema definitie van de XML data gemaakt worden. In de tabel met de XMLType kolom uit de eerdere voorbeelden kan een willekeurig well formed XML document worden opgeslagen. Naast informatie over kantoren kan dus bijvoorbeeld ook informatie over auto's worden opgeslagen. In onderstaand voorbeeld wordt eerst geprobeerd om een niet-well formed XML document op te slaan en wordt daarna, met succes, een XML document met informatie over een auto toegevoegd aan de tabel.

```
SQL> insert into acme_offices_xml values
  2  (5, xmltype('<office id="5"><city>Grand Junction</office>'))
  3  /
(5, xmltype('<office id="5"><city>Grand Junction</office>'))
    *
ERROR at line 2:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00225: end-element tag "office" does not match start-element tag
"city"
Error at line 1
ORA-06512: at "SYS.XMLTYPE", line 0
ORA-06512: at line 1


SQL> insert into acme_offices_xml values
  2  (5,
xmltype('<car><make>Lamborghini</make><model>Gallardo</model></car>'))
  3  /

1 row created.

SQL>
```

Door gebruik te maken van XML Schema zal dit niet meer mogelijk zijn. Het XML document hoeft dan namelijk niet alleen well formed zijn, maar moet valide zijn ten opzichte van de XML Schema definitie. De XML Schema definitie voor het voorbeeld met de kantoren ziet er als volgt uit:

```
<?xml version="1.0"?>
<xs:schema elementFormDefault="unqualified"
           attributeFormDefault="unqualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="office" type="officeType"/>
   <xs:complexType name="officeType">
      <xs:sequence>
         <xs:element name="city" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:positiveInteger"/>
   </xs:complexType>
</xs:schema>
```

Om dit XML schema te kunnen gebruiken in Oracle XDB moet deze eerste geregistreerd worden. Dit wordt gedaan met de DBMS_XMLSCHEMA database package. Hierbij krijgt het schema een naam waarnaar later verwezen zal worden. Voor deze naam is het gebruikelijk om de URL syntax te gebruiken. Deze URL hoeft niet echt te bestaan.

```
begin
   dbms_xmlschema.registerschema('http://www.cumquat.nl/xsd/office',
      '<?xml version="1.0"?>
       <xs:schema elementFormDefault="unqualified"
          ...
       </xs:schema>'
   );
end;
```

Bij het registreren van het schema genereert Oracle XDB een object-relationele opslagstructuur. Deze opslagstructuur kan bestaan uit types, nested tables en andere object-relationele database objecten. De ontwikkelaar kan deze objecten als een "black box" zien, maar zal in sommige situaties hier toch inzicht in willen en moeten hebben, bijvoorbeeld in het geval er (relationele) constraints toegevoegd moeten worden. Er worden altijd nieuwe objecten aangemaakt. Het is niet mogelijk om een mapping te maken van het schema naar bestaande database objecten. Wel is het mogelijk om annotations (of beter gezegd, extra attributen) op te nemen in het schema om naamgeving te bepalen en het aanmaken van de objecten enigszins te sturen. Zo kan bijvoorbeeld door de ontwikkelaar bepaald worden of een collection in het XML schema als een nested table of als een array geïmplementeerd wordt.

Opslagstructuur

Na het registreren van het XML schema kan de tabel uit de bovenstaande voorbeelden opnieuw aangemaakt worden, maar nu met een verwijzing naar het geregistreerde schema:

```
create table ACME_OFFICES_XML
( id        number(10,0)   not null
, xmldoc    xmltype        not null
)
xmltype column xmldoc xmlschema "http://www.cumquat.nl/xsd/office"
element "office"
```

Bij het inserten van records moet de XML data nu een verwijzing bevatten naar het XML schema (door gebruik te maken

van de 'xsi' namespace). Een handig alternatief wordt geboden door de member function 'createschemabasedxml' van XMLType. Door deze method uit te voeren op de XMLType waarde wordt hetzelfde bereikt.

```
insert into acme_offices_xml values
( 1 , xmltype('<office id="1">
                    <city>Costa Mesa</city>
                </office>'
      ).createschemabasedxml('http://www.cumquat.nl/xsd/office')
)
```

Het is nu niet meer mogelijk om "hele andere" XML documenten, die niet valid zijn ten opzichte van het XML schema, toe te voegen. Wanneer we bijvoorbeeld XML data proberen toe te voegen met informatie over een auto levert dat de volgende foutmelding op:

```
ERROR at line 2:
ORA-31043: Element 'car' not globally defined in schema
'http://www.cumquat.nl/xsd/office'
ORA-06512: at "SYS.XMLTYPE", line 0
ORA-06512: at line 1
```

Queries die gebruik maken van EXTRACT, EXTRACTVALUE of EXISTSNODE zullen bij object-relationele opslag herschreven worden naar queries op de onderliggende object-relationele structuur (query rewrite), en zodoende profiteren van de standaard query-optimalisatiefunctionaliteit van Oracle. Deze opslagstructuur kan ook direct benaderd worden in queries, zoals onderstaande query laat zien. Hierbij wordt het pseudo attribuut 'xmldata' gebruikt om de object relationele structuur te benaderen.

```
SQL> select t.xmldoc.xmldata."city" city
  2  from acme_offices_xml t
  3  /

CITY
--------------------------------------------------
Costa Mesa
El Segundo
San Rafael
Rocklin

SQL>
```

Voor complexere XML schema's is meer inzicht nodig in de mapping die Oracle XDB hanteert van de onderdelen van het schema op objectrelationele database-objecten. Inzicht in deze mapping is ook nodig voor het definiëren van relationele constraints. De beperkte ondersteuning van XML Schema voor constraints kan worden aangevuld met relationele constraints. Hierbij moet vooral gedacht worden aan check constraints en foreign key constraints. Constraints moeten gedefinieerd worden door gebruik te maken van de objectrelationele ("dot") notatie zoals in bovenstaande query, en zijn ook alleen maar mogelijk bij objectrelationele opslag. In het voorbeeld hieronder wordt een unique constraint gedefinieerd op het element 'city'. Ook XML Schema ondersteunt uniqueness constraints, maar deze geldt dan voor bínnen het XML document zelf. De relationele constraint dwingt af dat de waarde uniek is over de verschillende XML documenten die opgeslagen worden.

```
alter table acme_offices_xml
add constraint uk1 unique (xmldata."city")
```

Bij de definitie van indexen hoeft niet gebruik gemaakt te worden van de objectrelationele notatie, maar kan gebruik gemaakt worden van de member functions en XPath expressies:

```
create index i_city
on acme_offices_xml (xmldoc.extract('/office/city/text()').getstring-
val())
```

XML Schema validatie

Oracle XDB voert geen vólledige validatie uit wanneer een XML document toegevoegd wordt aan een tabel, ook al is de XMLType kolom gebaseerd op een XML schema. Hiervoor is door Oracle gekozen om de overhead zoveel mogelijk te beperken. In de meeste toepassingen zal de XML data ook al eerder (volledig) gevalideerd zijn, en is deze implementatiekeuze dus begrijpelijk. Indien er toch een volledige validatie uitgevoerd moet worden kan dit heel eenvoudig in een check constraint of trigger gedaan worden.
In onderstaand voorbeeld wordt een XML document toegevoegd dat niet valid is, de 'id' moet namelijk een positief geheel getal zijn volgens de XML Schema definitie.

```
SQL> insert into acme_offices_xml values
  2  (5, xmltype('<office id="-5"><city>San Jose</city></office>').cre-
ateschemabasedxml('http://www.
cumquat.nl/xsd/office'))
  3  /

1 row created.

SQL>
```

Het record wordt succesvol toegevoegd omdat er geen volledige validatie wordt gedaan. In feite controleert (valideert) Oracle

XDB niet veel meer dan of het XML document in de object-relationele structuur "past". Onjuiste datatypes (string in plaats van number), niet-gedefinieerde elementen en attributen et cetera leiden allemaal tot een fout, maar sub-datatypes (zoals de 'positiveInteger' in bovenstaand voorbeeld) en facetten worden niet afgedwongen. Hieronder is te zien hoe de volledige validatie kan worden uitgevoerd met een trigger of een check constraint en welke foutmeldingen er dan gegeven worden.

```
create or replace
trigger acme_bri
before insert
on acme_offices_xml
for each row
begin
   :new.xmldoc.schemaValidate();
end;
```

```
ERROR at line 1:
ORA-31154: invalid XML document
ORA-19202: Error occurred in XML processing
LSX-00210: value "-5" out of range for type "positiveInteger"
ORA-06512: at "SYS.XMLTYPE", line 0
ORA-06512: at "OPTIMIZE.ACME_BRI", line 2
ORA-04088: error during execution of trigger 'OPTIMIZE.ACME_BRI'
```

```
alter table acme_offices_xml
add constraint acme_ck1 check (XMLISVALID(xmldoc) = 1)
<< einde kader met computercode >>

<< begin kader met computercode >>
ERROR at line 1:
ORA-02290: check constraint (OPTIMIZE.ACME_CK1) violated
```

In de bovenstaande trigger wordt gebruik gemaakt van de member function 'schemaValidate' en in de check constraint van de SQL functie XMLISVALID.

Schema evolution

Een geregistreerd XML schema kan niet gewijzigd worden. Bij wijzigingen in het schema moeten de volgende stappen doorlopen worden:

- Registreer tijdelijk het nieuwe schema onder een andere naam.
- Maak een tijdelijke tabel op basis van dit schema.
- Kopieer de records naar deze tijdelijke tabel en maak hierbij eventueel gebruik van een XSLT stylesheet om de XML van het oude schema te transformeren naar het nieuwe schema. Dit is alleen nodig wanneer instanties van het oude schema niet valid zijn ten opzichte van het nieuwe schema.
- Drop de oude tabel.
- Drop het oude schema.
- Registreer het nieuwe schema onder de oorspronkelijke naam.
- Maak de nieuwe tabel aan onder de oorspronkelijke naam op basis van het nieuwe schema.
- Kopieer de records uit de tijdelijke tabel naar deze nieuwe tabel.
- Drop de tijdelijke tabel.
- Drop het tijdelijk geregistreerde schema.

In Oracle 10g kunnen deze stappen geautomatiseerd uitgevoerd worden met de 'copyEvolve' procedure in de package DBMS_XMLSCHEMA. Deze procedure kent (uiteraard) wel een aantal beperkingen en "aandachtspunten".
Ondanks de introductie van 'copyEvolve' ondersteunt Oracle XDB dus geen schemawijzigingen. Het is namelijk niet mogelijk om een schema te wijzigen zonder opnieuw de tabel te moeten aanmaken. Er zijn natuurlijk genoeg wijzigingen in XML schema's te bedenken waarvoor je ook niet kunt verwachten dat je deze kunt doorvoeren zonder de tabel opnieuw aan te moeten maken, net zoals dit geldt voor bepaalde wijzigingen in relationele datastructuren. Het is echter wel jammer dat ook voor eenvoudige wijzigingen bovengenoemde stappen doorlopen moeten worden. Het toevoegen van bijvoorbeeld een verplicht child element of een verplicht attribuut zou toch gemakkelijker moeten kunnen. Bij een normale relationele tabel zou dit gedaan worden door een optionele kolom toe te voegen, de kolom te vullen en de kolom verplicht te maken. Bij een vergelijkbare wijziging in een XML schema zouden de te volgen stappen kunnen zijn: wijzig het geregistreerde schema door een optioneel child element toe te staan, update de XML documenten door het child element toe te voegen en wijzig nogmaals het geregistreerde schema door het child element verplicht te maken.
Native XML databases die hun data niet-specifiek object-relationeel opslaan zoals Oracle XDB, maar in een generieke object model, gaan gemakkelijker om met schemawijzigingen. Deze databases missen echter weer een hoop voordelen die Oracle XDB biedt.

Piecewise update

Opgeslagen XML data kan op twee manieren geupdate worden. Ten eerste door de XMLType waarde in zijn geheel te vervangen door een nieuwe waarde. Dit is ook de enige optie bij opslag van de XMLType waarde als een CLOB. Bij object-relationele opslag is het ook mogelijk om een zogenaamde piecewise update te doen waarbij alleen een deel van de XML data wordt vervangen. Met name voor grotere XML documenten is dit een belangrijke optimalisatie. In het onderstaande voorbeeld wordt een piecewise update gedaan van het 'city' element.

```
update acme_offices_xml
set    xmldoc = UPDATEXML(xmldoc
                 ,         '/office/city'
                 ,         xmltype('<city>Menlo Park</city>')
                 )
where  id = 3
```

De piecewise update wordt uitgevoerd met behulp van de function UPDATEXML. De eerste parameter is de XMLType waarde die als uitgangspunt wordt genomen, de tweede parameter de XPath expressie die node(s) aanwijst die geupdate moeten worden en de derde de nieuwe node.

Transformaties

Naast XML Schema en XPath ondersteunt Oracle XDB vanzelfsprekend ook XSLT. Transformaties kunnen worden uitgevoerd met de member function 'transform'. Aan deze function wordt het XSLT stylesheet dat toegepast moet worden als een XMLType waarde meegegeven. De function wordt uitgevoerd op de XMLType waarde zelf en het resultaat van de transformatie wordt terug gegeven. In het onderstaande voorbeeld wordt een XMLType waarde getransformeerd waarbij het 'city' element wordt omgezet naar een gelijknamig attribuut.

```
declare

   l_xslt_stylesheet   xmltype;
   l_xmldoc             xmltype;
   l_xmldoc_result     xmltype;

begin

   l_xslt_stylesheet := xmltype(
'<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
                 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0"/>
  <xsl:template match="/">
   <office>
    <xsl:copy-of select="office/@id"/>
    <xsl:attribute name="city">
     <xsl:value-of select="office/city"/>
    </xsl:attribute>
   </office>
  </xsl:template>
 </xsl:stylesheet>
');

   l_xmldoc := xmltype('<office
id="123"><city>Sacramento</city></office>');

   l_xmldoc_result := l_xmldoc.transform(l_xslt_stylesheet);

   dbms_output.put_line(substr(l_xmldoc_result.extract('/').getstring-
val()
                          ,      1
                          ,      255
                          )
   );

end;
```

Het resultaat van de transformatie is:

```
<office id="123" city="Sacramento"/>
```

In het bovenstaande voorbeeld worden de XML data getransformeerd naar XML in een ander formaat. Het resultaat kan natuurlijk ook HTML zijn of een ander tekstformaat.

Toepassingen

In de Oracle database kan XML data op een efficiënte, native, manier worden opgeslagen, gemanipuleerd en opgevraagd. Door gebruik te maken van indexen kunnen queries op de XML data net zo optimaal worden uitgevoerd als op normale relationele data. Dit is met name interessant voor toepassingen waarbij een bedrijf of organisatie XML data ontvangen, deze data niet hóeven op te slaan in bestaande relationele datastructuren, maar wel efficiënt wil kunnen queryen. Het is dus niet nodig om de XML data om te zetten in relationele data en de inspanning om deze programmatuur te realiseren kan dus uitgespaard worden. Ook als de XML data verwerkt worden door andere systemen kan de oorspronkelijke XML data opgeslagen worden als een XMLType ten behoeve van het verkrijgen van management informatie en het doen van data-analyse. Bijkomende voordelen zijn de mogelijkheden die door Oracle XDB geboden worden om XML data en relationele data te integreren, onder andere door middel van constraints en triggers, en zodoende krachtige "hybride" oplossingen te creëren. Door gebruik te maken van Oracle XDB kan XML data de rol van relationele data vervangen in sommige toepassingen. In traditionele toepassingen zullen relationele data uiteraard de voorkeur blijven houden. Bij deze oplossingen gaat het om de bekende transactionele data en klassieke applicatieobjecten zoals 'afdeling', 'order' en 'productgroep': "data centric" informatie. Echter, voor "document centric" informatie (tekst, documenten, nieuwsberichten, forum entries, et cetera) geldt dat XML de aangewezen manier is om de informatie vast te leggen. XML is namelijk bij uitstek geschikt om tekst met abstracte opmaak en verrijking vast te leggen. Oracle XDB geeft dan ongekende mogelijkheden in document centric oplossingen, zoals semantisch zoeken, het selecteren van tekstonderdelen en het leggen (én afdwingen) van relaties vanuit tekst naar relationele data.

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