Cursistenmap

XML voor Oracle ontwikkelaars 11*g*

Dit naslagwerk kunt u na afloop van de cursus meenemen.

Versie 3.1 april-2011

Inhoudsopgave

Inhoudsopgave

Inhoudsopgave

Inhoudsopgave

1 Inleiding

De afgelopen decennia heeft de industrie XML omarmd als platform- en applicatieneutrale taal voor overdracht van gegevens via het internet. XML werd in theorie de oplossing voor het integreren van systemen. In de praktijk dienden echter nog lastige obstakels te worden overwonnen.

Eén obstakel was de interactie met de relationele database. In Oracle8 was het wel mogelijk om een XML-document in z'n geheel in een CLOB kolom op te slaan, maar de performance was matig en om de gegevens uit te lezen of te wijzigen was veel eigen code nodig. De ontwikkelaar diende zelf een interface te bouwen in Java of PL/SQL.
Het alternatief was om de gegevens uit het XML document te filteren en deze in relationele tabellen op te slaan. Dat maakte verwerking van de gegevens eenvoudiger. Het werd de aanbevolen manier om XML documenten met een simpele regelmatige structuur in de database op te slaan. Daarbij ging echter wel XML-functionaliteit verloren, zoals de positie van elementen in de XML boomstructuur, en de mogelijkheid om het document aan een XML Schema te toetsen.

Sinds Oracle9i bestaat het type XMLType en een XML repository: XMLDB. Deze uitbreidingen maken van het Oracle RDBMS een echte XML database. We kunnen XML bestanden eenvoudig met behulp van een internetverbinding in de database laden. XML documenten kunnen zonder extra code met behulp van het XMLType relationeel of als CLOB worden opgeslagen. Daarbij blijft de oorspronkelijke XML-functionaliteit behouden. XMLType member methoden maken het mogelijk gegevens uit een XMLType te benaderen, aan te passen of te transformeren. Met behulp van standaard SQL/XML functies kunnen we relationele gegevens omzetten naar XML formaat. We kunnen XML Schema's registreren in de database en op basis daarvan tabellen aanmaken en XML gegevens valideren.

Aan de hand van voorbeelden en opdrachten zult u in de cursus ervaring opdoen met bovengenoemde onderwerpen. In een inleidend hoofdstuk maakt u kennis met het type XMLType en met de XMLDB repository. Vervolgens behandelen we de standaard SQL/XML functies waarmee we XML uit relationele gegevens kunnen genereren. Hierna maken we XMLType tabellen en kolommen aan in de database. De XMLType gegevens kunnen we met standaard XPATH expressies in member functies benaderen. De mogelijkheden met betrekking tot het gebruik van XQuery zijn in Oracle 11g sterk toegenomen. We besteden een apart hoofdstuk aan deze mogelijkheden, met name voor de functies XMLQuery() en XMLTable(). Tabellen gebaseerd op XML Schema's bieden extra krachtige mogelijkheden, zoals het valideren van gegevens in triggers en constraints. Deze worden in het voorlaatste hoofdstuk behandeld. Het laatste hoofdstuk gaat over de PL/SQL-packages die deel uit maken van de XML SQL Utility. DBMS_XMLStore vergemakkelijkt het laden van XML gegevens in relationele tabellen. DBMS_XMLGen helpt bij het genereren van XML gegevens uit relationele tabellen.

Na afloop van de cursus zult u in uw eigen XML applicaties volop gebruik kunnen maken van de Oracle database voor het opslaan, verwerken en genereren van XML gegevens.

1 Inleiding

2 XML in de database

2.1 Inleiding

Sinds Oracle9*i* vormen XMLDB en XMLType de XML interface tussen het operating systeem en de database. In dit hoofdstuk bespreken we de structuur van XMLDB en de manier waarop XML bestanden in XMLDB kunnen worden geladen. We maken een WebDAV verbinding aan, en zullen een XML document via FTP in XMLDB opslaan. Ook wordt uitgelegd hoe een XML document in XMLDB kan worden benaderd vanuit SQL. Verder geeft dit hoofdstuk vast een inleiding op het nieuwe type XMLType. Dit onderwerp zullen we in volgende hoofdstukken nader bespreken.

2.2 XMLDB

Oracle XMLDB is een uitbreiding op de Oracle database. Deze uitbreiding vereenvoudigt het opslaan en het ophalen van XML gegevens. Hierdoor wordt het zowel een XML database, als een relationele database. XMLDB heeft onder meer de volgende kenmerken:

- Het ondersteunt de XML en XML Schema datamodellen, zoals deze zijn geformuleerd door het World Wide Web Consortion (W3C). XMLDB biedt standaard methoden voor het benaderen van XML gegevens.

- Het maakt het mogelijk XML gegevens op te halen, te wijzigen, op te slaan, of anderszins te bewerken. XML gegevens kunnen met SQL functies worden benaderd. Andersom kunnen ook relationele data met behulp van XML functies worden bewerkt.

- XML gegevens kunnen in een repository worden opgeslagen. Dit is een samenhangende verzameling tabellen waarin de eigenschappen van de XML gegevens zijn vastgelegd. We kunnen de XML gegevens in de repository benaderen alsof het bestanden in folders zijn. FTP, HTTP, en WebDAV ondersteuning maken het eenvoudig om XML-data van en naar de Oracle database te verplaatsen.

- Voor het bewerken van XML gegevens bestaan Application Programming Interfaces (API's) voor PL/SQL, Java, C en C++.

Oracle XMLDB is niet een afzonderlijke server, maar de naam van een groep technologieën die te maken hebben met het opslaan en ophalen van XML gegevens in en uit de database. In Oracle 9*i* was dit een afzonderlijk te installeren optie. Vanaf Oracle 10*g* maakt XMLDB standaard deel uit van de database.

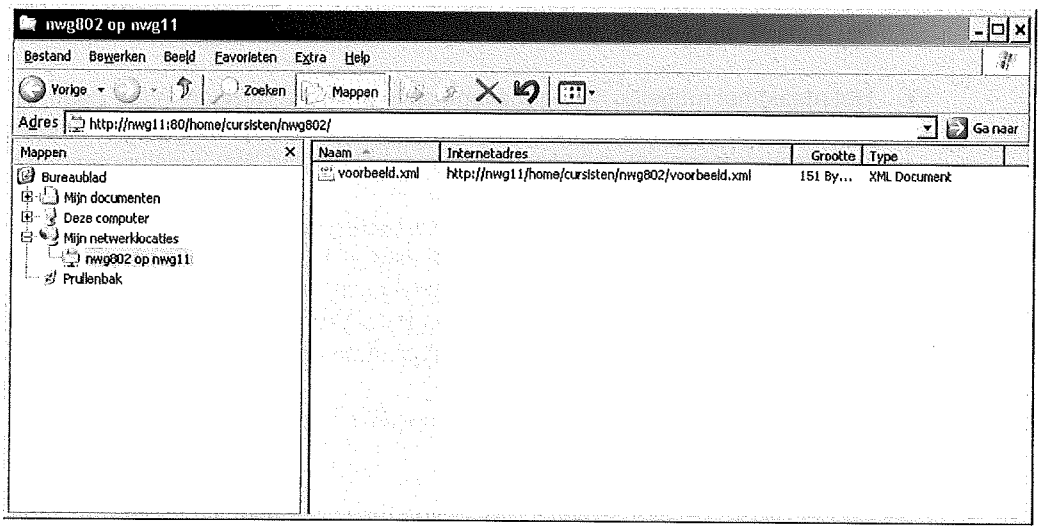
2.2.1 De XMLDB repository

De XMLDB repository is een verzameling database objecten, die alle XML en database schema's overkoepelt. Deze database objecten zijn gekoppeld aan padnamen die in een boomstructuur zijn geordend, vergelijkbaar met een Unix bestandssysteem.
De volgende lijst beschrijft enige termen die in verband met de XMLDB repository worden gebruikt.

- **Bron:** een object of knoop in de hiërarchie. Bronnen worden benaderd via een URL.
- **Folder**: een knoop of directory in de hiërarchie die een verzameling bronnen bevat. Een folder is zelf ook een bron.
- **Padnaam**: een hiërarchische naam bestaande uit een root element (de eerste /), scheidingstekens tussen elementen (/), en diverse subelementen, die pad elementen worden genoemd. Het volgende pad verwijst bijvoorbeeld naar een bron met de naam voorbeeld.xml: `/home/cursisten/nwg801/voorbeeld.xml`. De pad elementen `home`, `cursisten` en `nwg801` zijn folders.
- **Bron- of linknaam**: de naam van een bron binnen een folder. Bronnamen dienen uniek te zijn binnen een folder.
- **Inhoud**: de XMLType gegevens binnen een bron.
- **Access Control List (ACL)**: een XMLType object binnen de repository, welke gebruikt wordt om de toegang tot bronnen binnen het XMLDB bestandssysteem te regelen.

2.2.2 WebDAV

Met behulp van de standaard protocollen FTP, HTTP, en WebDAV, kunnen applicaties de database benaderen alsof het een bestandsysteem is. Verderop in dit naslagwerk zullen we hier gebruik van maken. Als voorbeeld wordt hier een screenshot getoond van een WebDAV-connectie. WebDAV, een uitbreiding op het HTTP protocol, staat voor "Web-based Distributed Authoring and Versioning". WebDAV maakt het onder meer mogelijk om internetlocaties te benaderen alsof ze deel uitmaken van het bestandssysteem.



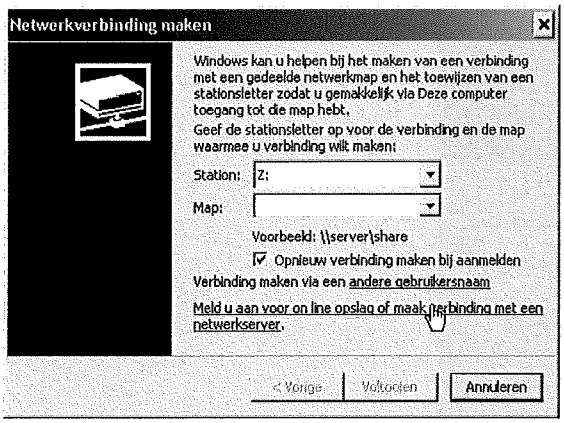
Te zien is dat in de repository de map /home/cursisten/nwg802 is aangemaakt, met daarin het bestand voorbeeld.xml. Dit bestand is als XMLType tabel in de repository opgeslagen, maar wordt via WebDAV als bestand weergegeven.

Folders binnen XMLDB worden met behulp van de package DBMS_XDB aangemaakt in de repository:

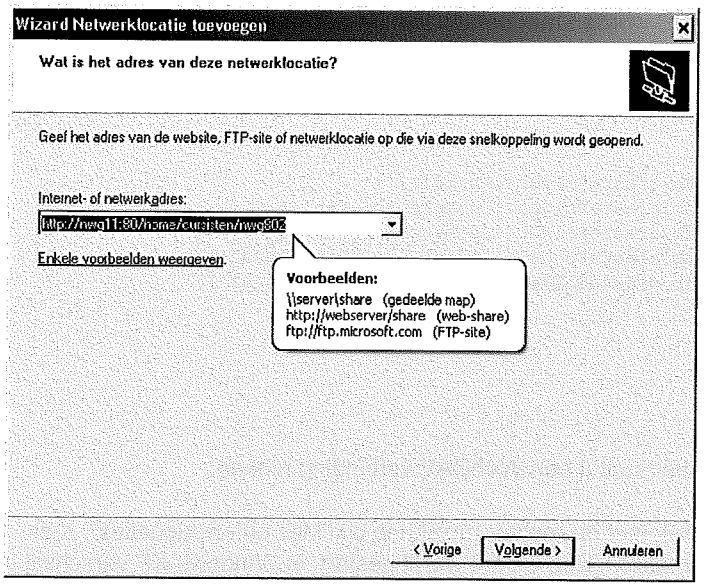
```
declare
  result boolean;
begin
  result := dbms_xdb.createFolder('/home');
  result := dbms_xdb.createFolder('/home/cursisten');
  result := dbms_xdb.createFolder('/home/cursisten/nwg801');
  result := dbms_xdb.createFolder('/home/cursisten/nwg802');
  result := dbms_xdb.createFolder('/home/cursisten/nwg803');
  -- etcetera
end;
/

commit;
```

Op de volgende manier kunt u een WebDAV verbinding maken.

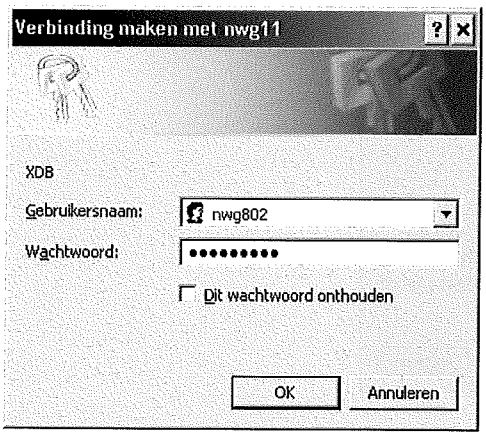
Open verkenner en klik met de rechter muisknop op "Mijn netwerklocaties". Kies "Netwerkverbinding maken". Een wizard wordt gestart. In Windows XP verschijnt dan het volgende scherm.



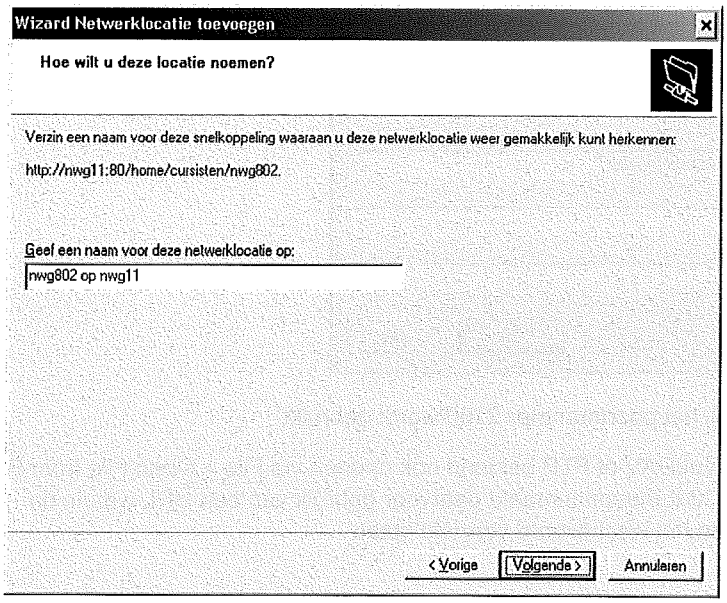
Kies een vrij station, kies eventueel "Opnieuw verbinding maken bij aanmelden", en klik dan op de link: "Meld u aan voor on line opslag of maak verbinding met een netwerkserver". Klik op volgende tot de locatie kan worden opgegeven.



Geef als netwerklocatie http://<servernaam>:80/<pad>/<gebruikersnaam>. Klik op Volgende.



Nu wordt om een gebruikersnaam en wachtwoord gevraagd. Dit zijn uw gebruikersnaam en wachtwoord voor de database. Vul deze in en klik op OK.



Geef een naam voor deze netwerklocatie en klik op Voltooien.

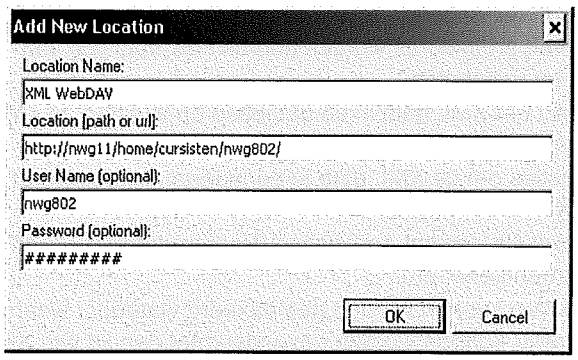
Nu wordt nogmaals om de gebruikersnaam en het wachtwoord gevraagd, waarna de netwerkverbinding wordt geopend.

Binnen een WebDAV folder kunnen we vanuit Explorer verder namen van bestanden wijzigen, bestanden kopiëren, bestanden verwijderen en nieuwe folders aanmaken. Gebruik daarvoor het menu of het popup-menu onder de rechter muisknop.

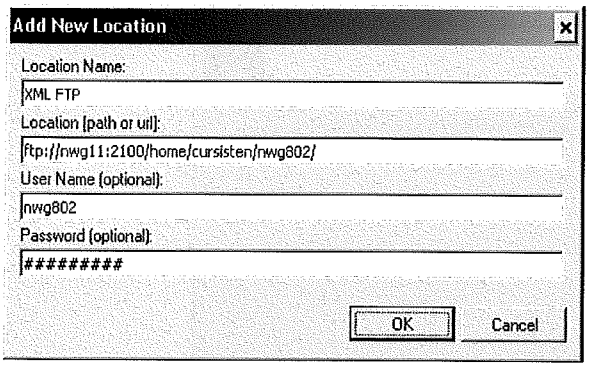
2.2.3 XMLDB bestanden bewerken

We kunnen een bron in de repository ook met behulp van WebDAV of FTP wijzigen. Hiervoor is een editor nodig waarmee we bestanden op een FTP of WebDAV server kunnen benaderen. Tijdens de cursus maken we gebruik van XMLBlueprint. Dit is een XML editor waarmee XML bestanden kunnen worden bewerkt en gevalideerd.
Start XMLBlueprint. Links zien we een lijst mappen met voorbeeldbestanden staan, die bij XMLBlueprint wordt meegeleverd. We gaan nu zelf drie nieuwe locaties toevoegen. Als eerste maken we een WebDAV locatie.

Klik op Add Location... en vul de gegevens op de volgende manier in:



Op een vergelijkbare manier kunnen we XMLDB benaderen via een FTP connectie. Bijvoorbeeld:



Merk op dat hierbij het poortnummer 2100 wordt gebruikt.

We kunnen een WebDAV of FTP bestand ook openen via File > Open File from FTP or WebDAV Server. XMLBlueprint maakt daarvoor gebruik van een URL waarin de gebruikersnaam en het wachtwoord vermeld staat.

2.2.4 XDBUriType

Binnen XMLDB kan gebruik worden gemaakt van URI-typen om via paden te verwijzen naar gegevens in de database. Het type XDBUriType kan worden gebruikt om met behulp van SQL gegevens uit de XMLDB repository op te halen. Dit type heeft onder meer de member functie getXml() waarmee de XML-inhoud van het bestand wordt opgehaald.

Bijvoorbeeld:

```
select xdburitype('/home/cursisten/<gebruikersnaam>/voorbeeld.xml').getXml()
from dual;

XDBURITYPE('/HOME/CURSISTEN/NWG802/VOORBEELD.XML').GETXML()
------------------------------------------------------------
<voorbeelden>
  <voorbeeld nr="1">
    Dit is een voorbeeld.
  </voorbeeld>
  <voorbeeld nr="2">
    En dit is een gewijzigd voorbeeld.
  </voorbeeld>
</voorbeelden>
```

2.3 XMLType

Het object datatype XMLType heeft de volgende eigenschappen:

- XMLType kan in PL/SQL stored procedures worden gebruikt als datatype van parameters, return waarden en variabelen.
- XMLType kan worden gebruikt om XML data in de database op te slaan, of om XML data uit de database op te halen.
- XMLType bevat member functies om de XML data te benaderen. Hierbij wordt onder meer gebruik gemaakt van XPath expressies.
- XMLType waarden kunnen ook via een set Application Program Interfaces (API's) in PL/SQL, Java, C, of C++ worden benaderd.
- XMLType kolommen en -tabellen kunnen worden geïndexeerd waardoor waarden binnen de XML-gegevens sneller kunnen worden gevonden met behulp van XPath expressies.

2.3.1 XMLType opslag opties

XMLType gegevens van een tabel of kolom kunnen in LOBs of object-relationeel worden opgeslagen. Bij het aanmaken van de tabel of kolom moet deze keuze worden gemaakt. De volgende paragrafen geven de verschillen aan tussen de opslag opties.

2.3.1.1 Ongestructureerde opslag

XMLType gegevens worden bij ongestructureerde opslag als CLOB opgeslagen. Dit lijkt op het opslaan in bestanden: de oorspronkelijke structuur van het XML document, inclusief whitespace (spaties, tabs en dergelijke) blijft exact behouden. Dit wordt ook wel *CLOB-opslag* genoemd, of *text-based persistance*.
Standaard worden XMLType kolommen die niet op een XML schema zijn gebaseerd (zie verderop) als CLOB opgeslagen.
De XMLType kolom wordt dan een virtuele kolom over de afgeschermde CLOB kolom. Het is niet mogelijk de CLOB kolom te benaderen. Het is echter wel mogelijk om de CLOB-karakteristieken van de kolom vast te leggen met behulp van de XMLType storage clausule.

2.3.1.2 Gestructureerde opslag (in tabellen en views)

We kunnen ook XMLType kolommen en tabellen maken op basis van een XML schema. Dit geeft onder meer de mogelijkheid de XML gegevens gestructureerd op te slaan in relationele tabellen. We noemen dit **object-relationele opslag** of **object-based persistance**.

Het Document Object Model (DOM) is een abstract model waarin een XML Document wordt beschouwd als een boomstructuur van aan elkaar gerelateerde knopen (nodes). Bij gestructureerde opslag van XMLType gegevens blijven de DOM-kenmerken van de gegevens behouden. De XML documenten worden opgedeeld in object-relationele tabellen of views. Kenmerken die niet van invloed zijn op de DOM-structuur (zoals extra whitespace) gaan hierbij verloren. Andere kenmerken blijven behouden, zoals:

- de volgorde van child elementen en attributen
- onderscheid tussen elementen en attributen
- mixed content (elementen met zowel text nodes als andere elementen)
- commentaar nodes

Veel van deze extra informatie past niet in het SQL object model. XMLType instanties bevatten daarom verborgen kolommen waarin deze extra informatie bewaard wordt. Deze informatie is beschikbaar via API's in SQL of Java, met behulp van XMLType member functies.

Gestructureerde opslag is alleen mogelijk voor tabellen die gebaseerd zijn op een XML schema. Het XML schema dient daarbij van tevoren bekend te zijn. We komen hierop terug in hoofdstuk 5.

2.3.1.3 Binary XML opslag

Vanaf Oracle 11g release 1 bestaat een derde manier om XMLType gegevens op te slaan, die de voordelen van CLOB en gestructureerde opslag combineert. XMLType gegevens worden in dit geval binair opgeslagen, waarbij de XML gegevens reeds geparsed zijn: het DOM model wordt opgeslagen. Deze manier van opslag wordt daarom ook wel **post-parse persistance** genoemd.
Binary XML is verder compact, en kan gebruikt worden voor data die gekoppeld zijn aan één of meer XML schema's, waarbij niet van tevoren bekend hoeft te zijn aan welk schema het document gekoppeld is.
Intern maakt binary XML opslag gebruik van één van de LOB-versies die sinds Oracle 11g gebruikt kunnen worden. De originele vorm staat nu bekend als BasicFile storage, dit is de default vorm. Nieuw in Oracle 11g is SecureFile storage. Het voert hier te ver om diep in te gaan op SecureFile storage, maar XMLType tabellen en kolommen die op deze manier zijn opgeslagen, hebben extra mogelijkheden in het gebruik, zoals:

- compressie van data
- encryptie van data
- voorkomen van herhaling van zelfde data (deduplicatie)
- gedeeltelijke updates (bij aanpassing van een deel van een XML document hoeft niet het hele document te worden vervangen)

2.3.1.4 Overwegingen voor opslag opties

Welke opslagoptie in een bepaalde situatie het meest geschikt is, hangt af van de toepassing. In het gebruik van XML gegevens onderscheiden we twee uitersten: data-georiënteerd of document-georiënteerd.

Bij *data-georiënteerd* gebruik van XML zijn de gegevens over het algemeen zeer gestructureerd. De structuur is statisch en maakt gebruik van een XML schema. Applicaties maken gebruik van deze structuur. Een typisch voorbeeld is een news-feed van een nieuwsaanbieder.
XMLType gegevens aan deze kant van het spectrum kunnen het best *object-relationeel* worden opgeslagen.

Bij *document-georiënteerd* gebruik is er meestal sprake van weinig structuur, waarbij de structuur vaak ook verandert in de loop van de tijd (evolutie). Indien de gegevens wel gestructureerd zijn, maken applicaties daar weinig gebruik van. De gegevens worden dus gebruikt alsof ze niet gestructureerd zijn. Elementen hebben vaak "mixed content": zowel tekst als andere sub-elementen binnen binnen één element. Een voorbeeld hiervan is een webdocument, of een hoofdstuk uit een boek.
XMLType gegevens aan deze kant van de schaal kunnen het best gebruik maken van *text-based persistance (CLOB)* of van *post-parse persistance (binary XML)*.

Tussen beide uitersten in zijn ook verschillende hybride vormen mogelijk. Een relatief ongestructureerd artikel kan bijvoorbeeld gestructureerde metadata bevatten (zoals de auteur, de titel en de datum). Andersom kan een werknemers-record (gestructureerd) ook extra beschrijvende informatie bevatten, zoals een curriculum vitae (ongestructureerd).
XMLType gegevens kunnen ook zo'n hybride vorm aannemen. Zo kan de basis *object-relationeel* worden opgeslagen, waarbij een deel als *CLOB* wordt opgeslagen.

2.3.1.5 Voor en nadelen van XML opslag opties

De volgende tabel geeft een overzicht van de voor- en nadelen van CLOB, Binary XML of gestructureerde opslag van XMLType gegevens:

Kenmerk	Gestructureerde opslag	CLOB opslag	Binary XML opslag
Verwerkingssnelheid	- Verwerken van gegevens gaat gepaard met samenvoegen van gegevens bij queries, en met opbreken van gegevens bij inserts. XML gegevens worden in diverse relationele tabellen opgeslagen.	+ Hoge snelheid voor het verwerken van gehele XML documenten	++ Hoge verwerkingssnelheid. Snel laden van DOM gegevens.
Ruimtegebruik	++ Zeer efficiënt	- Gebruikt de meeste schijfruimte: ook onbelangrijke whitespace en herhaalde tags worden bewaard.	+ Efficiënt
Data flexibiliteit	- Beperkte flexibiliteit. Alleen documenten die voldoen aan een XML schema kunnen in een XMLType tabel of kolom worden opgeslagen.	+ Flexibel. De structuur van een XML document kan eenvoudig worden veranderd.	+ Flexibel. De structuur van een XML document kan eenvoudig worden veranderd.
XML schema flexibiliteit	- Beperkte flexibiliteit. Data en metadata worden apart opgeslagen. Geen mogelijkheid om meerdere XML schema's voor dezelfde XMLType tabel te gebruiken.	+ Flexibel. Data en metadata zijn samen opgeslagen. Geen mogelijkheid om meerdere XML schema's voor dezelfde XMLType tabel te gebruiken.	++ Flexibel. Data en metadata kunnen apart of samen worden opgeslagen. Een XMLtype tabel kan aan meerdere XML schema's worden gekoppeld.
Nauwkeurigheid data opslag	- Extra lege regels en white space binnen tags worden verwijderd. De DOM kenmerken blijven behouden.	+ De originele XML wordt byte voor byte exact bewaard	- Extra lege regels en white space binnen tags worden verwijderd. De DOM kenmerken blijven behouden.
Update operaties (DML)	++ Alleen het deel van de XML-boom dat wordt gewijzigd, doet mee in de update.	- Als een deel van het document wordt gewijzigd, moet het hele document vervolgens opnieuw naar de harde schijf worden teruggeschreven.	+ Bij gebruik van SecureFile LOB storage is het mogelijk gedeeltelijke updates toe te passen.

Kenmerk	Gestructureerde opslag	CLOB opslag	Binary XML opslag
XPath-gebaseerde queries	++ XPath expressies kunnen vaak worden vertaald naar gebruik van de onderliggende object-relationele kolommen (XPath rewrite). Dit kan significante performance winst opleveren.	- XPath expressies worden geëvalueerd door een DOM-boom uit de CLOB data op te bouwen, hetgeen ten koste gaat van de performance. Gebruik van XMLIndex kan de performance verbeteren.	+ DOM constructie is niet nodig, meerdere XPath expressies kunnen tegelijk worden geëvalueerd, door één keer de data te doorlopen. XPath navigatie kan veel sneller dan bij CLOB storage. Gebruik van XMLIndex kan de performance verbeteren.
SQL constraints	+ SQL constraints worden ondersteund.	- SQL constraints worden niet ondersteund	+ SQL constraints worden ondersteund.
SQL scalaire datatypes (deze bevatten een enkele waarde, zonder interne componenten)	+ De 47 scalaire XML schema datatypen worden automatisch vertaald naar de 19 scalaire SQL datatypen.	- SQL scalaire datatypen worden niet ondersteund.	+ De SQL scalaire datatypen worden ondersteund.
Index ondersteuning	B-tree, Oracle Text en function based indexen.	XMLIndex, function-based en Oracle Text indexen.	XMLIndex, function-based en Oracle Text indexen
Geheugen-management	+ XML operaties kunnen worden geoptimaliseerd, waardoor minder beslag op het geheugen wordt gelegd.	- XML operaties op het document vereisen het opbouwen van een DOM-boom uit de CLOB gegevens.	+ XML operaties kunnen worden geoptimaliseerd, waardoor minder beslag op het geheugen wordt gelegd.
Validatie bij inserts	XML gegevens worden gedeeltelijk gevalideerd bij inserts.	XML gegevens die aan een schema zijn gekoppeld worden gedeeltelijk gevalideerd bij inserts.	+ XML gegevens die aan één of meer schema's zijn gekoppeld, worden geheel gevalideerd bij inserts: inserts falen indien de XML gegevens invalid zjin.

2.4 Nieuwe en verouderde features sinds Oracle 11g

XMLDB voldoet sinds Oracle 11g meer aan de SQL/XML standaard. Er zijn verschillende functies bijgekomen, en andere zijn verouderd. Het volgende overzicht is niet uitputtend, maar geeft wel een goed beeld van wat nieuw en wat verouderd is.

2.4.1 Nieuw sinds Oracle 11g release 1

Binary XML opslag	Het nieuwe opslag model voor XMLType, waarin de bestaande opslag modellen van gestructureerde (object-relationele) en ongestructureerde (CLOB) opslag zijn gecombineerd.
XMLIndex	Fragmenten van XML gegevens die met XPath expressies worden benaderd, kunnen met de nieuwe XMLIndex sneller worden gevonden.
Repository events	Applicaties kunnen nu listeners registreren voor events in de XMLDB repository. Deze events zijn vergelijkbaar met database triggers, en gaan bijvoorbeeld af als er een nieuwe bron in de repository wordt aangemaakt.
Web Services	Het is nu mogelijk om XMLDB te benaderen via Web services, met behulp van SQL of XQuery, of gebruik makend van PL/SQL procedures en functies.
Ondersteuning voor XLink en XInclude	XMLDB ondersteunt nu het gebruik van XLink en XInclude, waarmee XML gegevens van verschillende bronnen gecombineerd kunnen worden.
Grote XML nodes	In vorige versies had een XML node een maximum grootte van 64 Kb. Met de komst van nieuwe streaming "push en pull" API's in PL/SQL, Java en C geldt die beperking niet meer.
Import en export	XMLtype gegevens kunnen nu met behulp van Oracle data pump worden geëxporteerd en geïmporteerd.
XQuery standaard	XMLDB ondersteunt nu de laatste versie van de XQuery standaard, de W3C XQuery 1.0 Recommendation.
SQL/XML standaard	De XMLDB ondersteuning voor de SQL/XML standaard is up to date met de laatste versie van de standaard. Dit betreft onder meer de functies XMLExists() en XMLCast().

2.4.2 Nieuw sinds Oracle 11g release 2

Partitioneren	XML-gegevens kunnen nu bij het aanmaken van de tabel, of bij het aanmaken van het schema, gepartitioneerd worden opgeslagen. Dit gebeurt op vergelijkbare manier als het partitioneren van een tabel.
Performance	De performance van lees- en schijfacties in de XMLDB repository is verbeterd.
XMLIndex	XMLIndex kan nu zowel voor gestructureerde als voor ongestructureerde content worden gebruikt.

2.4.3 Verouderd sinds Oracle 11g release 2

De volgende constructies worden vanaf Oracle 11g release 2 gezien als verouderd. Ze werken nog wel, maar het wordt afgeraden om ze in nieuwe applicaties te gebruiken:

Verouderd	Nieuw alternatief
extract()	XMLQuery()
extractvalue()	XMLTable() of een combinatie van XMLQuery() en XMLCast()
existsnode()	XMLExists()
XMLSequence()	XMLTable()
function based indexen op XMLtype gegevens	XMLIndex()

3 XML genereren uit de database

3.1 Inleiding

Een logische stap in het werken met XML in combinatie met de database is het genereren van XML-gegevens uit relationele tabellen. Daartoe bestaan standaard SQL/XML functies, aangevuld met Oracle-specifieke functies. Deze functies maken het ook mogelijk om XMLType views aan te maken op relationele tabellen.

3.2 createXML() en de XMLType constructor

Om nieuwe instanties van het type XMLType aan te maken, kunnen we gebruik maken van XMLType zelf. Dan hebben we de keuze uit XMLType constructors en createXML() functies. Beide zijn door elkaar te gebruiken, en kunnen de XML gegevens onder meer als VARCHAR2, CLOB, object of ref cursor aan de constructor of de functie doorgeven. Voorbeeld met createXML() en een VARCHAR2 waarde:

```
select XMLType.createXML('<test nr="1">Dit is een test</test>')
from dual;

XMLTYPE.CREATEXML('<TESTNR="1">DITISEENTEST</TEST>')
--------------------------------------------------------------------------------
<test nr="1">Dit is een test</test>
```

Ander voorbeeld:

```
select xmltype.createxml('<naam>'||naam||'</naam>')
from ox_werknemers;

XMLTYPE.CREATEXML('<NAAM>'||NAAM||'</NAAM>')
--------------------------------------------------------------------------------
<naam>SMITS</naam>
<naam>ALKEMA</naam>
<naam>WALSTRA</naam>
<naam>PIETERS</naam>
<naam>VERGEER</naam>
<naam>KLAASEN</naam>
<naam>HEUVEL</naam>
<naam>SANDERS</naam>
<naam>KRAAY</naam>
<naam>DROST</naam>
<naam>ADELAAR</naam>
<naam>APPEL</naam>
<naam>VERMEULEN</naam>
<naam>MANDERS</naam>

14 rows selected.
```

Het nadeel van dit gebruik van createXML() is dat we de tags zelf precies moeten formuleren, anders gaat het mis. We kunnen daarom beter de functies gebruiken die in de volgende paragrafen besproken worden. In veel van deze functies wordt alleen de naam van een element meegegeven: begin- en eindtag worden automatisch correct gegenereerd. Bovendien maken deze functies deel uit van de SQL/XML standaard.

Toch hebben de constructor XMLType() en de functie createXML() enkele krachtige mogelijkheden. We kunnen bijvoorbeeld een cursor expressie als invoer gebruiken:

```
select xmltype(cursor(select * from ox_kantoren))
from dual;

XMLTYPE(CURSOR(SELECT*FROMKANTOREN))
--------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
  <ROW>
    <KANTNR>10</KANTNR>
    <NAAM>BOEKHOUDING</NAAM>
    <PLAATS>AMSTERDAM</PLAATS>
  </ROW>
  <ROW>
    <KANTNR>20</KANTNR>
    <NAAM>ONDERZOEK</NAAM>
    <PLAATS>UTRECHT</PLAATS>
  </ROW>
  <ROW>
    <KANTNR>30</KANTNR>
    <NAAM>VERKOOP</NAAM>
    <PLAATS>DEN HAAG</PLAATS>
  </ROW>
  <ROW>
    <KANTNR>40</KANTNR>
    <NAAM>PRODUCTIE</NAAM>
    <PLAATS>ARNHEM</PLAATS>
  </ROW>
</ROWSET>
```

Te zien is dat hierbij automatisch ROWSET en ROW elementen worden gegenereerd. De overige elementnamen zijn gebaseerd op de geselecteerde kolomnamen in de cursor.

We kunnen ook van geneste cursors XMLType gegevens genereren. Bijvoorbeeld:

```
select xmltype(
  cursor(select naam, plaats,
                cursor(select naam, functie, sal from ox_werknemers
                       where kantnr=k.kantnr
                       and functie='MANAGER') MANAGER
         from ox_kantoren k)) KANTOOR
from dual;

KANTOOR
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
  <ROW>
    <NAAM>BOEKHOUDING</NAAM>
    <PLAATS>AMSTERDAM</PLAATS>
    <MANAGER>
      <MANAGER_ROW>
        <NAAM>HEUVEL</NAAM>
        <FUNCTIE>MANAGER</FUNCTIE>
        <SAL>3450</SAL>
      </MANAGER_ROW>
    </MANAGER>
  </ROW>
```

```
  <ROW>
    <NAAM>ONDERZOEK</NAAM>
    <PLAATS>UTRECHT</PLAATS>
    <MANAGER>
      <MANAGER_ROW>
        <NAAM>PIETERS</NAAM>
        <FUNCTIE>MANAGER</FUNCTIE>
        <SAL>3975</SAL>
      </MANAGER_ROW>
    </MANAGER>
  </ROW>
  <ROW>
    <NAAM>VERKOOP</NAAM>
    <PLAATS>DEN HAAG</PLAATS>
    <MANAGER>
      <MANAGER_ROW>
        <NAAM>KLAASEN</NAAM>
        <FUNCTIE>MANAGER</FUNCTIE>
        <SAL>3850</SAL>
      </MANAGER_ROW>
    </MANAGER>
  </ROW>
  <ROW>
    <NAAM>PRODUCTIE</NAAM>
    <PLAATS>ARNHEM</PLAATS>
    <MANAGER/>
  </ROW>
</ROWSET>
```

Hierbij wordt de alias van de geneste cursor gebruikt als naam van de geneste ROWSET, en dezelfde naam met als achtervoegsel "_ROW" als genest ROW-element.

Om gegevens als XML-attributen te genereren kunnen we een objecttype aanmaken, en deze gebruiken in de XMLType constructor of in createXML. In het volgende voorbeeld worden, door het gebruik van "@" in de naamgeving, de kolommen persnr, kantnr en mgr als attribuut gegenereerd:

```
create or replace type "werknemer" as object (
  "@persnr" number, "@mgr" number, "@kantnr" number,
  "naam" varchar2(10), "functie" varchar2(9), "sal" number, "toeslag" number );

type "werknemer" Compiled.
```

In het volgende voorbeeld wordt de default constructor van het zojuist aangemaakte type gebruikt. Merk op dat persnr, mgr en kantnr als attribuut worden gegenereerd. Bovendien worden alle element- en attribuutnamen in kleine letters getoond, doordat de kolomnamen van dit type in kleine letters tussen dubbele aanhalingstekens zijn aangemaakt.

```
select xmltype("werknemer"(
persnr, mgr, kantnr, naam, functie, sal, toeslag)) werknemers
from ox_werknemers
where naam in ('SMITS', 'VERMEULEN');

WERKNEMERS
------------------------------------------------
<werknemer persnr="3381" mgr="7902" kantnr="20">
  <naam>SMITS</naam>
  <functie>KLERK</functie>
  <sal>2400</sal>
</werknemer>
<werknemer persnr="7902" mgr="3930" kantnr="20">
  <naam>VERMEULEN</naam>
  <functie>ANALIST</functie>
  <sal>3900</sal>
</werknemer>
```

De uitvoer is hier verfraaid om het makkelijker te kunnen lezen; de werkelijke uitvoer is niet ingesprongen. Volgens de SQL/XML standaard wordt namelijk nooit extra "whitespace" toegevoegd in de gegenereerde XML. In voorgaande versies kon de extract() functie worden gebruikt om het resultaat te verfraaien ("pretty printing"). Dit werd echter afgeraden, omdat dan een volledige DOM-representatie moet worden opgebouwd van elke rij die wordt opgehaald. Indien pretty-printing toch gewenst is, kan gebruik worden gemaakt van de SQL/XML functie XMLSerialize, die verderop in dit hoofdstuk wordt besproken.
In het vervolg zullen we een pretty print versie laten zien van de uitvoer van selecties, ook waar deze in werkelijkheid niet ingesprongen is.

3.3 SQL/XML functies

Om XML te genereren uit relationele gegevens bestaat een aantal standaard SQL/XML functies, ook wel **XML publishing** functies genoemd. XML publishing is een ISO/IEC standaard voor het genereren van XML-gegevens. We concentreren ons hier op functies die XML genereren uit de database. Er zijn meer SQL/XML functies die bewerkingen uitvoeren op XML gegevens. Deze zullen in volgende hoofdstukken behandeld worden.

In de volgende paragrafen zullen we uit de relationele tabellen werknemers en kantoren XML-gegevens genereren.

3.3.1 XMLElement()

De XMLElement() functie heeft als argumenten de naam van een element, een optionele collectie attributen van het element, en nul of meer argumenten die de inhoud van het element uitmaken. De functie retourneert het element, eventueel met attributen, als een instantie van het type XMLType. Het belangrijkste deel van de syntax is als volgt:

```
XMLELEMENT ( [ [ NAME ] naam | EVALNAME expressie ]
             [ , XML_attributes_clausule ]
             [ , waarde_expr [ AS alias ] [, waarde_expr ]... ]
           )
```

De naam van het element wordt de *identifier* genoemd. Deze wordt tussen dubbele quotes meegegeven. De waarde expressie wordt gebruikt om de inhoud van het XML-element te bepalen. Eventueel kan de naam van het element ook via een string expressie worden opgebouwd, met EVALNAME expressie. Om child-elementen aan te maken kunnen we binnen zo'n expressie de functie XMLElement() opnieuw aanroepen. Daarvoor bestaan overigens ook andere functies. Deze zullen we in de volgende paragrafen behandelen.

Het belangrijkste deel van de syntax van XMLATTRIBUTES is als volgt:

```
→ XMLATTRIBUTES
  (waarde_expr [ AS alias ]
    [, waarde_expr [ AS alias ]...
  )
```

Hiermee worden één of meer attributen aan het element toegevoegd.

Voorbeeld:

```
select xmlelement("verkoper",
                    xmlattributes(persnr as "persnr", mgr as "mgr" ) ,
                    xmlelement("naam", naam),
                    xmlelement("salaris", sal+toeslag)
                  ) as verkopers
from ox_werknemers
where functie='VERKOPER';


VERKOPERS
----------------------------------
<verkoper persnr="3462" mgr="4621">
  <naam>ALKEMA</naam>
  <salaris>2900</salaris>
</verkoper>

<verkoper persnr="3518" mgr="4621">
  <naam>WALSTRA</naam>
  <salaris>2750</salaris>
</verkoper>

<verkoper persnr="4510" mgr="4621">
  <naam>VERGEER</naam>
  <salaris>3650</salaris>
</verkoper>

<verkoper persnr="6500" mgr="4621">
  <naam>DROST</naam>
  <salaris>2500</salaris>
</verkoper>
```

Indien de waarde expressie van XMLELEMENT een NULL waarde oplevert, wordt een leeg element geconstrueerd:

```
select xmlelement("werknemer",
                    xmlelement("naam", naam),
                    xmlelement("toeslag",toeslag)
                  ) as resultaat
from ox_werknemers
where sal between 2500 and 3500;


RESULTAAT
----------------------------
<werknemer>
  <naam>DROST</naam>
  <toeslag>0</toeslag>
</werknemer>

<werknemer>
  <naam>ALKEMA</naam>
  <toeslag>300</toeslag>
</werknemer>

<werknemer>
  <naam>HEUVEL</naam>
  <toeslag></toeslag>
</werknemer>
```

Bij de XMLAttributes clausule werkt dat anders. Indien de waarde expressie daar een null waarde oplevert, wordt het hele attribuut niet aangemaakt:

```
select xmlelement("werknemer",
                    xmlattributes(toeslag as "toeslag"),
                    xmlelement("naam", naam)
                  ) as resultaat
from ox_werknemers
where sal between 2500 and 3500;

RESULTAAT
----------------------------
<werknemer toeslag="0">
  <naam>DROST</naam>
</werknemer>

<werknemer toeslag="300">
  <naam>ALKEMA</naam>
</werknemer>

<werknemer>
  <naam>HEUVEL</naam>
</werknemer>
```

3.3.2 XMLForest()

De XMLForest() functie genereert een groep XML elementen uit een opgegeven lijst argumenten. De syntax is als volgt:

```
XMLFOREST( waarde_expr [ AS { alias | EVALNAME expressie } ]
       [ , waarde_expr [ AS { alias | EVALNAME expressie } ]... ]
         )
```

De lijst waarde-expressies wordt geconverteerd naar XML formaat. Indien de AS alias clausule niet wordt meegegeven, dan wordt de kolomnaam gebruikt als de naam van het gegenereerde element. Eventueel kan met EVALNAME een alias gegenereerd worden uit een expressie.

Voor XMLForest() geldt dat van null waarden geen elementen worden gegenereerd:

```
select xmlelement("werknemer",
          xmlforest(naam as "naam",
                    sal as "salaris" ,
                    toeslag as "toeslag")
        ) as werknemers
from ox_werknemers
where sal between 2500 and 3500;

WERKNEMERS
----------------------------------------------
<werknemer>
  <naam>DROST</naam>
  <salaris>2500</salaris>
  <toeslag>0</toeslag>
</werknemer>

<werknemer>
  <naam>ALKEMA</naam>
  <salaris>2600</salaris>
  <toeslag>300</toeslag>
</werknemer>
```

```
<werknemer>
  <naam>HEUVEL</naam>
  <salaris>3450</salaris>
</werknemer>
```

Merk op dat voor werknemer HEUVEL geen element toeslag is aangemaakt.

Het is ook mogelijk om XML te genereren op basis van zelf gedefinieerde typen. We zullen hiervan een voorbeeld laten zien voor de functie XMLForest():

```
create type werkn_type as object ("@persnr" number, "naam" varchar2(100));
/

type werkn_type Compiled.

select xmlforest( werkn_type(persnr, naam) as "werknemer" ) as werknemers
from ox_werknemers
where kantnr = 20;

WERKNEMERS
--------------------------------------------------------------------------------
<werknemer persnr="3381">
  <naam>SMITS</naam>
</werknemer>

[...]
```

Hier is persnr als een attribuut gegenereerd, en naam als een element. In werkn_type werd persnr als "@persnr" gedefinieerd. Deze naamgeving is ontleend aan XPath: hierin wordt naar attributen verwezen door voor de naam van het attribuut het '@'-teken te zetten.

3.3.3 XMLConcat()

De functie XMLConcat() plakt alle argumenten die worden meegegeven aan elkaar om een XML fragment te creëren. De syntax is als volgt:

```
XMLCONCAT(XMLType_instantie [, XMLType_instantie ]...)
```

XMLConcat() kan in twee vormen worden gebruikt.

In de eerste vorm worden één of meer XMLSequence typen meegegeven. XMLSequence is een collectie-type (een VARRAY), waarvan elk element een XMLType instantie is. XMLConcat() retourneert de XMLType instantie die ontstaat als alle elementen uit de VARRAY(s) aan elkaar worden gekoppeld.

In de tweede vorm wordt een willekeurig aantal XMLType instanties direct aan elkaar gekoppeld.

Bijvoorbeeld:

```
select xmlconcat(xmlelement("naam",naam),xmlelement("sal",sal))
from ox_werknemers
where functie='ANALIST';

XMLCONCAT(XMLELEMENT("NAAM",NAAM),XMLELEMENT("SAL",SAL))
--------------------------------------------------------------------------------
<naam>SANDERS</naam>
<sal>4000</sal>

<naam>VERMEULEN</naam>
<sal>3900</sal>
```

3.3.4 XMLAgg()

De functie XMLAgg() zet een varray XML elementen om in een XML fragment.
XMLAgg() heeft de volgende syntax:

```
XMLAGG(XMLType_instantie [ order_by_clause ])
```

De order_by_clausule heeft hierin de volgende syntax:

```
ORDER BY [lijst van: expr [ASC|DESC] [NULLS {FIRST|LAST}]]
```

In de ORDER BY clausule wordt een nummer niet als kolompositie, maar gewoon als nummer geïnterpreteerd.
Deze functie kan worden gebruikt om XMLType instanties over meerdere rijen bij elkaar te voegen. Met behulp van de ORDER BY clausule kunnen deze instanties bij het samenvoegen worden gesorteerd.

XMLAgg() is een groepsfunctie. Zo'n functie produceert één samengevoegd XML resultaat voor elke groep. Als er geen GROUP BY clausule in de query staat, dan wordt een enkel XML resultaat voor alle rijen van de query gegenereerd.

In het volgende voorbeeld wordt een element "analisten", aangemaakt met als inhoud een de groep werknemers met de functie 'ANALIST':

```
select
   xmlelement("analisten",
     xmlagg(
       xmlelement("werknemer",
         xmlforest(naam "naam", sal "salaris")
         )
     )
    ) as analisten
from ox_werknemers
where functie='ANALIST';

ANALISTEN
--------------------------------------------------------
<analisten>
  <werknemer>
    <naam>SANDERS</naam>
    <salaris>4000</salaris>
  </werknemer>
  <werknemer>
    <naam>VERMEULEN</naam>
    <salaris>3900</salaris>
  </werknemer>
</analisten>
```

Het nut van de ORDER BY clausule wordt duidelijk in het volgende voorbeeld. Hierin worden alle functies getoond. Binnen elke functie worden de werknemers op volgorde van naam getoond:

```
select
 xmlelement("functies",
 xmlagg(
     xmlelement("functie", xmlattributes(functie as "naam"),
     xmlagg(
        xmlelement("werknemer",naam)
        order by naam
      )
 ))) functies
from ox_werknemers
group by functie;
```

```
FUNCTIES
-----------------------------------------------------------------
<functies>
  <functie naam="ANALIST">
    <werknemer>SANDERS</werknemer>
    <werknemer>VERMEULEN</werknemer>
  </functie>
  <functie naam="DIRECTEUR">
    <werknemer>KRAAY</werknemer>
  </functie>
  <functie naam="KLERK">
    <werknemer>ADELAAR</werknemer>
    <werknemer>APPEL</werknemer>
    <werknemer>MANDERS</werknemer>
    <werknemer>SMITS</werknemer>
  </functie>
  [...]
</functies>
```

Een logisch alternatief zou zijn:

```
select
<xmlagg query zonder order by>
from ox_werknemers
group by functie
order by functie, naam
```

De functie XMLAgg() wordt echter eerder uitgevoerd dan de ORDER BY clausule van het SELECT statement, dus dat heeft niet het gewenste effect.

De functie XMLAgg() kan ook worden gebruikt om relaties tussen tabellen weer te geven. In het volgende voorbeeld worden de kantoren op volgorde van naam getoond, met binnen elk kantoor de werknemers die er werken, ook op volgorde van naam.

```
SELECT
  XMLELEMENT("kantoren",
    XMLAGG(
      XMLELEMENT("kantoor",
        XMLATTRIBUTES(k.naam AS "naam"),
          (SELECT
            XMLAGG(
              XMLELEMENT("werknemer",
                XMLATTRIBUTES(w.naam AS "naam")
              )
            order by naam
            )
          FROM werknemers w
          WHERE w.kantnr = k.kantnr
          )
      )
    order by naam
    )
  ) kantoren
FROM ox_kantoren k;

KANTOREN
---------------------------------
<kantoren>
  <kantoor naam="BOEKHOUDING">
    <werknemer naam="HEUVEL"/>
    <werknemer naam="KRAAY"/>
    <werknemer naam="MANDERS"/>
  </kantoor>
  [...]
</kantoren>
```

3.3.5 XMLPI()

Met de XMLPI() functie kan een processing instruction worden gegenereerd. Dit is informatie voor de applicatie, die geen deel uitmaakt van de XML-inhoud. Processing instructions beginnen met <? en eindigen met ?>. Zo kunnen we aan een browser kenbaar maken dat een XML document met een XSLT-stylesheet moet worden bewerkt, met de volgende instructie:

```
<?xml-stylesheet href="test.xsl" type="text/xml" ?>
```

De syntax van XMLPI() is als volgt:

```
XMLPI( { [ NAME ] naam | EVALNAME naam_expressie }
       [ , waarde_expressie] )
```

Een naam (de "target") is verplicht, in bovenstaand voorbeeld is dat "xml-stylesheet". De applicatie dient alleen processing instructions te lezen voor targets die de applicatie herkent: andere processing instructions worden genegeerd. Bovenstaand voorbeeld kan als volgt worden gemaakt:

```
SELECT XMLPI(NAME "xml-stylesheet",
             'href="test.xsl" type="text/xml"') AS pi
from dual;

PI
--------------------------------------------------
<?xml-stylesheet href="test.xsl" type="text/xml"?>
```

3.3.6 XMLRoot()

De functie XMLRoot() behoorde ooit tot de XML/SQL standaard, maar is inmiddels verouderd. Oracle blijft er echter gebruik van maken.
Met XMLRoot() genereren we de XML declaratie voorafgaand aan het root element, zoals:

```
<?xml version="1.0" standalone="yes"?>
```

De syntax van XMLRoot() is alsvolgt:

```
XMLROOT( waarde_expressie, VERSION { versie_expressie | NO VALUE }
                    [ , STANDALONE { YES | NO | NO VALUE } ] )
```

Als VERSION de waarde NO VALUE krijgt, dan wordt *version="1.0"* gegenereerd. Bij STANDALONE heeft de waarde NO VALUE tot gevolg dat het standalone deel niet wordt gegenereerd. Overigens kan dit ook gewoon door STANDALONE helemaal weg te laten.

De waarde van standalone wordt door de parser gebruikt. Als de waarde "yes" is, hoeft de parser niet in een externe DTD zoeken naar eventuele entities of default waarden van attributen.

Voorbeeld:

```
select xmlroot(
         xmlelement("root",'wortel')
         , VERSION NO VALUE ) root
from dual

ROOT
----------------------------
<?xml version="1.0"?>
<root>wortel</root>
```

3.3.7 XMLSerialize()

De functie XMLSerialize() maakt een CLOB of VARCHAR2 representatie van XML gegevens. De syntax van XMLSerialize() is als volgt:

```
XMLSERIALIZE( { DOCUMENT | CONTENT } waarde_expressie
              [ AS { CLOB | VARCHAR | VARCHAR2 | BLOB [ENCODING encoding_string] } ]
              [ VERSION versie_string ]
              [ NO INDENT | INDENT [ SIZE = nummer ]
              [ { HIDE | SHOW } DEFAULTS ]
            )
```

De waarde expressie moet hierbij een XMLType instantie opleveren.
Indien DOCUMENT wordt meegegeven, dan dienen de XML gegevens een well-formed document te zijn, met één root element. Als CONTENT wordt meegegeven, dan wordt niet gecontroleerd of de XML gegevens well-formed zijn. Indien BLOB of CLOB als uitvoer wordt gebruikt, wordt bij DOCUMENT standaard ingesprongen, bij CONTENT niet. Dit kan met INDENT of NO INDENT worden gewijzigd.

Als datatype van de uitvoer kunnen we CLOB, VARCHAR, VARCHAR2, of BLOB opgeven, waarbij bij Oracle VARCHAR automatisch wordt omgezet naar VARCHAR2. Het default datatype is CLOB. Indien BLOB wordt opgegeven, kan ook de ENCODING worden gespecificeerd (zoals bijvoorbeeld utf-8).

Indien de XML gegevens zijn gekoppeld aan een XML schema, kan met HIDE DEFAULTS of SHOW DEFAULTS worden aagegeven of default waarden, zoals in het schema gedefinieerd, in de uitvoer getoond moeten worden.

Sinds Oracle11g gebruiken we deze functie om een "pretty-print" versie van XML gegevens te genereren. In het volgende voorbeeld doen we dat voor de analisten:

```
select
  xmlserialize( DOCUMENT
   xmlelement("analisten",
     xmlagg(
       xmlelement("werknemer",
         xmlforest(naam "naam", sal "salaris")
         )
     )
    ) as CLOB
      INDENT SIZE=4)
     as analisten
from ox_werknemers
where functie='ANALIST';

ANALISTEN
------------------------------
<analisten>
    <werknemer>
        <naam>SANDERS</naam>
        <salaris>4000</salaris>
    </werknemer>
    <werknemer>
        <naam>VERMEULEN</naam>
        <salaris>3900</salaris>
    </werknemer>
</analisten>
```

3.3.8 XMLParse()

De XMLParse() functie doet in zekere zin het omgekeerde van de XMLSerialize() functie: hiermee wordt een string omgezet in een XMLType instantie. De syntax is als volgt:

```
XMLPARSE ( { DOCUMENT | CONTENT } waarde_expressie [ WELLFORMED ] )
```

De waarde expressie dient een VARCHAR2 waarde te zijn, die XML gegevens bevat. Deze gegevens dienen well-formed te zijn. Indien DOCUMENT wordt meegegeven, geldt bovendien dat er één root element moet zijn. Bij CONTENT kan het een XML fragment zijn, waarin mogelijk meerdere root elementen voorkomen.
Met WELLFORMED geven we aan dat niet gecontroleerd hoeft te worden of het document wellformed is. Als we daar al zeker van zijn, kan dat performance-winst opleveren.

We hebben eerder gezien dat we met de XMLType constructor, of met de methode createXML() een XMLType instantie kunnen aanmaken. In de meeste gevallen heeft het gebruik van XMLParse() de voorkeur, omdat dit een standaard XML/SQL functie is.

Voorbeeld:

```
select xmlparse(DOCUMENT '<naam>'||naam||'</naam>')
from ox_werknemers;
```

3.3.9 XMLComment()

Met de functie XMLComment() kunnen we commentaarregels genereren. De syntax is als volgt:

```
XMLComment ( waarde_expressie )
```

Bijvoorbeeld (commentaar.sql):

```
select XMLElement("voorbeeld",
          XMLComment('dit is een leeg element met commentaar')
          ) as voorbeeld
from dual;

VOORBEELD
--------------------------------------------------------------------------
<voorbeeld><!--dit is een leeg element met commentaar--></voorbeeld>
```

3.4 Oracle supplied SQL functies

Oracle heeft de standaard SQL/XML functies aangevuld met enige Oracle-specifieke functies. We zullen deze hieronder bespreken.

3.4.1 XMLSequence()

De XMLSequence() functie genereert een varray van XMLType instanties. Omdat deze functie een collectie retourneert, kan XMLSequence() worden gebruikt in de FROM clausule van SQL queries.

De syntax van XMLSequence() is als volgt:

```
XMLSEQUENCE( XMLType_instantie
           | sys_refcursor_instantie [, formaat ]
           )
```

XMLSequence kan in twee vormen worden gebruikt.

De eerste vorm heeft een XMLType instantie als invoer, en retourneert een VARRAY van XML fragmenten. Deze vorm wordt in het volgende hoofdstuk besproken.

De tweede vorm van XMLSequence heeft een REFCURSOR als argument en retourneert een XMLSequenceType instantie, die een varray van XMLType fragmenten bevat. Deze vorm kan worden gebruikt om XML te genereren uit standaard SQL queries.

Standaard genereert XMLSequence() een varray van XML documenten met als root een element met de naam ROW. Met behulp van een XMLFormat object kunnen we daar een andere naam aan geven. Dit object heeft onder meer een methode createFormat() waarmee de nieuwe naam kan worden meegegeven. Bijvoorbeeld:

```
select xmlsequence(
  cursor(
    select * from ox_werknemers where functie='ANALIST'
  )
  , xmlformat.createformat('WERKNEMER')
) as analisten
from dual;

ANALISTEN
------------------------
SYS.XMLTYPE( <WERKNEMER>
  <PERSNR>5931</PERSNR>
  <NAAM>SANDERS</NAAM>
  <FUNCTIE>ANALIST</FUNCTIE>
  <MGR>3930</MGR>
  <SAL>4000</SAL>
  <KANTNR>20</KANTNR>
 </WERKNEMER>
, <WERKNEMER>
  <PERSNR>7902</PERSNR>
  <NAAM>VERMEULEN</NAAM>
  <FUNCTIE>ANALIST</FUNCTIE>
  <MGR>3930</MGR>
  <SAL>3900</SAL>
  <KANTNR>20</KANTNR>
 </WERKNEMER>
)
```

De gegenereerde elementen binnen elk root element krijgen dezelfde naam als de corresponderende kolommen. Indien aliassen worden gebruikt, worden deze namen overgenomen. Het is raadzaam om expressies een alias te geven. Bijvoorbeeld:

```
select xmlsequence(
         cursor(
           select initcap(naam) as naam
           from ox_werknemers
           where functie='DIRECTEUR'
         )
       ) as directeur
from dual;

DIRECTEUR
------------------------------------------------
<ROW>
  <NAAM>Kraay</NAAM>
</ROW>
```

Een XMLSequenceType instantie kan met behulp van de table() functie worden geconverteerd naar een virtuele tabel van losse rijen. Op zo'n virtuele tabel kunnen we queries los laten. Bijvoorbeeld:

```
select *
from table(
        xmlsequence(
         cursor(
           select * from ox_werknemers where functie='VERKOPER'
         )
       ));

COLUMN_VALUE
------------------------------
 <ROW>
  <PERSNR>3462</PERSNR>
  <NAAM>ALKEMA</NAAM>
  <FUNCTIE>VERKOPER</FUNCTIE>
  <MGR>4621</MGR>
  <SAL>2600</SAL>
  <TOESLAG>300</TOESLAG>
  <KANTNR>30</KANTNR>
 </ROW>

[...]
```

De table functie heeft de XMLSequenceType instantie opgebroken in vier losse rijen, met één pseudokolom: column_value. We hadden ook "select column_value from table (...) " kunnen doen.

Een alternatief is de volgende notatie, waarbij de virtuele tabel een alias krijgt, die aan de functie value() wordt meegegeven.

```
select value(v)
from table(
        xmlsequence(
         cursor(
           select * from ox_werknemers where functie='VERKOPER'
         )
       )) v ;
```

In het algemeen geldt dat met value() de instanties opgehaald kunnen worden die in een objecttabel staan opgeslagen.

Sinds Oracle 11g wordt dit gebruik van table() in combinatie met XMLSequence() afgeraden, omdat er een beter XML/SQL alternatief is: de functie XMLTable(). Deze functie maakt gebruik van de XML query taal XQuery. Dit onderwerp komt in een volgend hoofdstuk aan de orde.

3.4.2 XMLColAttVal

De functie XMLColAttVal() genereert een groep XML elementen die de waarden bevatten van de meegegeven parameters. Dit is een Oracle Database uitbreiding op de SQL/XML ANSI-ISO standaard functies.

De XMLColAttVal() functie heeft de volgende syntax:

```
XMLCOLATTVAL(waarde_expressie [ AS alias ]
             [, waarde_expressie [ AS alias ]...
            )
```

Door deze functie worden elementen gegenereerd met de naam "column" en een attribuut "name". Dit name attribuut krijgt de kolomnaam (of alias) als waarde.
Deze functie is vooral handig om kolomnamen of aliassen met spaties te genereren. Bij de vergelijkbare XMLForest functie kunnen alleen underscores worden gebruikt, anders zou ongeldige XML ontstaan. Elementnamen mogen immers geen spaties bevatten.

Bijvoorbeeld:

```
select xmlelement("tabel",
          xmlagg(xmlelement("rij",
          xmlforest(naam "naam_kantoor",
                    plaats "plaats_kantoor")))) kantoren
from ox_kantoren;

KANTOREN
--------------------------------------------------------------------
<tabel>
  <rij>
    <naam_kantoor>BOEKHOUDING</naam_kantoor>
    <plaats_kantoor>AMSTERDAM</plaats_kantoor>
  </rij>
[...]
</tabel>
```

Bij het gebruik van de XMLColAttVal() functie is de gegenereerde naam een waarde van het attribuut "name". Een attribuutwaarde mag wel spaties bevatten:

```
select xmlelement("tabel",
          xmlagg(xmlelement("rij",
          xmlcolattval(naam "naam kantoor",
                       plaats "plaats kantoor")))) kantoren
from ox_kantoren;


KANTOREN
-----------------------------------------------------------------------------
<tabel>
  <rij>
    <column name="naam kantoor">BOEKHOUDING</column>
    <column name="plaats kantoor">AMSTERDAM</column>
  </rij>
  [...]
</tabel>
```

3.4.3 SYS_XMLGEN

De SYS_XMLGen() functie is vergelijkbaar met XMLElement, met het verschil dat SYS_XMLGen() een enkel argument heeft en het resultaat converteert naar XML. Anders dan de andere functies die XML genereren, retourneert SYS_XMLGen() altijd XML documenten, dus elk met één root. Andere functies kunnen ook XML DocumentFragmenten genereren, die meerdere roots kunnen bevatten.

De syntax van SYS_XMLGEN is als volgt:

```
SYS_XMLGEN(expressie [, formaat ])
```

Voorbeeld:

```
select sys_xmlgen(naam)
from ox_werknemers;
```

```
SYS_XMLGEN(NAAM)
------------------------------------------
<NAAM>SMITS</NAAM>
<NAAM>ALKEMA</NAAM>
<NAAM>WALSTRA</NAAM>
<NAAM>PIETERS</NAAM>
[...]
```

Indien geen kolom maar een expressie wordt meegegeven, dan worden er elementen met de naam ROW gegenereerd:

```
select sys_xmlgen(lower(naam))
from ox_kantoren;

SYS_XMLGEN(LOWER(NAAM))
-------------------------------------
<ROW>boekhouding</ROW>
<ROW>onderzoek</ROW>
<ROW>verkoop</ROW>
<ROW>productie</ROW>
```

Om de naamgeving te beïnvloeden kan als tweede argument van SYS_XMLGen() een XMLFormat object worden meegegeven. We gebruiken daarvoor de functie createFormat() van dit objecttype:

```
select sys_xmlgen(lower(naam),
       xmlformat.createformat('kantoor'))
from kantoren;

SYS_XMLGEN(LOWER(NAAM),XMLFORMAT.CREATEFORMAT('KANTOOR'))
--------------------------------------------------------------------------------
<kantoor>boekhouding</kantoor>
<kantoor>onderzoek</kantoor>
<kantoor>verkoop</kantoor>
<kantoor>productie</kantoor>
```

3.4.4 SYS_XMLAGG

De SYS_XMLAGG() functie groepeert XML documenten of fragmenten en produceert een enkel XML document per groep. Om elke groep wordt een omsluitend element met de default naam ROWSET geplaatst. Om deze naam aan te passen kan weer gebruik worden gemaakt van een XMLFormat object.

De syntax lijkt op die van SYS_XMLGEN:

```
SYS_XMLAGG(expressie [, formaat ])
```

Voorbeeld :

```
select sys_xmlagg(
         sys_xmlgen(naam, xmlformat.createformat('kantoor')),
         xmlformat.createformat('kantoren')) as kantoren
from ox_kantoren;

KANTOREN
--------------------------------
<?xml version="1.0"?>
<kantoren>
<kantoor>BOEKHOUDING</kantoor>
<kantoor>ONDERZOEK</kantoor>
<kantoor>VERKOOP</kantoor>
<kantoor>PRODUCTIE</kantoor>
</kantoren>
```

3.5 XMLType views op relationele tabellen

We hebben gezien dat XML documenten direct in XMLType kolommen of -tabellen kunnen worden opgeslagen. Vaak echter zijn de gebruikte gegevens al gewoon in relationele tabellen opgeslagen. Met behulp van voorgaande functies kunnen we XMLType views maken op gegevens in relationele tabellen.

XMLType views lijken op object views. Elke rij van een XMLType view komt overeen met een XMLType instantie. Elke object view heeft ook een *object identifier* die het mogelijk maakt een referentie te maken naar een object in de view. Voor XMLType views maken we gebruik van de SQL/XML functies XMLCast en XMLQuery voor deze object identifiers.

Er zijn grofweg vier typen XMLType views te onderscheiden, op basis van twee tweedelingen:

- gebaseerd op een XML schema, versus
- niet gebaseerd op een XML schema
- aangemaakt met objecttypen en SYS_XMLGen(), versus
- aangemaakt met behulp van SQL/XML publishing functies

We geven nu een voorbeeld van een XMLType view, die niet gebaseerd is op een XML schema. Verderop in de cursus komt XML schema uitgebreid aan de orde.

We maken een view op de tabellen werknemers en kantoren, met behulp van SQL/XML publishing functies. In de view xml_kantoren_vw komt elk kantoor als een XMLType voor. We geven bij het maken van een XMLType view aan welk kenmerk elk gegenereerd XMLType document uniek identificeert. Dit wordt gespecificeerd door de OBJECT ID clausule in het volgende voorbeeld:

```
create view xml_kantoren_vw of xmltype
with object id (XMLCast(
                  XMLQuery('/kantoor/@kantnr'
                           PASSING OBJECT_VALUE RETURNING CONTENT)
               as INTEGER)) AS
SELECT xmlelement
  ("kantoor", xmlattributes(k.kantnr as "kantnr"),
     xmlforest(k.naam "naam", k.plaats "plaats"),
       (select xmlagg(
          xmlelement("werknemer",xmlattributes(w.persnr as "persnr",
                                               w.mgr as "mgr"),
                     xmlforest(w.naam "naam",
                               w.sal "sal",
                               w.toeslag "toeslag",
                               w.functie "functie"
                     )
          )
       )
    from ox_werknemers w
    where w.kantnr=k.kantnr) as "werknemers"
 ) kantoren
from ox_kantoren k;

View created.
```

We zien hierin de eerder behandelde functies XMLAgg(), XMLElement(), XMLAttributes en XMLForest() om de XMLType inhoud te genereren. Met behulp van XMLCast() en XMLQuery() geven we aan dat het attribuut @kantnr elk object in deze XMLType view uniek identificeert. Deze functies worden behandeld in een volgend hoofdstuk, over het bewerken van XMLType gegevens.

Informatie over XMLType views staat in de datadictionary views USER_XML_VIEWS en USER_XML_VIEW_COLS. In het volgende hoofdstuk zullen we de view xml_kantoren_vw met behulp van XMLType functies benaderen.

4 XMLType gegevens bewerken

4.1 Inleiding

Dit hoofdstuk behandelt het aanmaken en bewerken van XMLType gegevens. XMLType is een objecttype, op basis waarvan tabellen of kolommen kunnen worden aangemaakt. We bespreken hoe dergelijke tabellen en kolommen kunnen worden aangemaakt, en hoe deze met gegevens kunnen worden gevuld. Vervolgens behandelen we functies waarmee XMLType gegevens kunnen worden benaderd en bewerkt. Daarvoor gebruiken we functies die los kunnen worden aangeroepen, of als member functie van het datatype XMLType.

4.2 XMLType tabellen en kolommen

In het vorige hoofdstuk hebben we behandeld hoe XMLType gegevens kunnen worden gegenereerd uit de database. In dit hoofdstuk gaan we uit van tabellen, kolommen of variabelen van het type XMLType. De komende paragrafen beschrijven hoe XMLType gegevens kunnen worden benaderd en bewerkt. Hier wordt uitgelegd hoe tabellen en kolommen van het type XMLType kunnen worden gemaakt.

De syntax van het aanmaken van een XMLType tabel is als volgt:

```
create table <tabelnaam> of xmltype [store as {object relational | clob}]
```

In het volgende voorbeeld wordt XMLType als type van de gehele tabel gebruikt.

```
create table mijn_documenten of xmltype;
```

Hierbij worden dus geen kolommen onderscheiden. Met describe kunnen we de structuur van deze tabel opvragen:

```
desc mijn_documenten

Name                          Null?    Type
----------------------------- -------- --------------------
SYS_NC_ROWINFO$                        XMLTYPE()
```

*De naam SYS_NC_ROWINFO$ is eigenlijk verouderd: nu wordt de voorkeur gegeven aan OBJECT_VALUE. In SQL*Plus wordt de structuur van deze tabel weergegeven als Name: TABLE of PUBLIC.XMLTTYPE, waarbij Type leeg blijft.*

In het volgende voorbeeld wordt een tabel aangemaakt, met een XMLType kolom:

```
create table xml_kladblok
( id number primary key
, naam varchar2(50)
, datum date default sysdate
, gegevens xmltype);

create table succeeded.
```

```
desc xml_kladblok

 Name                          Null?    Type
 ----------------------------- -------- --------------------
 ID                            NOT NULL NUMBER
 NAAM                                   VARCHAR2(50)
 DATUM                                  DATE
 GEGEVENS                               XMLTYPE
```

Het is ook mogelijk een schema te registreren met behulp van de DBMS_XMLSCHEMA package. Als het schema is geregistreerd kan een tabel of kolom worden aangemaakt dat conformeert aan dit schema. In dit geval slaat Oracle de XML data op in onderliggende object-relationele kolommen. Queries en DML op object-relationele tabellen zullen over het algemeen sneller worden uitgevoerd, omdat daarvoor een beter execution plan kan worden gevonden. Het gebruik van XML Schema in de database zullen we in hoofdstuk 7 behandelen.

4.2.1 XML-gegevens toevoegen

XMLType gegevens kunnen via INSERT statements, via datapump of met behulp van SQL*Loader in een tabel worden geladen. Hierbij dienen de XML-gegevens correct gestructureerd te zijn ("well-formed"). We bespreken hier eerst het gebruik van het INSERT statement.

Het volgende voorbeeld maakt gebruik van een INSERT ... VALUES statement:

```
insert into xml_kladblok(id, naam, gegevens)
values (1,'Hendrix',
        xmlType.createXML('<a><b>test</b><b><c>test2</c></b></a>'));

1 rows inserted.

 insert into mijn_documenten
 values (xmlType.createXML('<document>
   <naam>testdocument</naam>
   <omschrijving>Dit is een testdocument</omschrijving>
   <paragraaf>
     <kop>kopje1</kop>
     Deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
     elementen als tekst in voorkomt
     </paragraaf>
   <paragraaf>
     <kop>kopje2</kop>
     Ook deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
     elementen als tekst in voorkomt
     <opmerking>Dit is de tweede paragraaf in dit document</opmerking>
     </paragraaf>
 </document>'))
/

1 rows inserted.
```

Omdat de gehele tabel mijn_documenten van het type XMLType is kunnen we voor de inhoud niet verwijzen naar een kolomnaam. Als we de waarde op willen halen, zijn twee notatiewijzen mogelijk. Op de eerste manier wordt gebruik gemaakt van een tabel-alias, waarnaar verwezen wordt in de value() functie:

```
select value(d)
from mijn_documenten d;
```

```
VALUE(D)
--------------------------------------------------------------------------------
<document>
   <naam>testdocument</naam>
   <omschrijving>Dit is een testdocument</omschrijving>
   <paragraaf>
     <kop>kopje1</kop>
     Deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
     elementen als tekst in voorkomt
     </paragraaf>
   <paragraaf>
     <kop>kopje2</kop>
     Ook deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
     elementen als tekst in voorkomt
     <opmerking>Dit is de tweede paragraaf in dit document</opmerking>
     </paragraaf>
 </document>
```

We kunnen naar de waarde van een object_type tabel ook als volgt verwijzen. Deze notatiewijze heeft de voorkeur:

```
select object_value
from mijn_documenten;
```

4.2.2 XML gegevens laden met SQL*Loader

Vaak worden tabellen periodiek gevuld met externe gegevens. Dit geldt des te meer voor XML gegevens, omdat dit formaat bij uitstek geschikt is voor gegevensuitwisseling met externe partijen. Verderop zullen we manieren behandelen om dergelijke gegevens in tabellen te laden, door ze aan de XMLDB repository toe te voegen. In deze paragraaf geven we enkele voorbeelden van het gebruik van SQL*Loader.

SQL*Loader is een client-side tool voor het laden van externe gegevens in tabellen. Deze tool maakt gebruik van een control file, waarin specificaties staan over de manier van laden van de betreffende gegevens. De externe gegevens kunnen in deze control file staan, of in een apart databastand. SQL*Loader gebruikt voor de control file een eigen syntax, die we hier niet zullen behandelen. We verwijzen daarvoor naar de cursus Oracle 11g Database Administration - deel 1B.

In het volgende voorbeeld zullen we gegevens toevoegen aan de tabel mijn_documenten, waarbij de gegevens in de control file staan:

```
LOAD DATA
INFILE *
INTO TABLE mijn_documenten APPEND
XMLType(xmldata)
(
xmldata char(4000)
)
BEGINDATA
<voorbeeld type="inline" nr="1"><beschrijving>Dit XML bestand wordt inline aangeboden
aan SQL*Loader via de control file</beschrijving></voorbeeld>
<voorbeeld type="inline" nr="2"><beschrijving>Ook dit XML bestand wordt inline
aangeboden aan SQL*Loader via de control file</beschrijving></voorbeeld>
```

We geven in dit geval met INFILE * aan dat de gegevens in de control file zelf staan. Met APPEND voegen we de gegevens toe aan de tabel. Alternatieven zijn:

- TRUNCATE (tabel wordt eerst met TRUNCATE leeggemaakt),
- REPLACE (tabel wordt eerst met DELETE leeggemaakt, eventuele DELETE triggers gaan af), en
- INSERT (voegt rijen toe aan een lege tabel. Als de tabel niet leeg is volgt een foutmelding)

Let op: SQL*Loader gaat er default vanuit dat elke nieuwe regel in het bestand ook een nieuw record in de tabel wordt.

Om deze gegevens te laden openen we een DOS box (Start > Run > cmd) en voeren we de volgende commando's uit:

```
H:\>set path=c:\oralg\bin;%path%
H:\>sqlldr <gebruikersnaam>/vh_cursus@nwg11 h:\oraclexml\inline.ctl
log=h:\oraclexml\inline.log

SQL*Loader: Release 11.1.0.6.0 - Production on Thu Oct 8 16:15:44 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
Commit point reached - logical record count 2
```

Controleer hierna de inhoud van de tabel mijn_documenten.

```
select * from mijn_documenten;

SYS_NC_ROWINFO$
--------------------
<document>
  <naam>testdocument</naam>
  <omschrijving>Dit is een testdocument</omschrijving>
  <paragraaf>
    <kop>kopje1</kop>
    Deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
    elementen als tekst in voorkomt
  </paragraaf>
  <paragraaf>
    <kop>kopje2</kop>
    Ook deze paragraaf heeft mixed content, hetgeen betekent dat er zowel
    elementen als tekst in voorkomt
    <opmerking>Dit is de tweede paragraaf in dit document</opmerking>
  </paragraaf>
</document>

<voorbeeld type="inline" nr="1"><beschrijving>Dit XML bestand wordt inline aangeboden
aan SQL*Loader via de control file</beschrijving></voorbeeld>
<voorbeeld type="inline" nr="2"><beschrijving>Ook dit XML bestand wordt inline
aangeboden aan SQL*Loader via de control file</beschrijving></voorbeeld>

3 rows selected
```

In het volgende voorbeeld laden we XML gegevens via een extern databastand:

```
LOAD DATA
INFILE *
INTO TABLE mijn_documenten APPEND
XMLType(xmldata)
FIELDS(fill filler CHAR(1),
xmldata LOBFILE (CONSTANT "h:\oraclexml\extern.dat")
TERMINATED BY '<!-- einde rij -->')
BEGINDATA
0
0
```

Met "fill filler CHAR(1)" geven we aan dat het eerste karakter niet deel uitmaakt van de in te lezen gegevens. Dat gaat in dit geval om de nullen onder BEGINDATA: hiermee geven we aan hoeveel regels we uit het externe bestand moeten inlezen. Het bestand extern.dat wordt als LOBFILE ingelezen: in brokken van 64kb. Het externe bestand hoeft daardoor niet geheel in het geheugen te passen. Bij het inlezen als LOBFILE geldt niet dat elke regel als record wordt gezien. We moeten in dit geval in het bestand aangeven waar een record eindigt. In het data bestand is dat aangegeven met <!-- einde rij -->:

```
<voorbeeld type="extern" nr="3">
  <beschrijving>Dit is een rij uit een extern bestand.</beschrijving>
</voorbeeld>
<!-- einde rij -->
<voorbeeld type="extern" nr="4">
  <beschrijving>Dit is de volgende rij uit hetzelfde externe bestand</beschrijving>
</voorbeeld>
<!-- einde rij -->
```

Met CONSTANT geven we aan dat we voor elke rij hetzelfde bestand gebruiken: extern.dat.

Op de volgende manier kunnen we deze rijen toevoegen:

```
H:\>sqlldr <gebruikersnaam>/vh_cursus@nwg11 h:\oraclexml\extern.ctl
log=h:\oraclexml\extern.log
```

Het laden van losse rijen uit een bestand is misschien wat omslachtig, omdat we ergens mee aan moeten geven waar een rij eindigt. Wat vaker voor zal komen, is dat losse XML bestanden als rijen aan een tabel worden toegevoegd. Dit zien we in het volgende voorbeeld:

```
load data
infile *
replace
into table xml_kladblok APPEND
fields terminated by ","
( id integer external,
  naam char,
 lob_file FILLER char,
 gegevens LOBFILE(lob_file) TERMINATED BY EOF
 )
begindata
5,Inge,h:\oraclexml\test5.xml
6,Arthur,h:\oraclexml\test6.xml
```

Ook hier maken we gebruik van FILLER, om aan te geven dat de bestandsnamen niet ingelezen moeten worden. Met LOBFILE(lob_file) geven we aan dat we de inhoud van die bestanden wel willen inlezen.

4.3 XMLType functies

Het XMLType is een object datatype voor het opslaan en verwerken van XML gegevens. Voor het verwerken van XMLType data zijn diverse member functies beschikbaar. Member functies zijn functies die vanuit het object zelf worden aangeroepen. Enkele XMLType functies bestaan ook als standaard Oracle functie.

4.3.1 XPath expressies

Om gegevens binnen een XMLType te kunnen benaderen zijn binnen Oracle enkele functies beschikbaar die gebruik maken van XPath expressies. Dergelijke expressies worden onder meer binnen XSLT en DOM gebruikt om nodes binnen een XML-boomstructuur te lokaliseren.
Een XPath expressie wordt geëvalueerd en levert een object op van één van de volgende typen:
- numeriek
- string
- boolean
- node-set

XPath bevat onder meer functies, operatoren en locatiepaden. We zullen hier alleen locatiepaden bespreken.
Een locatie pad verwijst vanaf een bepaalde context node naar een andere (of dezelfde) node of node set. Het pad bestaat uit één of meer stappen, gescheiden door een forward slash '/'.

Als voorbeeld bekijken we het volgende XML document. Stel dat de context node het element werknemers is:

```
<kantoor kantnr="10">
  <naam>BOEKHOUDING</naam>
  <plaats>AMSTERDAM</plaats>
  <werknemers>
    <werknemer persnr="5810" mgr="6221">
      <naam>HEUVEL</naam>
      <sal>3450</sal>
      <functie>MANAGER</functie>
    </werknemer>
    <werknemer persnr="6221">
      <naam>KRAAY</naam>
      <sal>6000</sal>
      <functie>DIRECTEUR</functie>
    </werknemer>
    <werknemer persnr="8222" mgr="5810">
      <naam>MANDERS</naam>
      <sal>2300</sal>
      <functie>KLERK</functie>
    </werknemer>
  </werknemers>
</kantoor>
```

Vanaf deze context kunnen we onder meer de volgende nodes benaderen:

uitgeschreven	afgekort	node of nodeset
/	/	root element
parent::node()	..	het omvattende element kantoor
self::node()	.	het element werknemers zelf
child::node()	*	alle elementen werknemer
descendant::naam	.//naam	alle naam elementen ergens binnen werknemers
descendant::naam[position()=1]	.//naam[1]	eerste naam element binnen werknemers
parent::node()/attribute::*	../@*	alle attributen van kantoor

Tussen vierkante haken kunnen we extra voorwaarden meegeven. Het volgende overzicht toont meer voorbeelden van locatiepaden:

`//werknemer`	alle werknemer nodes, ergens onder de root node
`//werknemer[sal>2000]`	de werknemer nodes met een sal node groter dan 2000
`//werknemer[@persnr=110]/*`	alle nodes met als parent de werknemer-node met een persnr attribuut met de waarde 110
`//werknemer[@persnr=110]/@*`	alle attribuut nodes van de werknemer node met een persnr attribuut met de waarde 110
`//werknemer[5]/sal`	de sal-node onder de vijfde werknemer node
`//werknemer[toeslag][sal>2000]`	alle werknemers met een toeslag node die meer verdienen dan 2000 euro
`//werknemer[contains(naam,"A")]`	alle werknemers met een A in de naam
`//werknemer[not(toeslag)]`	alle werknemers zonder toeslag
`//kantoor[count(//werknemer)>2]`	alle kantoren met meer dan 2 werknemers (ergens onder het kantoor)
`//werknemer/naam|//werknemer/sal`	de namen en salarissen van alle werknemers

In deze paden wordt begonnen met het root element. Het zijn daardoor absolute paden; niet relatief ten opzichte van een context node. In de Oracle functies die van XPath locatiepaden gebruik maken, kunnen we geen context node meegeven. De context node is daar steeds de root node. In deze functies zullen we dus absolute paden gebruiken.

4.3.1.1 existsNode()

De existsNode() functie controleert of een XPath evaluatie resulteert in één of meer XML elementen of tekst nodes. Indien dit het geval is wordt de waarde 1 geretourneerd, anders retourneert het de waarde 0. Deze functie heeft de volgende syntax:

```
EXISTSNODE (XMLType-instantie, XPath-expressie)
```

Neem bijvoorbeeld het volgende XML document:

```
<werknemer ID="145923">
  <naam>Hendrix</naam>
  <voorletters>M. G.</voorletters>
  <functieHistorie>
    <functie beginDatum="21-4-1995">trainee</functie>
    <functie beginDatum="13-6-1998">junior medewerker</functie>
    <functie beginDatum="01-04-2000">senior medewerker</functie>
  </functieHistorie>
</werknemer>
```

Een XPath expressie zoals `/werknemer//functie` resulteert in drie nodes. Daarom zal een existsNode() functie de waarde 1 retourneren.
Het volgende voorbeeld toont het kantoor met kantnr 40 uit de view xml_kantoren_vw :

```
select object_value
from xml_kantoren_vw
where existsnode(object_value,'//kantoor[@kantnr=40]') = 1;


OBJECT_VALUE
--------------------------------------------------------------------
<kantoor kantnr="40">
  <naam>PRODUCTIE</naam>
  <plaats>ARNHEM</plaats>
</kantoor>
```

Het gebruik van vierkante haken in een XPath-expressie is vergelijkbaar met een where-clausule in een SQL-query. Door [@kantnr="40""] wordt gezocht naar kantoren met een kantoorelement, waarvan een kantnr attribuut bestaat met de waarde "40".

ExistsNode() bestaat ook als member functie van XMLType. In dit geval wordt het XMLType object niet als parameter meegegeven, maar wordt de functie vanuit het object aangeroepen. We dienen in dit geval gebruik te maken van value(<alias>) in plaats van object_value. Bovenstaande query kan als volgt worden herschreven:

```
select object_value
from xml_kantoren_vw k
where value(k).existsnode('//kantoor[@kantnr=40]') = 1;
```

Vanaf Oracle 11g wordt het gebruik van existsNode() afgeraden. De hierna te bespreken functie XMLExists() zal existsNode() vervangen.

4.3.1.2 XMLExists()

Dit is een standaard SQL/XML functie, waarmee we kunnen controleren of een "XQuery expressie" een niet lege "XQuery sequence" oplevert. In een volgend hoofdstuk zullen we XQuery nader toelichten. Voor dit moment is het voldoende te weten dat we met XMLExists() dezelfde informatie kunnen ophalen als met existsNode(), alleen levert XMLExists() een boolean op, in plaats van een getal, waardoor we het in een WHERE clause en in een CASE kunnen gebruiken.

De syntax van XMLExists() is als volgt:

```
XMLExists ( XQuery_string ) [ XML_passing_clause ] )
```

waarbij XML_passing_clause er als volgt uitziet:

```
PASSING [BY VALUE] expressie [ AS identifier ] [, expressie [ AS identifier ] ... ]
```

Met de XML passing clause zonder identifier geven we de context aan voor de XQuery string. Eventueel kunnen er meer expressies mee worden gegeven, die dan wel een identifier moeten krijgen. Naar deze identifiers kan in de XQuery string verwezen worden. Het gebruik van identifiers wordt verderop nader toegelicht.

In het volgende voorbeeld wordt geen identifier gebruikt: met passing object_value geven we aan dat de XQuery string op object_value moet worden toegepast:

```
select object_value
from xml_kantoren_vw
where XMLExists('//kantoor[@kantnr=40]' PASSING object_value);
```

4.3.1.3 extract()

De functie extract() gebruikt een XPath expressie om een collectie nodes op te halen. De nodes worden geretourneerd als een instantie van XMLType. Dit resultaat kan een document zijn (één root met daarbinnen één of meer elementen), of een documentfragment (meerdere roots, dus geen geldig XML-document).

De functie extract() werd gebruikt in Oracle 10g en eerdere versies. Vanaf Oracle 11g wordt het gebruik van van extract() afgeraden. Gebruik in plaats daarvan de functie XMLQuery(), die in het volgende hoofdstuk behandeld wordt.

De functie extract() heeft de volgende syntax:

```
EXTRACT (XMLType-instantie, XPath-expressie)
```

Het volgende voorbeeld heeft een documentfragment als resultaat:

```
select extract(object_value,'//werknemer')
from xml_kantoren_vw
where existsnode(object_value,'//kantoor[@kantnr=10]') = 1;


EXTRACT(OBJECT_VALUE,'//WERKNEMER')
--------------------------------------------------------------------------------
<werknemer persnr="5810" mgr="6221">
  <naam>HEUVEL</naam>
  <sal>3450</sal>
  <functie>MANAGER</functie>
</werknemer>
<werknemer persnr="6221">
  <naam>KRAAY</naam>
  <sal>6000</sal>
  <functie>DIRECTEUR</functie>
</werknemer>
<werknemer persnr="8222" mgr="5810">
  <naam>MANDERS</naam>
  <sal>2300</sal>
  <functie>KLERK</functie>
</werknemer>
```

Vooruitlopend op een uitgebreider behandeling van de functie XMLQuery() tonen we hierbij vast hoe deze query kan worden vertaald naar een versie die geen gebruik maakt van extract() en existsnode():

```
select xmlquery('//werknemer' passing object_value returning content)
from xml_kantoren_vw
where xmlexists('//kantoor[@kantnr=10]' passing object_value);
```

4.3.1.4 extractValue()

De SQL functie extractValue() kan worden gebruikt om een tekstwaarde op te halen. Dit kan een text-node zijn, of de tekstwaarde van een element of attribuut. Deze functie heeft de volgende syntax:

```
EXTRACTVALUE (XMLType-instantie, XPath-expressie)
```

De functie extractValue() werd gebruikt in Oracle 10g en eerdere versies. Vanaf Oracle 11g wordt het gebruik van van extract() afgeraden. Gebruik in plaats daarvan de functie XMLTable(), of XMLQuery en XMLCast() die in het volgende hoofdstuk behandeld worden.

De functie extractValue() mag maar één waarde ophalen, anders volgt de volgende foutmelding:

```
SQL Error: ORA-19025: EXTRACTVALUE returns value of only one node
19025. 00000 -  "EXTRACTVALUE returns value of only one node"
*Cause:    Given XPath points to more than one node.
*Action:   Rewrite the query so that exactly one node is returned.
```

In het volgende voorbeeld wordt de functie van Manders opgehaald:

```
select extractValue(object_value,'//werknemer[naam="MANDERS"]/functie/text()')
from xml_kantoren_vw
where XMLExists('//kantoor[@kantnr=10]' PASSING object_value);

EXTRACTVALUE(OBJECT_VALUE,'//WERKNEMER[NAAM="MANDERS"]/FUNCTIE/TEXT()')
--------------------------------------------------------------------------------
KLERK
```

Zoals gezegd kunnen we vanaf Oracle 11g beter gebruik maken van andere functies. Deze functies zullen in het volgende hoofdstuk nader worden toegelicht. Hier laten we alvast zien hoe we met XMLQuery() en XMLCast(), of met XMLTable() de gewenste gegevens kunnen ophalen.

In onderstaand voorbeeld gebruiken we XMLQuery() om de nodes te vinden, en XMLCast om die nodes naar een scalar type om te zetten:

```
select xmlcast(
          xmlquery('//werknemer[naam="MANDERS"]/functie/text()'
                   passing object_value returning content)
       as varchar2(50))
from xml_kantoren_vw
where XMLExists('//kantoor[@kantnr=10]' PASSING object_value);
```

In dit en het volgende voorbeeld hoeven we ons niet te beperken tot één node.

Bovenstaand voorbeeld levert per gevonden kantoor één string op, die alle gevraagde functies bevat. Als we een rij per functie willen zien, kunnen we gebruik maken van XMLTable().

We zoeken nu naar alle functies van kantoor 10:

```
select xtab.column_value
from xml_kantoren_vw, xmltable('//werknemer/functie/text()'
                               passing object_value) xtab
where XMLExists('//kantoor[@kantnr=10]' PASSING object_value);


COLUMN_VALUE
---------------
MANAGER
DIRECTEUR
KLERK

3 rows selected
```

XMLTable() levert in dit geval een virtuele tabel op, met een rij per gevonden node().

4.4 Wijzigen van XML inhoud

In de volgende paragrafen zullen we enkele functies bespreken waarmee we (fragmenten van) een XMLType tabel of kolom kunnen aanpassen. Traditioneel kunnen we daar de functie updateXML() voor gebruiken. Sinds Oracle 11g hebben we daarnaast ook beschikking over deleteXML om fragmenten te verwijderen en diverse andere functies om XML fragmenten toe te voegen.

4.4.1 updateXML()

Met behulp van updateXML() kan een fragment van een XML-document in het geheugen worden aangepast. Deze aanpassing kunnen we vervolgens tonen of opslaan.
Stel bijvoorbeeld dat we in het eerste element b de tekst "test" willen wijzigen in "test1". Met een standaard update statement zouden we het hele XML document moeten vervangen. De updateXML() functie maakt het mogelijk om alleen die elementen te wijzigen die met een XPath-expressie worden geselecteerd.

De syntax van updateXML() is als volgt:

```
UPDATEXML (XMLType-instantie, XPath-expressie, waarde-expressie [, namespace])
```

Hierbij wordt een kopie van de XMLType instantie bewerkt. De XPath expressie bepaalt welk fragment van die XMLType instantie wordt bewerkt, de waarde-expressie geeft de nieuwe waarde van dat fragment. Optioneel kan de namespace van de xpath expressie worden meegegeven.

Indien de XPath-expressie een element ophaalt, dient de waarde-expressie een XMLType op te leveren. Als de XPath-expressie een attribuut of een text-node ophaalt, dan kan als waarde-expressie een *scalar* datatype worden gebruikt. Dit zijn de standaard datatypen, zoals number, date en varchar2.

In het volgende voorbeeld halen we eerst de xml_gegevens uit xml_kladblok op, van de rij met id=1.

```
select gegevens
from xml_kladblok
where id=1;
```

```
GEGEVENS
-------------------------------------
<a>
  <b>test</b>
  <b>
    <c>test2</c>
  </b>
</a>
```

We gaan nu de waarde van het eerste b-element wijzigen van test naar test1. Hiertoe dienen we expliciet aan te geven dat we de text node binnen dit element b willen wijzigen, dus niet het element b zelf:

```
update xml_kladblok
set gegevens = updatexml(gegevens,'//b[1]/text()','test1')
where id=1;


1 rows updated.


select gegevens
from xml_kladblok
where id=1;

GEGEVENS
-------------------------------------
<a>
  <b>test1</b>
  <b>
    <c>test2</c>
  </b>
</a>
```

Merk op dat de wijziging wordt vastgelegd door het update statement, niet door de functie updateXML zelf. Deze functie wijzigt slechts de tijdelijke representatie van het XML document in het geheugen. In dit voorbeeld hebben we de oorspronkelijke waarde van gegevens van id=1 vervangen door de aangepaste waarde van gegevens. Op vergelijkbare manier zouden we ook een xmltype tabel kunnen bijwerken, zoals:

```
update mijn_documenten
set object_value = updatexml(object_value, '//voorbeeld[@nr=1]/@nr','1b')
where xmlexists('//voorbeeld[@nr=1]' passing object_value);
```

We kunnen de updateXML() functie dus ook voor andere statements gebruiken, bijvoorbeeld in views om delen van het XML document af te schermen voor gebruikers. In de volgende selectie worden gegevens van de directeur weergegeven.

```
select xmlquery('//werknemer[functie="DIRECTEUR"]'
                passing object_value returning content)
from xml_kantoren_vw
where xmlexists('//werknemer[functie="DIRECTEUR"]' passing object_value);

XMLQUERY('//WERKNEMER[FUNCTIE="DIRECTEUR"]'PASSINGOBJECT_VALUERETURNINGCONTENT)
--------------------------------------------------------------------------------
<werknemer persnr="6221">
  <naam>KRAAY</naam>
  <sal>6000</sal>
  <functie>DIRECTEUR</functie>
</werknemer>

1 rows selected>
```

We kunnen een view maken om het salaris van de directeur af te schermen:

```
create or replace view xml_kantoren_vw_2 of xmltype
as select updatexml(object_value,'//werknemer[functie="DIRECTEUR"]/sal',null)
from xml_kantoren_vw;

create or replace view succeeded.
```

Door aan het element sal de waarde null toe te kennen wordt het een leeg element. Indien de tabel was aangemaakt op basis van een XML Schema, dan zou het element sal zijn verdwenen.

```
select xmlquery('//werknemer[functie="DIRECTEUR"]'
                passing object_value returning content)
from xml_kantoren_vw_2
where xmlexists('//werknemer[functie="DIRECTEUR"]'
                passing object_value);


XMLQUERY('//WERKNEMER[FUNCTIE="DIRECTEUR"]'PASSINGOBJECT_VALUERETURNINGCONTENT)
--------------------------------------------------------------------------------
<werknemer persnr="6221">
  <naam>KRAAY</naam>
  <sal/>
  <functie>DIRECTEUR</functie>
</werknemer>
```

Het is ook mogelijk om meerdere wijzigingen tegelijk uit te voeren met updateXML. In het volgende voorbeeld worden de naam en het persnr van de directeur gewijzigd.

```
select updatexml(xmlquery('//werknemer[functie="DIRECTEUR"]' passing object_value
returning content),
                 '/werknemer/@persnr','9999',
                 '/werknemer/naam/text()','GRIJS') directeur
from xml_kantoren_vw k
where xmlexists('//werknemer[functie="DIRECTEUR"]' passing object_value);

DIRECTEUR
--------------------------------------------------------------------------------
<werknemer persnr="9999">
  <naam>GRIJS</naam>
  <sal>6000</sal>
  <functie>DIRECTEUR</functie>
</werknemer>
```

4.4.2 deleteXML()

De functie deleteXML() is vergelijkbaar met updateXML(), en kan worden gebruikt om nodes uit een XML document te verwijderen. Ook hierbij gaat het weer om het bewerken van een kopie in het geheugen: om de wijziging in een tabel door te voeren, moeten we deleteXML() in een update statement toepassen.

De syntax van deleteXML() is als volgt:

```
DELETEXML (XMLType-instantie, XPath-expressie [, namespace])
```

In de vorige paragraaf hebben we een view aangemaakt, waarbij het salaris van de directeur leeg werd gemaakt met updateXML(). In het volgende voorbeeld vervangen we deze view door een versie waarbij het sal-element wordt verwijderd.

```
create or replace view xml_kantoren_vw_2 of xmltype
as select deletexml(object_value,'//werknemer[functie="DIRECTEUR"]/sal')
from xml_kantoren_vw;

create or replace view succeeded.
```

Als we nu het resultaat bekijken, zien we dat het element sal is verdwenen voor de directeur:

```
select xmlquery('//werknemer[functie="DIRECTEUR"]'
                passing object_value returning content)
from xml_kantoren_vw_2
where xmlexists('//werknemer[functie="DIRECTEUR"]'
                passing object_value);

<werknemer persnr="6221">
  <naam>KRAAY</naam>
  <functie>DIRECTEUR</functie>
</werknemer>
```

Merk op dat dit niet zomaar met updateXML() had gekund: het te wijzigen element zou dan het hele werknemer-element zijn, waarvan het grootste deel van de inhoud gelijk moet blijven. Met de meegegeven waarde-expressie zou die inhoud dan opnieuw moeten worden opgebouwd.

4.4.3 Elementen toevoegen

Voor het toevoegen van elementen zijn diverse functies beschikbaar. Met een XPath expressie wordt de node aangegeven waar de nieuwe inhoud moet worden toegevoegd. Afhankelijk van het soort functie, komt de inhoud dan voor of achter deze node, op hetzelfde niveau, of als child node.

4.4.3.1 insertChildXML()

Met deze functie kunnen we één of meer child nodes toevoegen. Een child node kan een sub-element of een attribuut zijn. De syntax ziet er als volgt uit:

```
INSERTCHILDXML (XMLtype-instantie , XPath_naar_parent, child_name, child_data
                [, namespace] )
```

Met een XPath expressie geven we één of meer parent nodes aan. Onder elke parent node wordt een child element of attribuut aangemaakt. Daarbij geven we eerst de naam van het element aan. Door voor de naam een @-teken te zetten kunnen we er een attribuut van maken. Vervolgens geven we de child data op. In het geval van een attribuut moet dat een VARCHAR2-waarde zijn. Als het een element is, dan is het een XMLType, waarvan het top-level element dezelfde naam moet hebben, als de opgegeven child name.

In het volgende voorbeeld selecteren we alle elementen van kantoor 10. Als werknemers geen element toeslag hebben, dan wordt deze aangemaakt met een waarde van 0:

```
select insertchildxml(object_value,
         '//werknemer[not(toeslag)]','toeslag',
          xmlelement("toeslag",0)) kantoor10
  from xml_kantoren_vw
  where xmlexists('//kantoor[@kantnr=10]'
                  passing object_value);
```

```
KANTOOR10
----------
<kantoor kantnr="10">
  <naam>BOEKHOUDING</naam>
  <plaats>AMSTERDAM</plaats>
  <werknemer persnr="5810" mgr="6221">
    <naam>HEUVEL</naam>
    <sal>3450</sal>
    <functie>MANAGER</functie>
    <toeslag>0</toeslag>
  </werknemer>
  <werknemer persnr="6221">
    <naam>KRAAY</naam>
    <sal>6000</sal>
    <functie>DIRECTEUR</functie>
    <toeslag>0</toeslag>
  </werknemer>
  <werknemer persnr="8222" mgr="5810">
    <naam>MANDERS</naam>
    <sal>2300</sal>
    <functie>KLERK</functie>
    <toeslag>0</toeslag>
  </werknemer>
</kantoor>

1 rows selected
```

In principe wordt het nieuwe element als laatste element toegevoegd. Als het document echter gekoppeld is aan een XML Schema, dan bepaalt het schema waar het element wordt geplaatst. Meer informatie over XML Schema's volgt in volgend hoofdstuk.

4.4.3.2 insertChildXMLBefore() en insertChildXMLAfter()

Soms is het wenselijk om zelf de plek te kunnen bepalen waar een child-element wordt aangemaakt. Stel bijvoorbeeld dat we het toeslag-element van het vorige voorbeeld willen aanmaken voor het element functie. We kunnen dan gebruik maken van de functies insertChildXMLBefore() en insertChildXMLAfter().

De syntax van insertChildXMLBefore ziet er als volgt uit:

```
INSERTCHILDXMLBEFORE (XMLType-instance, XPath_naar_parent, XPath_naar_child,
child_data [, namespace])
```

De syntax en werking van insertChildXMLAfter() ziet er hetzelfde uit: we beperken ons in de beschrijving verder tot insertChildXMLBefore()

Hierbij bepaalt XPath_naar_parent onder welk(e) element(en) de nieuwe child node moet worden aangemaakt. XPAth_naar_child geeft vervolgens aan voor welk element het nieuwe element moet worden geplaatst. Deze XPath expressie geeft impliciet ook het datatype aan van het nieuwe element.

In het volgende voorbeeld plaatsen we het element toeslag voor het element functie:

```
select insertChildXMLBefore(object_value,
         '//werknemer[not(toeslag)]','functie',
          xmlelement("toeslag",0)) kantoor10
from xml_kantoren_vw
where xmlexists('//kantoor[@kantnr=10]'
                 passing object_value);
```

```
KANTOOR10
--------------------
<kantoor kantnr="10">
  <naam>BOEKHOUDING</naam>
  <plaats>AMSTERDAM</plaats>
  <werknemer persnr="5810" mgr="6221">
    <naam>HEUVEL</naam>
    <sal>3450</sal>
    <toeslag>0</toeslag>
    <functie>MANAGER</functie>
  </werknemer>
  <werknemer persnr="6221">
    <naam>KRAAY</naam>
    <sal>6000</sal>
    <toeslag>0</toeslag>
    <functie>DIRECTEUR</functie>
  </werknemer>
  <werknemer persnr="8222" mgr="5810">
    <naam>MANDERS</naam>
    <sal>2300</sal>
    <toeslag>0</toeslag>
    <functie>KLERK</functie>
  </werknemer>
</kantoor>
```

4.4.3.3 insertXMLBefore() en insertXMLAfter()

Vergelijkbaar met insertChildXMLBefore() en insertChildXMLAfter() zijn de functies insertXMLBefore() en insertXMLAfter(). Het enige verschil is dat hierbij geen XPath_naar_parent wordt meegegeven.

De syntax van insertXMLBefore() is dus als volgt:

```
INSERTXMLBEFORE (XMLType-instance, XPath_naar_child, child_data [, namespace])
```

Hierbij is XPath_naar_child het pad naar het element, waarvoor het nieuwe element zal worden geplaatst. Analoog hieraan geeft XPath_naar_child bij insertXMLAfter() het element aan waarachter het nieuwe element zal worden geplaatst.

Het volgende voorbeeld toont een nieuwe werknemer voor de werknemer Kraay:

```
select insertXMLBefore(object_value,'/kantoor[@kantnr=10]/werknemer[naam="KRAAY"]',
       XMLParse(CONTENT
       '<werknemer persnr="9999" mgr="5810">
         <naam>GROEN</naam>
         <sal>2000</sal>
         <functie>KLERK</functie>
        </werknemer>'
        ))
from xml_kantoren_vw
where xmlexists('/kantoor[@kantnr=10]' passing object_value);
nieuwe werknemer
-----------------------
<kantoor kantnr="10">
  <naam>BOEKHOUDING</naam>
  <plaats>AMSTERDAM</plaats>
  <werknemer persnr="5810" mgr="6221">
    <naam>HEUVEL</naam>
    <sal>3450</sal>
    <functie>MANAGER</functie>
  </werknemer>
```

```
  <werknemer persnr="9999" mgr="5810">
    <naam>GROEN</naam>
    <sal>2000</sal>
    <functie>KLERK</functie>
  </werknemer>
  <werknemer persnr="6221">
    <naam>KRAAY</naam>
    <sal>6000</sal>
    <functie>DIRECTEUR</functie>
  </werknemer>
  <werknemer persnr="8222" mgr="5810">
    <naam>MANDERS</naam>
    <sal>2300</sal>
    <functie>KLERK</functie>
  </werknemer>
</kantoor>
```

4.4.3.4 appendChildXML()

De functie appendChildXML() voegt één of meer elementen als laatste child nodes toe aan een opgegeven parent element. De syntax van appendChildXML() is als volgt:

```
APPENDCHILDXML( XMLType-instance, XPath_naar_parent, child_data [,namespace])
```

De werking van deze functie is goeddeels gelijk aan die van insertChildXML(), met dit verschil dat appendChildXML() geen rekening houdt met een eventueel XML Schema.

4.4.3.5 Performance

Van de behandelde insert functies hebben insertChildXML(), insertChildXMLBefore() en insertChildXMLAfter() de voorkeur, omdat ze geoptimaliseerd zijn voor een betere performance.

4.5 XMLTransform()

De functie XMLTransform() transformeert een XML document met behulp van een XSLT stylesheet, en retourneert een XMLType. Deze functie heeft de volgende syntax:

```
XMLTRANSFORM(XMLType_instance, XMLType_instance)
```

Hoewel XMLType gegevens met behulp van XMLType functies en updateXML() kunnen worden aangepast, kan het handiger zijn om gebruik te maken van een stylesheet:

- Een XSLT stylesheet kan vaak op meerdere xml documenten worden toegepast;
- XSLT stylesheets zijn applicatie- en platform onafhankelijk, waardoor ze ook buiten de Oracle database gebruikt kunnen worden.
- Een XSLT stylesheet is zelf ook een XMLType, waardoor we deze kunnen bewerken voordat deze wordt toegepast;
- Bepaalde bewerkingen kunnen veel eenvoudiger met een XSLT stylesheet worden uitgevoerd dan met behulp van XMLType functies.

In het volgende voorbeeld wordt een XSLT stylesheet toegepast op de werknemers in xml_kantoren_vw. We zullen de werknemers presenteren in het default formaat zoals Oracle deze genereert uit relationele gegevens: elke set gegevens tussen ROWSET tags, elke rij tussen ROW tags, en elke kolomnaam in hoofdletters. XML bestanden in dit formaat kunnen met behulp van de package DBMS_XMLStore direct in een relationele tabel worden ingelezen (zie hoofdstuk 6). Hiervoor gebruiken we een stylesheet met o.a. de volgende templates:

- Elke root representeert een kantoor met werknemers. Hier begint elk ROWSET element.

- Elke werknemer representeert een rij: daar begint elk ROW element. Het attribuut kantnr uit het omliggende kantoorelement wordt daarin ook opgenomen.
- Elk attribuut en elk element wordt als element gegenereerd, met de naam van het element in hoofdletters. In XSLT bestaat helaas geen functie om tekst in hoofdletters om te zetten. We gebruiken daarom de functie translate, waarmee karakters in een string worden omgezet.

Hiervoor kunnen we het volgende stylesheet gebruiken.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
       <xsl:output method="xml" indent="yes" encoding="iso-8859-1" />
       <xsl:template match="/">
       <!-- start van de root, dan start van ROWSET -->
              <ROWSET>
                     <xsl:apply-templates select="//werknemer"/>
              </ROWSET>
       </xsl:template>
       <xsl:template match="werknemer">
       <!-- start van de element werknemer, dan start van ROWSET -->
              <ROW>
                     <xsl:apply-templates select="@*|*|ancestor::node()/@kantnr"/>
              </ROW>
       </xsl:template>
       <xsl:template match="*">
       <!-- elk volgend element, naam in hoofdletters -->
              <xsl:element name="{translate(name(),
              'abcdefghijklmnopqrstuvwxyz','ABCDEFGHIJKLMNOPQRSTUVWXYZ')}">
                     <xsl:value-of select="."/>
                     <xsl:apply-templates select="*|@*"/>
              </xsl:element>
       </xsl:template>
       <xsl:template match="@*">
       <!-- elk attribuut, maak hier ook een element van, naam in hoofdletters -->
              <xsl:element name="{translate(name(),
              'abcdefghijklmnopqrstuvwxyz','ABCDEFGHIJKLMNOPQRSTUVWXYZ')}">
                     <xsl:value-of select="."/>
              </xsl:element>
       </xsl:template>
</xsl:stylesheet>
```

We maken nu eerst een tabel aan om stylesheets in te bewaren:

```
create table stylesheets
    (naam varchar2(50), stylesheet xmltype);

create table succeeded.
```

Vervolgens voegen we deze stylesheet toe aan de tabel:

```
insert into stylesheets values ('werknemers.xsl',
xmltype.createxml(
'<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
....
    <xsl:element name="{translate(name(),
    ''abcdefghijklmnopqrstuvwxyz'',''ABCDEFGHIJKLMNOPQRSTUVWXYZ'')}"
....
</xsl:stylesheet>'))
/
```

We kunnen de stylesheet nu gebruiken:

```
select xmltransform(value(k), s.stylesheet) "werknemers kantoor 10"
from xml_kantoren_vw k, stylesheets s
where s.naam='werknemers.xsl'
and xmlexists('//kantoor[@kantnr=10]' passing object_value);


werknemers kantoor 10
<?xml version="1.0" encoding="utf-8"?>

<ROWSET>
  <ROW>
    <KANTNR>10</KANTNR>
    <PERSNR>5810</PERSNR>
    <MGR>6221</MGR>
    <NAAM>HEUVEL</NAAM>
    <SAL>3450</SAL>
    <FUNCTIE>MANAGER</FUNCTIE>
  </ROW>
  [...]
</ROWSET>
```

Sinds Oracle 11g kunnen we gebruik maken van XQuery in plaats van XSLT om complexe bewerkingen op XML gegevens te doen. Dit onderwerp wordt in het volgende hoofdstuk behandeld.

.

5 XQuery

5.1 Inleiding

Sinds Oracle 11g kunnen we gebruik maken van de taal XQuery om XML gegevens te op te vragen en te bewerken. Deze taal is geschikt om XML uit verschillende gegevensbronnen te benaderen, zoals tabellen, webservices, en fysieke documenten.
Oracle ondersteunt het gebruik van XQuery via de SQL/XML functies XMLQuery() en XMLTable(), aangevuld met diverse Oracle-specifieke functies. Daarnaast kan XQuery onder meer in PL/SQL en via het XQuery commando in SQL*Plus gebruikt worden.

5.2 De taal XQuery

XQuery is een taal vergelijkbaar met SQL en PL/SQL. SQL wordt gebruikt om relationele tabellen te benaderen, XQuery heeft hetzelfde doel voor XML-gegevens. De overeenkomst met PL/SQL is vooral het gebruik van loops en variabelen, die ook in XQuery veel gebruikt worden. XQuery is ontwikkeld om XML-gegevens van verschillende bronnen te kunnen raadplegen en/of te wijzigen. Dat kan bijvoorbeeld gaan om gegevens in tabellen, in losse bestanden, of beschikbaar gemaakt via web-services.
Naast het raadplegen, kan XQuery ook gebruikt worden voor het opbouwen van XML.-gegevens. In dit opzicht kan XQuery worden beschouwd als een alternatief voor XSLT. Het is daarom soms moeilijk te bepalen wanneer we voor XSLT moeten kiezen, en wanneer voor XQuery. Michael Kay, ontwikkelaar van de XSLT- en XQuery processor Saxon, geeft daarover de volgende aanbevelingen:

- gebruik XQuery als de gegevens in een database staan
- het kopiëren van een document met kleine wijzigingen is eenvoudiger in XSLT
- als we relatief weinig informatie uit een document willen halen, dan kunnen we beter XQuery gebruiken
- XQuery is makkelijker te leren en eenvoudiger voor kleine taken
- XQuery is geschikter voor duidelijk gestructureerde gegevens, XSLT voor gegevens die losser gestructureerd zijn
- Voor grote applicaties die gebruik maken van herbruikbare componenten is XSLT het meest geschikt.

Ten slotte wordt aangeraden om beide talen te leren, omdat 80% van de syntax in beide talen toepasbaar is.

XQuery maakt gebruik van een nieuw data model, gebaseerd op sequences. Het resultaat van elke XQuery expressie is een sequence: een reeks van nul of meer items, die elk enkelvoudige waarden, of XML nodes kunnen bevatten. Datatypen van een items zijn gebaseerd op XML Schema datatypen. Dit type systeem is een grote verbetering ten opzichte van XPath 1.0, waarin alleen simpele typen worden herkend, zoals boolean, number en string.

Sequences kunnen genest worden. In tegenstelling tot andere talen blijft de binnenste sequence-structuur dan niet intact: het resultaat van het nesten is een langere sequence. Met andere woorden: als we een sequence van drie nodes hebben, en we voegen op de tweede plek een sequence van twee nodes toe, dan is het resultaat een lange sequence van vijf nodes. De twee toegevoegde nodes zijn daarin niet meer als losse sequence te herkennen.

5 XQuery

De taal XQuery bestaat uit expressies die geëvalueerd worden en sequences retourneren. In principe is XQuery ook transparant, in de zin dat dezelfde expressie binnen dezelfde context ook altijd dezelfde waarde zal opleveren. Uitzondering hierop vormen de expressies die hun waarde ontlenen aan interactie met de externe omgeving. Denk hierbij aan het ophalen van de huidige tijd, of het laden van een extern document.

5.2.1 XQuery expressies

XQuery expressies zijn hoofdlettergevoelig. We hebben al voorbeelden gezien van XQuery expressies die gebruik maken van XPath. Daarnaast kan XQuery van diverse andere soorten expressies gebruik maken. De volgende tabel toont een overzicht.

Primaire expressies	Enkelvoudige waarden, variabelen en toegepaste functies. Een variabelenaam begint met een $-teken.
XPath expressies	Elk soort XPath expressie kan ook als XQuery expressie worden gebruikt, met hetzelfde resultaat. De XPath 2.0 standaard maakt deel uit van XQuery.
FLWOR expressies	Spreek uit: flower-expressies. Dit zijn de belangrijkste en krachtigste expressies, die in zichzelf eigenschappen bevatten van een programmeertaal. FLWOR expressies worden opgebouwd uit de volgende, naamgevende onderdelen: for, let, where, order by en return.
XQuery sequences	De komma (,) constructor maakt een sequence. Er bestaan ook sequence-manipulerende functies zoals union en intersect. XQuery sequences zijn een-diminsionaal: zodra sequences genest worden, ontstaat één uitgebreidere sequence. Zo wordt de volgende geneste sequence (1, 2, (3, 4), 5) verwerkt alsof er dit staat: (1, 2, 3, 4, 5)
Directe constructies	Met XQuery kan direct XML worden opgebouwd. Als binnen een XQuery expressie zoiets staat: <sport>wielrennen</sport>, dan wordt daarmee automatisch het betreffende item aangemaakt.
Dynamische constructies	Runtime kunnen waarden ook worden berekend. Zo kan het volgende item: `<som uitkomst="10" />` worden opgebouwd met deze XQuery expressie: `<som>{attribute uitkomst { 4+6} }</som>`. Tussen accolades wordt het dynamische deel meegegeven. Op deze manier kunnen zowel namen van elementen of attributen, als waarden worden berekend.
Conditionele expressies	Met if, then en else kunnen we voorwaarden meegeven.
Rekenkundige en relationele expressies	Met XQuery kunnen we ook berekeningen uitvoeren, of vergelijkingen evalueren, zoals $var * 3, $a < $b, $x eq $y
Quantifier expressies	Met de boolean expressies some <items> satisfies <conditie>, en every <items> satisfies <conditie> kunnen we achterhalen of sommige of alle items uit een sequence voldoen aan de betreffende conditie.
Reguliere expressies	XQuery ondersteunt het gebruik van reguliere expressies, gebaseerd op Perl en XML Schema 1.0.
Type expressies	Expressies die een XQuery type representeren, zoals node(), element(), attribute(), text() en enkelvoudige typen als xs:integer en xs:string.

5.2.2 FLWOR expressies

Van bovengenoemde expressies is de FLWOR expressie de krachtigste. FLWOR, uitgesproken als "flower" staat voor for, let, where, order by en return. Een FLWOR expressie heeft minimaal één for of let clausule, en een return clausule. Optioneel kan de expressie worden uitgebreid met een where clausule en/of een order by clausule.

De syntax van een FLWOR expressie is als volgt:

```
 { <for clausule> | <let clausule> } [[ <for clausule> | <let clausule> ] ...]
[ where <expressie> ] [ <order by clausule> ] return <expressie>
```

Er kunnen één of meer for- of let-clausules voorkomen, in willekeurige volgorde, optioneel gevolgd door where <expressie>, optioneel gevolgd door een order by clausule.

We zullen hier een overzicht geven van de vijf genoemde clausules.

for
Eén of meer variabelen krijgen in een loop achtereenvolgens één of meer waarden. In het volgende voorbeeld wordt de variabele $i achtereenvolgens aan 4, 6 en 7 gekoppeld:

```
for $i in (4, 6, 7)
```

Als er meer variabelen in een for clausule voorkomen, kan in een volgende variabele gebruik worden gemaakt van de waarde van een eerder genoemde variabele. We gebruiken de in operator om aan te geven dat de variabele achtereenvolgens de daarachter genoemde waarden krijgt. In het volgende voorbeeld krijgt $j bij de tweede iteratie de waarde 7 (6 + 1):

```
for $i in (4, 6, 7), $j in ($i, $i+1, $i+2)
```

We kunnen ook een reeks opgeven met een start- en eind waarde. Met de operator to geven we aan dat alle waarden daartussen moeten worden doorlopen, waarbij de waarde steeds met 1 wordt opgehoogd:

```
for $x in (1 to 10)
```

Een for loop kan ook gebaseerd zijn op een XPath expressie: hierbij worden alle nodes doorlopen die met de XPath expressie worden aangeduid, zoals:

```
for $klerk in $kantoor//werknemer[functie="KLERK"]
```

In dit geval zijn $kantoor en $klerk variabelen die XML gegevens bevatten.

let
Met let worden één of meer variabelen van een waarde voorzien. Net als bij for kan een volgende variabele gebruik maken van de waarde van een eerder genoemde variabele. We gebruiken de operator := voor het toekennen van een waarde. Bijvoorbeeld:

```
let $i := 3,  $j :=  3 + 1
```

Een veelgebruikte toepassing van let is het laden van een XML-document. In het volgende voorbeeld wordt een document kantoren.xml geladen. De variabele $kantoor bevat een sequence van kantoor-elementen.

```
let $kantoor := doc("kantoren.xml")//kantoor
```

De variabele $kantoor kan vervolgens weer met een for loop worden doorlopen.

where
Met where kunnen we filteren welke variabele waarden van let en for clausules zijn toegestaan. Het gebruik van where is vergelijkbaar met de SQL where clausule.
In het volgend voorbeeld worden alleen de even nummers tussen 1 en 10 geselecteerd (de mod operator geeft de rest van een deling: als de rest van een deling door 2 de waarde 0 heeft, weten we dat het een even nummer is):

```
for $i in (1 to 10)
where $i mod 2 = 0
```

In het volgende voorbeeld gebruiken we de where clausule om twee sequences aan elkaar te koppelen, in dit geval op basis van het attribuut kantnr:

```
for $kantoor in $kantoren/kantoor
for $werknemer in $werknemers/werknemer
where $werknemer/@kantnr=$kantoor/@kantnr
```

order by
Met order by kunnen we de volgorde van de uitvoer sturen. Zoals gezegd is de uitvoer van een XQuery expressie altijd een sequence (ook als het maar één item bevat). Een FLWOR expressie is de enige soort expressie waarin de volgorde van die items kan worden gestuurd. Bij andere expressies wordt de document-volgorde aangehouden.

return
Met return geven we het resultaat van de geordende en gefilterde waarden. Het resultaat van de gehele FLWOR expressie wordt als platte sequence geretourneerd.

Samengevat kunnen for en let worden gezien als de FROM clausule van een SQL statement, waarbij where en order by op dezelfde manier worden gebruikt als in SQL. De return clause is vergelijkbaar met de SELECT clausule van een SQL query. Een FLWOR expressie wordt dus in andere volgorde opgebouwd dan een SELECT statement, maar heeft er verder veel gelijkenis mee.

5.3 XMLQuery() en XMLTable()

Er zijn binnen Oracle XML DB twee SQL/XML functies waarmee we XQuery expressies kunnen uitvoeren: XMLQuery() en XMLTable(). Deze twee standaard functies vormen de interface tussen SQL en XQuery. Met behulp van deze functies kunnen we XML gegevens aanmaken op basis van relationele data, relationele gegevens opvragen alsof het in XML-formaat staat en relationele gegevens opbouwen uit XML gegevens.

Beide functies evalueren een XQuery expressie. In het geval van XMLQuery() wordt het resultaat - altijd een sequence van nodes - als één XML document of - fragment geretourneerd. De functie XMLTable() geeft de sequence terug als losse rijen van een SQL tabel, waarbij elke rij een item uit de sequence representeert.

5.3.1 XMLQuery()

De SQL/XML functie XMLQuery() wordt gebruikt om XML gegevens aan te maken of op te vragen. De functie XMLQuery heeft als eerste argument een string die de XQuery expressie bevat. Daarnaast kan een SQL expressie mee worden gegeven, waarmee het "context item" wordt bepaald. Dit context item geeft de XPath context aan, vanaf waar de XQuery expressie wordt geevalueerd. Vervolgens kunnen nog extra SQL expressies mee worden gegeven, waarvan de waarden aan XQuery variabelen worden gekoppeld tijdens het evalueren van de XQuery expressie. De functie retourneert het resultaat als een XMLType instance.

De syntax van XMLQuery() ziet er als volgt uit:

```
XMLQUERY( XQuery_string [ XML_passing_clause ] RETURNING CONTENT [ NULL ON EMPTY ] )
```

Waarbij de XML_passing_clause er als volgt uit ziet:

```
PASSING [ BY VALUE ] expressie [AS identifier] [, expressie [ AS identifier] ...]
```

De XQuery string is een complete XQuery expressie in string-vorm. De XML passing clause is het sleutelwoord PASSING, gevolgd door één of meer instanties van een enkelvoudig SQL type (dus geen object of collectie). De expressies geven de context aan. Met "AS identifier" kunnen we de waarde van zo'n expressie koppelen aan een XQuery variabele binnen de XQuery expressie. Er mag één expressie zijn zonder "AS identifier": het resultaat van die expressie is dan de context van de hele XQuery expressie.
Oracle ondersteunt niet alle specificaties van de SQL/XML functie XMLQuery(). Zo is PASSING impliciet een PASSING BY VALUE, omdat PASSING BY REFERENCE niet door Oracle wordt ondersteund. Verder is RETURNING CONTENT verplicht. Dit geeft aan dat het resultaat een XMLType fragment is met hooguit één root-node. Het alternatief, RETURNING SEQUENCE, wordt door Oracle niet ondersteund.

De context van een XQuery expressie kan een XMLType tabel, kolom, of view zijn. Daarnaast kan via de fn:doc() functie een XML document in de XML DB repository worden benaderd. Ook is het mogelijk om relationele tabellen te benaderen via ora:view(). In dit laatste geval wordt on the fly een XML-view over de betreffende tabellen aangemaakt.

5.3.1.1 XMLQuery met XMLType gegevens

In het volgende voorbeeld benaderen we de XML gegevens van kantoren en werknemers. We maken een overzicht van alle functies, met de gemiddelde salarissen van die functies.

We maken eerst een nieuwe view xml_kantoren_vw_3 aan, met één root element kantoren, zodat we de gemiddelde salarissen per functie over alle kantoren, in plaats van per kantoor kunnen berekenen:

```
create or replace view xml_kantoren_vw_3 of xmltype with object oid (1)
as select xmlelement("kantoren",(select xmlagg(object_value) from xml_kantoren_vw))
from dual;
```

We zullen nu de query stap voor stap opbouwen.
Eerst halen we alle werknemers op, als een XPath expressie:

```
select xmlquery(
   '//werknemer'
   passing object_value
returning content) as werknemers
from xml_kantoren_vw_3;
```

```
WERKNEMERS
----------------------------------------------------------
<werknemer persnr="5810" mgr="6221"><naam>HEUVEL</naam><sal>3450</sal>...</werknemer>

1 rows selected
```

Hierna gebruiken we voor dezelfde selectie een eenvoudige xquery expressie. Zoals eerder vermeld bestaat een xquery expressie minimaal uit een let of een for clause, en een return clause.

```
select xmlquery(
   'let $werknemers := //werknemer
    return $werknemers'
   passing object_value
returning content) as werknemers
from xml_kantoren_vw_3;

WERKNEMERS
----------------------------------------------------------
<werknemer persnr="5810" mgr="6221"><naam>HEUVEL</naam><sal>3450</sal>...</werknemer>
```

Nu halen we alle verschillende functies op. In SQL doen we dat met DISTINCT, in XQuery gebruiken we distinct-values:

```
select xmlquery(
   'let $werknemers := //werknemer
    for $functie in distinct-values($werknemers/functie)
    return $functie'
   passing object_value
returning content) as functies
from xml_kantoren_vw_3;


FUNCTIES
--------------------------------------------------------------------------------
ANALIST DIRECTEUR KLERK MANAGER VERKOPER
```

Merk op dat nu alleen de text() nodes van de functies worden weergegeven, en geen functie-elementen. De distinct-values() functie haalt dus de tekstwaarden op.

Als we er toch functie-elementen van willen maken, moeten we die zelf aanmaken:

```
select xmlquery(
   'let $werknemers := //werknemer
    for $functie in distinct-values($werknemers/functie)
    return <functie>{$functie}</functie>'
   passing object_value
returning content) as functies
from xml_kantoren_vw_3;


FUNCTIES
----------------------------
<functie>ANALIST</functie><functie>DIRECTEUR</functie>...<functie>VERKOPER</functie>
```

Merk op dat hier de waarde van elke functie tussen accolades staat, om deze waarde te onderscheiden van de letterlijk over te nemen tags <functie> en </functie>. In het algemeen worden accolades geplaatst om berekende waarden, als de context letterlijk moet worden geinterpreteerd. Dit zien we ook in de volgende stappen.

In voorgaand voorbeeld is duidelijk te zien dat de uitvoer van een XQuery expressie een sequence van items is. Stel dat we daar één root element functies omheen willen zetten, dan moeten we de hele XQuery expressie weer tussen accolades zetten.

```
select xmlquery(
   '<functies>{
    let $werknemers := //werknemer
    for $functie in distinct-values($werknemers/functie)
    return <functie>{$functie}</functie>
    }</functies>'
   passing object_value
returning content) as functies
from xml_kantoren_vw_3;


FUNCTIES
----------------------------
<functies><functie>ANALIST</functie>...<functie>VERKOPER</functie></functies>
```

Nu berekenen we ook de gemiddelde salarissen van de functies. Per gevonden functie maken we een variabele $sal aan, die het gemiddelde salaris bevat.

```
select xmlquery(
'<functies>{
   let $werknemers := //werknemer
   for $functie in distinct-values($werknemers/functie)
    let $sal := $werknemers[functie=$functie]/sal
   return <functie>{$functie} {avg($sal)}</functie>}
 </functies>' passing object_value
returning content) as functies
from xml_kantoren_vw_3;


FUNCTIES
------------------------------
<functies><functie>ANALIST3950</functie>...<functie>VERKOPER2400</functie></functies>
```

Zoals gezegd doorlopen we met for alle verschillende functies. Per functie die we tegenkomen maken we een variabele $sal aan, die de salarissen bevat van alle werknemers met dezelfde functie. Dit geven we aan met `[functie=$functie]`.

Nu staat de naam en het gemiddelde salaris samen binnen de functie-tags. Het is logischer om er twee sub-elementen of twee attributen van te maken. We maken er twee attributen van:

```
select xmlquery(
'<functies>{
   let $werknemers := //werknemer
   for $functie in distinct-values($werknemers/functie)
    let $sal := $werknemers[functie=$functie]/sal
   return <functie naam="{$functie}" avgsal="{avg($sal)}"/>}
 </functies>' passing object_value
returning content) as functies
from xml_kantoren_vw_3;

FUNCTIES
---------------------------------
<functies><functie naam="ANALIST" avgsal="3950"></functie>...</functies>
```

Merk op dat $functie een element is, en geen text-node, en dat we dat hele element gebruiken als waarde van een attribuut.

Tot slot een voorbeeld waarbij het element functie, en de attributen naam en avgsal, dynamisch worden aangemaakt:

```
select xmlquery(
'<functies>{
   let $werknemers := //werknemer
   for $functie in distinct-values($werknemers/functie)
    let $sal := $werknemers[functie=$functie]/sal
   return element functie { attribute naam {$functie},
                            attribute avgssal {avg($sal)}}
}</functies>' passing object_value
returning content) as functies
from xml_kantoren_vw_3;
```

Hier is gebruik gemaakt van de constructors element en attribute. Deze krijgen als parameters eerst de naam, en daarna de waarde. Hier staat dus dat het element functie een attribuut naam krijgt met de waarde van $functie, en een attribuut avgsal met de waarde avg($sal).

We hebben in deze voorbeelden gebruik gemaakt van een nieuwe view, die één record met alle kantoren bevat. Een alternatief is gebruik maken van xmlagg() in de from clause:

```
select xmlquery('<functies>{
  let $werknemers := //werknemer
  for $functie in distinct-values($werknemers/functie)
    let $sal := $werknemers[functie=$functie]/sal
    return element functie { attribute naam {$functie},
                             attribute avgssal {avg($sal)}}
}</functies>' passing kantoren
returning content) as functies
from (select xmlagg(object_value) as kantoren
      from xml_kantoren_vw);
```

5.3.1.2 XMLQuery met XML DB documenten

XQuery kan ook werken met externe documenten, via de functie doc(). De functie XMLQuery() kan op vergelijkbare manier werken met documenten die in XML DB zijn geladen.

Door de functie doc() kunnen we eenvoudig de gegevens uit meerdere documenten tegelijk benaderen. Dit is te vergelijken met een join of een subselectie in SQL. We halen nu de stafleden op van het ziekenhuis AMC die in de afdeling Psychiatrie werken:

```
select xmlquery(
'let $pad := "/home/cursisten/&1/ziekenhuizen/",
     $zie := doc(concat($pad,"ziekenhuizen.xml")
                 )//ziekenhuis[naam="AMC"],
     $afd := doc(concat($pad,"afdelingen.xml")
                 )//afdeling[naam="Psychiatrie"][@ziekhnr=$zie/@ziekhnr]
 for $sta in doc(concat($pad,"stafleden.xml")
                 )//staflid[@ziekhnr=$afd/@ziekhnr][@afdnr=$afd/@afdnr]
 return $sta'
 returning content ) as "Stafleden AMC Psychiatrie"
from dual;
```

```
Stafleden AMC Psychiatrie
--------------------------
<staflid ziekhnr="10" afdnr="6" persnr="3526">
  <naam>Dinter B.</naam>
  <functie>Verpleegster</functie>
  <dienst>A</dienst>
  <sal>17400</sal>
</staflid>
<staflid ziekhnr="10" afdnr="6" persnr="3198">
  <naam>Hursman J.</naam>
  <functie>Zaalknecht</functie>
  <dienst>A</dienst>
  <sal>13500</sal>
</staflid>
```

Eerst worden hier een paar variabelen aangemaakt. Het pad naar de map ziekenhuizen wordt steeds geplakt aan de bestandsnaam, met de functie concat(). Met de functie doc() kunnen we de gegevens van een document laden. Direct daarachter kunnen we met een XPath expressie delen van dat document benaderen, zoals `doc(...)//ziekenhuis`.
Ook is het mogelijk om in zo'n XPath-expressie te verwijzen naar een eerder genoemde variabele, zoals: `doc(...)//afdeling[@ziekhnr=$zie/@ziekhnr]`
Op deze manier wordt een koppeling gemaakt tussen het ene document het andere document.

We kunnen het koppelen van gegevens ook in de where clause regelen, in plaats van in de XPath expressie. We moeten dan wel goed rekening houden met de manier waarop XQuery vergelijkingen worden opgebouwd.
We doen dat eerst op een intuïtieve maar foutieve manier:

```
select xmlquery(
'let $pad := "/home/cursisten/&1/ziekenhuizen/",
     $zie := doc(concat($pad,"ziekenhuizen.xml")
                 )//ziekenhuis[naam="AMC"],
     $afd := doc(concat($pad,"afdelingen.xml")
                 )//afdeling[naam="Psychiatrie"]
 for $sta in doc(concat($pad,"stafleden.xml")
                 )//staflid
 where $zie/@ziekhnr = $afd/@ziekhnr
 and   $afd/@ziekhnr = $sta/@ziekhnr
 and   $afd/@afdnr = $sta/@afdnr
 return $sta'
 returning content ) as "Stafleden AMC Psychiatrie"
from dual;

Stafleden AMC Psychiatrie
--------------------------------------------------
<staflid ziekhnr="10" afdnr="6" persnr="3526">
  <naam>Dinter B.</naam>
  <functie>Verpleegster</functie>
  <dienst>A</dienst>
  <sal>17400</sal>
</staflid>
<staflid ziekhnr="10" afdnr="6" persnr="3198">
  <naam>Hursman J.</naam>
  <functie>Zaalknecht</functie>
  <dienst>A</dienst>
  <sal>13500</sal>
</staflid>
<staflid ziekhnr="20" afdnr="6" persnr="2315">
  <naam>Horst D.</naam>
  <functie>Verpleegster</functie>
  <dienst>N</dienst>
  <sal>18300</sal>
</staflid>
```

```
<staflid ziekhnr="20" afdnr="6" persnr="8574">
  <naam>Beek G.</naam>
  <functie>Zaalknecht</functie>
  <dienst>N</dienst>
  <sal>12600</sal>
</staflid>
```

We zien nu teveel stafleden verschijnen: deze stafleden werken niet allemaal op ziekenhuis nummer 10. Hoe komt dat?

De variabele $zie bevat het ziekenhuis AMC (ziekhnr 10). De variabele $afd bevat een sequence van *alle* afdelingen met de naam Psychiatrie, ongeacht het ziekenhuisnummer. In de for loop wordt voor elk staflid de where clause geëvalueerd. Het gaat mis aan het begin: "... where $zie/@ziekhnr = $afd/@zieknhr...". Daar wordt namelijk één waarde (10) vergeleken met een sequence van meerdere waarden (10, 20).

In XQuery kunnen ook sequences met elkaar vergeleken worden, met operatoren als =, !=, <, >, <= en >=. De vergelijking geeft dan de waarde true, als voor minimaal één combinatie van waarden links en rechts van de vergelijking de waarde true is. Dus bijvoorbeeld (3, 5) < (2, 4) is true, want 3 < 4 is true. Voor onze query betekent dit dat *alle* afdelingen in $afd worden doorlopen, omdat er *één* afdeling tussen zit met @ziekhnr=10.

Om er toch voor te zorgen dat alleen de afdeling met @ziekhnr=10 mee doet, kunnen we elke afdeling doorlopen met een for loop, in plaats van een sequence aan te maken met let.

Dit doen we in het volgende voorbeeld:

```
select xmlquery(
'let $pad := "/home/cursisten/&1/ziekenhuizen/",
     $zie := doc(concat($pad,"ziekenhuizen.xml")
                 )//ziekenhuis[naam="AMC"]
 for $afd in doc(concat($pad,"afdelingen.xml")
                 )//afdeling[naam="Psychiatrie"],
     $sta in doc(concat($pad,"stafleden.xml")
                 )//staflid
 where $zie/@ziekhnr = $afd/@ziekhnr
 and   $afd/@ziekhnr = $sta/@ziekhnr
 and   $afd/@afdnr = $sta/@afdnr
 return $sta'
 returning content ) as "Stafleden AMC Psychiatrie"
from dual;

Stafleden AMC Psychiatrie
------------------------------------------------
<staflid ziekhnr="10" afdnr="6" persnr="3526">
  <naam>Dinter B.</naam>
  <functie>Verpleegster</functie>
  <dienst>A</dienst>
  <sal>17400</sal>
</staflid>
<staflid ziekhnr="10" afdnr="6" persnr="3198">
  <naam>Hursman J.</naam>
  <functie>Zaalknecht</functie>
  <dienst>A</dienst>
  <sal>13500</sal>
</staflid>
```

Met de for loop controleren we nu voor elke afdeling afzonderlijk of het ziekenhuisnummer gelijk is aan $zie/@ziekhnr.

5.3.1.3 XMLQuery met relationele tabellen - ora:view

De Oracle extensie-functie ora:view maakt runtime een XMLType view aan over relationele tabellen. Alle velden worden daarin als elementen opgenomen met de element- of aliasnaam als elementnaam. Deze elementen staan per rij van de relationele tabel in een ROW-element. Al deze ROW-elementen worden samen in één rij weergegeven.

De syntax van ora:view is als volgt:

```
ora:view ([schemanaam STRING,] tabelnaam STRING) RETURNS document-
node(element())*
```

De functie ora:view krijgt dus optioneel een schemanaam mee, en verplicht een tabelnaam. Het sterretje achter document-node(element()) geeft aan dat er nul of meer document-nodes van het type element getourneerd worden. Voor relationele tabellen gaat dat om ROW elementen. De functie ora:view kan ook voor XMLType tabellen worden gebruikt. In dat geval worden geen ROW elementen gegenereerd: de XML-structuur van die tabellen blijft dan gewoon gehandhaafd.

In het volgende eenvoudige voorbeeld zien we hoe ora:view een relationele tabel omzet in XML (voor de leesbaarheid met pretty-print weergegeven). Toon de patienten uit Utrecht:

```
select xmlquery(' for $rij in ora:view("OX_PATIENTEN")/ROW[PLAATS="Utrecht"]
                  return $rij
                ' returning content) FROM dual;


RIJ
--------------------------------
<ROW>
  <PATNR>11321</PATNR>
  <NAAM>Koopmans M.</NAAM>
  <PLAATS>Utrecht</PLAATS>
  <GEBDAT>1966-12-11</GEBDAT>
  <MV>M</MV>
  <ZIEKFNR>3542764</ZIEKFNR>
</ROW>
<ROW>
  <PATNR>25218</PATNR>
  <NAAM>Dekker B.</NAAM>
  <PLAATS>Utrecht</PLAATS>
  <GEBDAT>1954-11-05</GEBDAT>
  <MV>M</MV>
  <ZIEKFNR>8466355</ZIEKFNR>
</ROW>
<ROW>
  <PATNR>50333</PATNR>
  <NAAM>Horst E.</NAAM>
  <PLAATS>Utrecht</PLAATS>
  <GEBDAT>1964-04-12</GEBDAT>
  <MV>M</MV>
  <ZIEKFNR>1232988</ZIEKFNR>
</ROW>
```

Merk op dat we nu geen "passing object_value" hebben staan: van dual gebruiken we geen object_value. De basis waarop de XPath expressies uitvoeren is nu ora:view("OX_PATIENTEN").

Stel nu dat we elke rij niet als ROW maar als patient willen aanduiden, en dat we ook alle kolomnamen in kleine letters willen weergeven. Dan zullen we de gewenste gegevens zelf moeten opbouwen, zoals in het volgende voorbeeld, voor de patienten uit Nijmegen:

```
select xmlquery(' for $rij in ora:view("OX_PATIENTEN")/ROW[PLAATS="Nijmegen"]
                  return <patient>{
                  for $kolom in  $rij/*
                  return element {lower-case(name($kolom))} {$kolom/text()}
                  }</patient>' returning content) as "patient"
                  FROM dual;

patient
--------------------
<patient>
  <patnr>38911</patnr>
  <naam>Jong H.</naam>
  <plaats>Nijmegen</plaats>
  <gebdat>1982-01-12</gebdat>
  <mv>M</mv>
  <ziekfnr>4656238</ziekfnr>
</patient>
```

In dit geval doorlopen we elk "kolom-element" met een for loop. Met de constructor element maken we een nieuw element aan voor elke kolom. De naam van het element wordt berekend met lower-case(name($kolom)). Hiermee halen we de door ora:view gemaakte elementnaam op, en zetten die in kleine letters om. De inhoud van het nieuwe element wordt de text-node van het kolom-element.

5.3.2 XMLTable()

Met de SQL/XML functie XMLTable() kunnen we het resultaat van een XQuery expressie opbreken in rijen en kolommen van een nieuwe virtuele tabel. Zo'n XMLTable() functie wordt in de FROM clause gebruikt van een query.

De syntax van XMLTable() ziet er als volgt uit:

```
XMLTABLE ( [ XML_namespaces_clause , ] XQuery_string XMLTable_options )
```

Hierbij is dit de XML_namespaces_clause:

```
XMLNAMESPACES ( [ string AS identifier [ , string AS identifier [...] ] ]
[ DEFAULT string ] )
```

Dit is de syntax van de XMLTable_options:

```
[ XML_passing_clause ] [ COLUMNS XML_table_column [, XML_table_column [ , ...] ] ]
```

Hiervan hebben we de XML_passing_clause eerder gezien:

```
PASSING [ BY VALUE ] expressie [ AS identifier [ , expressie [ AS identifier ]
[, ...] ] ]
```

Dit is de syntax van XML_table_column:

```
column [ FOR ORDINALITY | datatype [ PATH string ] [ DEFAULT expressie ] ]
```

Er bestaan dus veel onderdelen van XMLTable(). Het enige verplichte deel daarvan is de XQuery_string. Aan de hand van enkele voorbeelden zullen we de belangrijkste clausules van de XMLTable functie nader toelichten.

In het volgende voorbeeld gebruiken we weer ora:view om runtime een XMLType view te maken op de tabel OX_Patienten. We hebben gezien dat een query met XMLQuery op zo'n view het gehele resultaat als één rij weergeeft. We gebruiken nu XMLTable om elk ROW element als losse rij weer te geven:

```
select *
from  XMLTable('for $rij in ora:view("OX_PATIENTEN")/ROW
                return $rij');

COLUMN_VALUE
----------------------------------------
<ROW><PATNR>11321</PATNR><NAAM>Koopmans M.</NAAM>...</ROW>
<ROW><PATNR>12816</PATNR><NAAM>Schouten W.</NAAM>...</ROW>
<ROW><PATNR>19381</PATNR><NAAM>Elbers M.</NAAM>...</ROW>
<ROW><PATNR>25218</PATNR><NAAM>Dekker B.</NAAM>...</ROW>
[...]
```

We zien hier dat elk ROW element als losse rij wordt getoond, als een virtuele kolom met de naam column_value. Bekijk in de uitvoer ook de waarden van het element GEBDAT. Die krijgen het formaat yyyy-mm-dd: het standaard date formaat van het XML Schema datatype xs:date.

We kunnen ook verder gaan, en per rij meerdere kolommen benoemen. Na de XQuery string geven we deze kolommen aan met:

```
COLUMNS KOLOM DATATYPE PATH XPATH-EXPRESSIE
```

waarbij XPath-expressie steeds een verwijzing is naar een plek ten opzichte van de XQuery string. We zien dat in het volgende voorbeeld (xmltable_patienten.sql):

```
select naam, plaats, gebdat
from  XMLTable('for $rij in ora:view("OX_PATIENTEN")/ROW
                return $rij' columns
                naam varchar2(30) path 'NAAM',
                plaats varchar2(30) path 'PLAATS',
                gebdat date path 'GEBDAT')
order by gebdat;

NAAM                           PLAATS                         GEBDAT
------------------------------ ------------------------------ ---------------------
Lammers T.                     Arnhem                         12-APR-43
Ravenhorst P.                  Eindhoven                      04-FEB-48
Dekker B.                      Utrecht                        05-NOV-54
Horst E.                       Utrecht                        12-APR-64
Koopmans M.                    Utrecht                        11-DEC-66
Manders G.                     Den Bosch                      11-DEC-70
Schouten W.                    Den Haag                       23-APR-73
Elbers M.                      Amsterdam                      01-JAN-76
Feenstra A.                    Breda                          27-FEB-77
Jong H.                        Nijmegen                       12-JAN-82
```

De context, van waaruit het pad naar elke kolom wordt bepaald, is het ROW element. Daaronder wordt per kolom de naam, het datatype en het pad bepaald. Merk op dat de gegenereerde kolom GEBDAT weer in het default date formaat wordt weergegeven.

5.3.2.1 Meerdere niveaus opbreken met XMLTable

Met behulp van de functie XMLTable() kunnen we ook XML gegevens die in meerdere niveaus staan opbreken. Voor elk niveau gebruiken we dan een nieuwe XMLTable() aanroep in de from clause. We kunnen in de from clause verwijzen naar de gegevens van een vorige tabel of XMLTable() aanroep met de passing clause. Bekijk bijvoorbeeld dit fragment:

```
... FROM xml_ziekenhuizen z,
         XMLTable('//ziekenhuis' passing z.object_value) ...
```

Hiermee geven we aan dat de context van de XPath expressie '//ziekenhuis' de object_value is van xml_ziekenhuizen. Ook kunnen we met aliassen werken, bijvoorbeeld:

```
... FROM xml_ziekenhuizen z,
         XMLTable('$zie//ziekenhuis' passing z.object_value as "zie")
```

Bekijk nu het volgende voorbeeld (xmltable_kantoren.sql):

```
select w.werknemer, w.functie, k.kantoor
from xml_kantoren_vw_3 ko,
    XMLTable('//kantoor' passing ko.object_value
              columns kantoor varchar2(15) path 'naam',
                      werknemer xmltype path 'werknemer') k,
    XMLTable('/werknemer' passing k.werknemer
               columns werknemer varchar2(15) path 'naam',
                       functie   varchar2(15) path 'functie') w
```

```
WERKNEMER       FUNCTIE         KANTOOR
--------------- --------------- ---------------
HEUVEL          MANAGER         BOEKHOUDING
KRAAY           DIRECTEUR       BOEKHOUDING
MANDERS         KLERK           BOEKHOUDING
SMITS           KLERK           ONDERZOEK
PIETERS         MANAGER         ONDERZOEK
SANDERS         ANALIST         ONDERZOEK
ADELAAR         KLERK           ONDERZOEK
VERMEULEN       ANALIST         ONDERZOEK
ALKEMA          VERKOPER        VERKOOP
WALSTRA         VERKOPER        VERKOOP
VERGEER         VERKOPER        VERKOOP
KLAASEN         MANAGER         VERKOOP
DROST           VERKOPER        VERKOOP
APPEL           KLERK           VERKOOP
```

We zullen de from clause van deze query nu in detail bespreken.
De basis van de query vormt de view xml_kantoren_vw_3. Bekijk zelf nog eens de structuur van deze view: er bestaat een root element "kantoren", met daarbinnen kantoorelementen. Binnen elk kantoorelement kunnen ook werknemerelementen voorkomen.
xml_kantoren_vw_3 krijgt de alias "ko"
In de eerste aanroep van XMLTable() is **ko.object_value** de context van het pad '//kantoor'. Dit betekent dat alle kantoorelementen van de view xml_kantoren_vw doorlopen worden. Vanuit elk kantoorelement maken we de kolom kantoor aan, die verwijst naar het element naam onder kantoor: `columns kantoor varchar2(15) path 'naam'`
Daarnaast maken we een kolom werknemer aan, van het type xmltype, die verwijst naar elke werknemer onder kantoor: `werknemer xmltype path 'werknemer'`. Deze kolom wordt voor de volgende XMLTable() aanroep gebruikt. De eerste XMLTable() aanroep krijgt de alias k.

Bij de tweede XMLTable() aanroep is **k.werknemer** de context voor het pad '/werknemer'. Dit betekent dat alle werknemerelementen doorlopen worden, per kantoor van de vorige XMLTable() aanroep. Hier maken we de kolommen werknemer en functie aan, gebaseerd op de elementen naam en functie onder werknemer.

Het doorgeven van XML gegevens met "passing" naar een volgende XMLTable() aanroep, werkt als een gewone inner join. In dit geval worden dus alleen de kantoren getoond waar werknemers werkzaam zijn. Het kantoor zonder werknemers wordt niet getoond. Als we die wel willen zien, kunnen we dat aangeven met (+) direct achter de virtuele tabel die met lege waarden moet worden aangevuld:

```
select w.werknemer, w.functie, k.kantoor
from xml_kantoren_vw_3 ko,
    XMLTable('//kantoor' passing ko.object_value
             columns kantoor varchar2(15) path 'naam',
                     werknemer xmltype path 'werknemer') k,
    XMLTable('/werknemer' passing k.werknemer
              columns werknemer varchar2(15) path 'naam',
                      functie   varchar2(15) path 'functie')(+) w
```

```
WERKNEMER       FUNCTIE         KANTOOR
--------------- --------------- --------------
HEUVEL          MANAGER         BOEKHOUDING
KRAAY           DIRECTEUR       BOEKHOUDING
MANDERS         KLERK           BOEKHOUDING
SMITS           KLERK           ONDERZOEK
PIETERS         MANAGER         ONDERZOEK
SANDERS         ANALIST         ONDERZOEK
ADELAAR         KLERK           ONDERZOEK
VERMEULEN       ANALIST         ONDERZOEK
ALKEMA          VERKOPER        VERKOOP
WALSTRA         VERKOPER        VERKOOP
VERGEER         VERKOPER        VERKOOP
KLAASEN         MANAGER         VERKOOP
DROST           VERKOPER        VERKOOP
APPEL           KLERK           VERKOOP
                                PRODUCTIE
```

Zoals gezegd kunnen we ook meer waarden doorgeven via passing. Er mag maar één waarde zonder alias worden doorgegeven: die geeft de default context aan. Andere waarden krijgen een alias, waar we in het pad naar kunnen verwijzen. We zien dit in de volgende uitbreiding, waarin ook de managers van de werknemers worden opgehaald :

```
select w.werknemer, w.functie, k.kantoor, m.manager
from xml_kantoren_vw_3 ko,
    XMLTable('//kantoor' passing ko.object_value
       columns kantoor varchar2(15) path 'naam',
               werknemer xmltype path 'werknemer') k,
    XMLTable('/werknemer' passing k.werknemer
       columns werknemer varchar2(15) path 'naam',
               functie   varchar2(15) path 'functie',
               mgr       number path '@mgr') w,
    XMLTable('//werknemer[@persnr=$mgr]' passing ko.object_value, w.mgr as "mgr"
       columns manager varchar2(15) path 'naam') m;
```

```
WERKNEMER       FUNCTIE         KANTOOR         MANAGER
--------------- --------------- --------------- ---------------
HEUVEL          MANAGER         BOEKHOUDING     KRAAY
MANDERS         KLERK           BOEKHOUDING     HEUVEL
SMITS           KLERK           ONDERZOEK       VERMEULEN
PIETERS         MANAGER         ONDERZOEK       KRAAY
SANDERS         ANALIST         ONDERZOEK       PIETERS
ADELAAR         KLERK           ONDERZOEK       SANDERS
VERMEULEN       ANALIST         ONDERZOEK       PIETERS
ALKEMA          VERKOPER        VERKOOP         KLAASEN
WALSTRA         VERKOPER        VERKOOP         KLAASEN
VERGEER         VERKOPER        VERKOOP         KLAASEN
KLAASEN         MANAGER         VERKOOP         KRAAY
DROST           VERKOPER        VERKOOP         KLAASEN
APPEL           KLERK           VERKOOP         KLAASEN
```

In de tweede XMLTable() wordt nu als extra kolom het mgr attribuut opgehaald van de werknemers. In de derde XMLTable() worden nu twee waarden gebruikt:

```
... passing ko.object_value, w.mgr as "mgr"
```

De default context is het hele XML document in ko.object_value. Daarnaast is er een context met de alias `"mgr"`, die verwijst naar het mgr attribuut van de betreffende werknemer. met het pad `'//werknemer[@persnr=$mgr]'` geven we aan dat we zoeken naar de werknemer, waarvan het persnr attribuut gelijk is aan het opgehaalde mgr attribuut. Van die werknemer willen we het element naam als ophalen. We noemen deze kolom "manager".

5.4 XMLCast()

Soms is het nodig om een waarde uit XML gegevens naar een standaard Oracle datatype als number, varchar2 of date te vertalen. Daarvoor gebruiken we de functie XMLCast(). Deze functie wordt vaak in combinatie met XMLQuery() gebruikt, om van één leaf-node (een node die niet verder is onderverdeeld) of van een attribuut de waarde naar een bepaald datatype om te zetten.

De syntax van XMLCast() is als volgt:

```
XMLCAST(waarde_expressie AS datatype)
```

Hierbij zijn de volgende doel-datatypen mogelijk:

NUMBER, VARCHAR2, CHAR, CLOB, BLOB, REF XMLTYPE
en elk SQL date of time datatype.

De Oracle versie van XMLCast() wijkt af van de SQL/XML versie, in die zin dat alleen van XML naar een SQL datatype kan worden gecast, en niet van een SQL datatype naar XML of van XML naar XML.

In het volgende voorbeeld willen we met de functie LIKE zoeken naar werknemers waarvan de naam begint met een A.

```
select w.naam
from xml_kantoren_vw_3 x,
     xmltable('//werknemer' passing x.object_value
              columns naam xmltype path '/werknemer/naam') w
where xmlcast(naam as varchar2(30)) like 'A%'
```

```
NAAM
--------------------
<naam>ADELAAR</naam>
<naam>ALKEMA</naam>
<naam>APPEL</naam>
```

In dit voorbeeld zorgen we er eerst voor dat er steeds maar één leaf-node tegelijk wordt gecast. Dit gebeurt door de naam van elke werknemer, middels de XMLTable() functie, als een aparte rij te benaderen. Merk op dat elke naam die wordt opgehaald van het type XMLType is. We kunnen XMLCast() gebruiken om de text() node binnen naam om te zetten naar VARCHAR2.

De functie extractValue(), die voorheen wordt gebruikt om een text-waarde uit een XML-node op te halen, is vanaf Oracle 11g release 2 deprecated.

5.5 XMLIndex

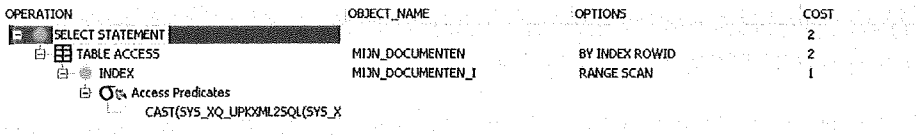
Om XML documenten snel te kunnen vinden, kunnen function based indexen worden gemaakt. Deze kunnen worden aangemaakt op basis van gegevens die uniek zijn per XML document. Een voorbeeld:

```
create index mijn_documenten_i on mijn_documenten
( xmlcast(xmlquery('/voorbeeld/@nr'
                   passing object_value returning content)
  as number)
);
```

Deze index kan bij de volgende query worden gebruikt:

```
select xmlquery('/voorbeeld' passing object_value returning content)
from mijn_documenten
where xmlcast(
      xmlquery('/voorbeeld/@nr' passing object_value returning content)
      as number) = 1;
```

In SQL developer is dit te controleren met F6 (Execute Explain Plan).



Veelal zijn we echter niet op zoek naar hele documenten, maar naar fragmenten daaruit, die we bijvoorbeeld via XMLTable() opdelen in virtuele rijen. In het geval van object-relationeel opgeslagen XML-gegevens kunnen we dan nog steeds gebruik maken van een normale B-Tree index index (zie paragraaf 2.3.1). Dit zijn gegevens die gekoppeld zijn aan een XML Schema. Voor het benaderen van (semi)-ongestructureerd opgeslagen XML gegevens kunnen we sinds Oracle 11g release 2 gebruik maken van XMLIndex. Met de introductie van XMLIndex is de function based index voor het benaderen van XML-fragmenten via een XPath pad verouderd (deprecated).

XMLIndex is speciaal bedoeld voor het benaderen XML gegevens. Gewone indexen worden alleen gebruikt als via de WHERE clause naar geïndexeerde gegevens wordt gezocht. Een XMLIndex wordt gebruikt om XML-fragmenten te benaderen via XMLQuery(), XMLTable(), XMLExists() en XMLCast().

Voor het volgende voorbeeld maken we eerst een tabel aan op basis van xml_kantoren_vw_3:

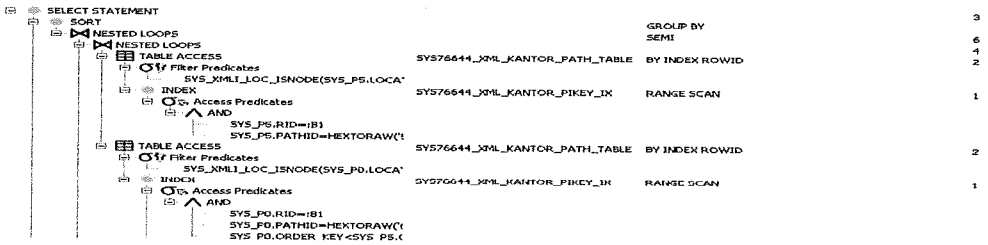
```
create table xml_kantoren of xmltype as select object_value from xml_kantoren_vw_3;
```

Nu maken we een index aan op deze tabel. Default indexeert XMLIndex alle mogelijk XPath paden binnen het XML document:

```
create index xml_kantoren_i on xml_kantoren(object_value)
indextype is xdb.xmlindex;
```

Nu bekijken we het execution plan van een XMLQuery() selectie op xml_kantoren (F6):

```
select xmlquery('//kantoor[@kantnr=10]/werknemer'
passing object_value returning content)
from xml_kantoren;
```



We zien hier onder meer dat delen van de query worden benaderd "BY INDEX ROWID", via een "PATH TABLE". Dit is een systeemtabel die wordt aangemaakt bij het creëren van een XMLIndex, waarin diverse mogelijke XPath paden worden opgeslagen. We kunnen de naam van die systeemtabel eventueel ook zelf specificeren, maar dit heeft weinig nut, omdat we deze systeemtabel niet zelf kunnen benaderen.

Een path table bevat onder meer een unieke identifier voor elk pad, een order key die de positie in de hiërarchie weergeeft, een rowid van de tabel die de xmlgegevens bevat, en een value. De value is de tekstwaarde van de leaf-node, of van het attribuut. Als het geïndexeerde pad zelf geen leaf-node is, dan is wordt de value samengesteld door alle text-nodes vanaf dat pad te concateneren. Samengestelde waarden groter dan VARCHAR2(4000) worden afgebroken.

Het begin van de tabel xml_kantoren zou de volgende path-id's kunnen opleveren in een XMLIndex.:

```
PATHID XPath
1      /kantoren
2      /kantoren/kantoor
3      /kantoren/kantoor/@kantnr
4      /kantoren/kantoor/naam
5      /kantoren/kantoor/plaats
```

In de path table is dit dan op de de volgende manier opgeslagen:

```
PATHID RID ORDER_KEY VALUE
1      R1  1         10BOEKHOUDINGAMSTERDAM.....
2      R1  1.1       10BOEKHOUDINGAMSTERDAM.....
3      R1  1.1.1     10
4      R1  1.1.2     BOEKHOUDING
5      R1  1.1.3     AMSTERDAM
```

Bovenstaand gebruik van XMLIndex, waarbij alle mogelijke XPath-paden in een path table zijn opgeslagen, wordt vooral gebruikt voor relatief ongestructureerde gegevens.
Als XML gegevens duidelijk gestructureerd zijn, zoals in het geval van de tabel xml_kantoren, kunnen we meer indexgegevens relationeel opslaan. We hebben eerder behandeld hoe wel met behulp van XMLTable() XML gegevens in stukken kunnen opdelen. Op vergelijkbare manier kunnen we een XMLIndex aanmaken op het gestructureerde deel van XML gegevens. We zien dat in het volgende voorbeeld:

```
BEGIN
DBMS_XMLINDEX.registerParameter(
'myparam',
'ADD_GROUP GROUP werknemer
XMLTable kantoor_idx_tab ''/kantoren/kantoor''
COLUMNS kantnr number PATH ''@kantnr'',
kantoornaam VARCHAR2(30) PATH ''naam'',
kantoorplaats VARCHAR2(30) PATH ''plaats'',
werknemers XMLType PATH ''werknemer'' VIRTUAL
XMLTable kantoor_idx_werknemer ''/werknemer'' PASSING werknemers
COLUMNS persnr number PATH ''@persnr'',
mgr number PATH ''@mgr'',
naam VARCHAR2(30) PATH ''naam'',
functie VARCHAR2(30) PATH ''functie'',
sal NUMBER PATH ''toeslag'',
toeslag NUMBER PATH ''toeslag''');
END;
/

ALTER INDEX xml_kantoren_i PARAMETERS('PARAM myparam');
```

Eerst registreren we in dit voorbeeld een XMLIndex-parameter met de naam "myparam". Met ADD_GROUP voegen we een groep toe aan de index. Een XMLIndex is een zogenaamde "domein-index": daarvan bestaat er hooguit één per kolom. In plaats van meerdere XMLIndexen aan te maken voor meerdere soorten queries, worden in dit geval meer groepen toegevoegd aan een bestaande index. Na het registreren van de parameter en het toekennen van de parameter aan de index xml_kantoren_i, zijn er twee extra path tables aangemaakt: kantoor_idx_tab en kantoor_idx_werknemer. Oracle kan vanaf dit moment kiezen of een query sneller kan worden uitgevoerd met behulp van de relationeel opgeslagen gegevens, of met behulp van de eerder aangemaakte path table.

We hebben hiermee een klein deel van de mogelijkheden van XMLIndex behandeld. Voor meer informatie verwijzen we naar de online documentatie.

5.6 Het XQuery commando

Vanaf Oracle 11g kunnen we in SQL*Plus en in SQL Developer gebruik maken van het XQuery commando. Deze begint met XQuery, en eindigt met een slash (/) op een nieuwe regel (geen puntkomma) Bijvoorbeeld:

```
xquery for $i in (1 to 3) return $i
/

COLUMN_VALUE
--------------
1
2
3
```

In SQL*Plus krijgt de uitvoer de kop "Result Sequence".

Tabellen kunnen worden geraadpleegd via ora:view. Eerder hebben we het volgende voorbeeld gezien:

```
select xmlquery(
   '//werknemer'
   passing object_value
returning content) as werknemers
from xml_kantoren_vw_3;
```

Dit leverde één rij op, met daarin alle werknemers aan elkaar gekoppeld (probeer dit eventueel nogmaals uit)

Met behulp van het XQuery commando kunnen we dat vertalen naar het volgende statement:

```
xquery let $werknemers := ora:view("xml_kantoren_vw_3")//werknemer
return $werknemers
/

COLUMN_VALUE
--------------------------------------------------------------------
<werknemer persnr="5810" mgr="6221"><naam>HEUVEL</naam><sal>3450</sal
<werknemer persnr="6221"><naam>KRAAY</naam><sal>6000</sal><functie>DI
<werknemer persnr="8222" mgr="5810"><naam>MANDERS</naam><sal>2350</sa
<werknemer persnr="9999" mgr="5810"><naam>WIERINGA</naam><sal>2000</s
<werknemer persnr="3381" mgr="7902"><naam>SMITS</naam><sal>2400</sal>
<werknemer persnr="3930" mgr="6221"><naam>PIETERS</naam><sal>3975</sa
<werknemer persnr="5931" mgr="3930"><naam>SANDERS</naam><sal>4000</sa
<werknemer persnr="6681" mgr="5931"><naam>ADELAAR</naam><sal>2100</sa
<werknemer persnr="7902" mgr="3930"><naam>VERMEULEN</naam><sal>3900</
<werknemer persnr="3462" mgr="4621"><naam>ALKEMA</naam><sal>2600</sal
<werknemer persnr="3518" mgr="4621"><naam>WALSTRA</naam><sal>2250</sa
<werknemer persnr="4510" mgr="4621"><naam>VERGEER</naam><sal>2250</sa
<werknemer persnr="4621" mgr="6221"><naam>KLAASEN</naam><sal>3850</sa
<werknemer persnr="6500" mgr="4621"><naam>DROST</naam><sal>2500</sal>
<werknemer persnr="7900" mgr="4621"><naam>APPEL</naam><sal>1950</sal>
```

Merk op dat in dit geval wel losse rijen worden gegenereerd. In SQLDeveloper kunnen we onder de motorkap zien hoe dat komt, als we expres een fout in de syntax maken. Haal daarvoor de dubbele punt weg en voer het nogmaals uit. We zien dan de volgende foutmelding verschijnen:

```
Error starting at line 1 in command:
select column_value from xmltable('let $werknemers =
ora:view("xml_kantoren_vw_3")//werknemer
return $werknemers')
Error at Command Line:1 Column:8
Error report:
SQL Error: ORA-19114: XPST0003 - error during parsing the XQuery expression:
LPX-00801: XQuery syntax error at '='
1   let $werknemers = ora:view("xml_kantoren_vw_3")//werknemer
```

Met andere woorden: een xquery commando is een verkapt select statement met gebruik van een XMLTable() functie.

6 De XMLDB repository

6.1 Inleiding

De XMLDB repository beheert de bronnen die aan XMLDB worden toegevoegd. Met behulp van de datadictionary views PATH_VIEW en RESOURCE_VIEW kunnen die bronnen in SQL worden benaderd. Bronnen kunnen niet alleen uit de repository worden opgehaald, het is ook mogelijk ze aan te passen of te verwijderen. We kunnen zelfs handmatig nieuwe bronnen aan de repository toevoegen.
Dit hoofdstuk behandelt hoe de datadictionary views PATH_VIEW en RESOURCE_VIEW kunnen worden gebruikt om de XMLDB repository te beheren. Daarnaast komen enkele functies aan de orde die het werken met deze datadictionary views vergemakkelijken.

6.2 RESOURCE_VIEW en PATH_VIEW

De datadictionary views RESOURCE_VIEW en PATH_VIEW lijken op elkaar.

RESOURCE_VIEW bevat een rij per bron in de XMLDB repository:

kolom	datatype	omschrijving
RES	XMLType	Een bron in de repository
ANY_PATH	VARCHAR2	Een pad dat verwijst naar de bron
RESID	RAW	Het id van de bron

PATH_VIEW bevat een rij voor elk pad naar een bron in de repository.

kolom	datatype	omschrijving
PATH	VARCHAR2	Een pad dat verwijst naar de bron
RES	XMLType	Een bron in de repository
LINK	XMLType	Een link naar de bron
RESID	RAW	Het id van de bron

Er kunnen meerdere paden naar dezelfde bron bestaan, omdat het mogelijk is om links te maken naar een bron. Een link in XMLDB is vergelijkbaar met een hard link in Unix: nadat de link is aangemaakt bestaan twee gelijkwaardige paden naar dezelfde bron. We zullen dat nu proberen voor het bestand voorbeeld.xml.

We maken eerst met de functie createfolder uit de package dbms_xdb een nieuwe folder aan om een link in te plaatsen. We doen dit hier in een if statement, omdat deze functie een boolean retourneert:

```
begin
  if dbms_xdb.createfolder('/home/cursisten/<gebruikersnaam>/links') then null;
  end if;
end;
```

Leg de wijziging vast met commit (F11).

Vervolgens kunnen we een link aan maken in de nieuwe folder links. Dit doen we met de procedure link uit dezelfde package dbms_xdb. Aan deze procedure geven we drie parameterwaarden mee: het pad naar de bron, het pad waar de link moet komen te staan, en de naam van de link.

```
call dbms_xdb.link('/home/cursisten/<gebruikersnaam>/voorbeeld.xml' ,
    '/home/cursisten/<gebruikersnaam>/links','voorbeeld.xml');
```

Sla ook deze wijziging weer op met commet (F11).

Nu bestaan twee paden naar de bron voorbeeld.xml. De paden kunnen worden gevonden in path_view.

```
select path from path_view
where under_path(res,'/home/cursisten/<gebruikersnaam>',1)=1
and path(1) like '%voorbeeld.xml';

PATH
--------------------------------------------------------------
/home/cursisten/nwg802/voorbeeld.xml
/home/cursisten/nwg802/links/voorbeeld.xml
```

Vrij vertaald wordt hier gevraagd om alle paden onder /home/cursisten/<gebruikersnaam> te tonen, waarvan het pad eindigt op voorbeeld.xml. De functies path() en under_path() zullen in een volgende paragraaf nader worden besproken.

Een soortgelijke query kan ook op resource_view worden uitgevoerd. Hierin wordt slechts één van de alternatieve paden naar de bron getoond:

```
select any_path from resource_view
where under_path(res,'/home/cursisten/<gebruikersnaam>',1)=1
and path(1) like '%voorbeeld.xml';

ANY_PATH
--------------------------------------------------------------
/home/cursisten/nwg802/links/voorbeeld.xml
```

Hieronder zullen de kolommen van resource_view en path_view nader worden besproken.

6.2.1 PATH en ANY_PATH

Path uit path_view en any_path uit resource_view representeren virtuele paden naar bronnen in de XMLDB repository. Een pad is een hiërarchische naam bestaande uit een root element (de eerste /), scheidingstekens tussen subpaden /, en diverse padelementen. Een padelement kan bestaan uit karakters in de database character set, behalve \ and /, die een speciale betekenis hebben in XMLDB. De forward slash is de default separator in een pad naam, en de backward slash wordt gebruikt als escape karakter om speciale karakters te genereren.

6.2.2 RESID

De kolom RESID is een raw kolom die het unieke id van de betreffende resource bevat. Dit id wordt vooral gebruikt voor versioning met behulp van de package dbms_xdb_version. Het kan ook gebruikt worden in joins. In het volgende voorbeeld worden alle paden opgehaald van bronnen waarvoor meerdere paden bestaan:

```
select a.path from path_view a, path_view b
    where a.path!=b.path
    and a.resid=b.resid;


PATH
--------------------------------------------------
/home/cursisten/nwg802/voorbeeld.xml
/home/cursisten/nwg802/links/voorbeeld.xml
```

6.2.3 LINK

De link kolom van path_view is van het type XMLType en bevat informatie over de positie van het pad in de hiërarchie. Link bevat onder meer de elementen ChildName en ParentName, waarvan de ChildName het laagste niveau van het betreffende pad betreft, en ParentName het daarboven liggende niveau.
Links naar bronnen in de repository kunnen harde of zachte links zijn. Beide soorten links zijn referenties naar fysieke gegevens. Ze verwijzen niet naar paden of namen, maar wel naar de resource identifiers (RESID).
Dit zorgt ervoor dat een link geldig blijft, ook als het pad naar de bron gewijzigd wordt.
Het verschil tussen harde en zachte links heeft te maken met het verwijderen van bronnen. Zolang er harde links naar een bron bestaan, blijft de bron zelf ook bestaan. Alleen als de laatste harde link wordt verwijderd, verdwijnt ook de bron.
Het is wel mogelijk een bron te verwijderen waar nog zachte links naar bestaan. De werking is dan andersom: bij het verwijderen van de bron, worden automatisch ook alle zachte links verwijderd.
Het link type is terug te vinden in het element LinkType. Deze kan de waarde Weak of Strong bevatten.

6.2.4 RES

De bron zelf wordt met de kolom RES in path_view en in resource_view aangeduid. Dit is een XMLType kolom, met als root element Resource. Dit element bevat onder meer de volgende attributen en elementen (attributen aangeduid met @ prefix).

naam	datatype	omschrijving
@Container	boolean	Of deze resource een folder is of niet
CreationDate	dateTime	Moment dat resource is aangemaakt
ModificationDate	dateTime	Laatste keer dat bron is aangepast
Creator	string	De Oracle gebruiker die de bron heeft aangemaakt
DisplayName	string	Naam van de bron
Comment	string	Commentaar
RefCount	integer	Aantal paden dat naar deze bron bestaat. Voor bronnen waarvan de RefCount groter dan 1 is, zijn dus links aangemaakt.
ACL	XML	verleende privileges over deze bron
Contents		Inhoud van de bron (datatype staat open)

Zo kunnen we met behulp van RES bijvoorbeeld alleen alle eigen paden ophalen die geen folder zijn:

```
select path from path_view
where xmlexists('declare namespace ns =
"http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
   /ns:Resource[@Container = xs:boolean("false")][ns:Creator="NWG802"]'
passing res)

PATH
------------------------------------
/home/cursisten/nwg802/voorbeeld.xml
/home/cursisten/nwg802/links/voorbeeld.xml
/home/cursisten/nwg802/ziekenhuizen/afdelingen.xml
/home/cursisten/nwg802/ziekenhuizen/bezetting.xml
/home/cursisten/nwg802/ziekenhuizen/patienten.xml
/home/cursisten/nwg802/ziekenhuizen/stafleden.xml
/home/cursisten/nwg802/ziekenhuizen/ziekenhuizen.xml
```

In dit voorbeeld zien we een niet eerder getoond gebruik van XQuery: het gebruik van namespaces. Dit is nodig, omdat het Resource element, waar we informatie over ophalen, behoort tot de namespace "http://xmlns.oracle.com/xdb/XDBResource.xsd".

Voor dit soort gevallen wordt een variabele van het type namespace aangemaakt, die verderop in de XPath expressie gebruikt wordt. Alles tussen (: en :) wordt in XQuery als commentaar gezien. In dit geval is dat toegevoegd, omdat in SQL*Plus anders de puntkomma achter de gedeclareerde namespace wordt gezien als een einde van het statement.

Het effect van deze query is dat binnen res gezocht wordt naar Resource elementen, waarvan het Container attribuut false is, en het Creator element gelijk is aan de gebruikersnaam (= het schemanaam, in hoofdletters).

6.3 Pad functies

Om paden en bronnen te selecteren zijn enkele functies beschikbaar: under_path, equals_path, level en path. De functies path en level zijn hulpfuncties, die met een correlatie index verwijzen naar de functies equals_path of under_path elders in het statement. Zo'n correlatie index is vergelijkbaar met een tabel alias, maar dan niet voor een tabel maar een functie. We komen hier in bij de betreffende functies op terug.

6.3.1 UNDER_PATH

De functie under_path gebruikt de hiërarchie van de XMLDB repository om snel paden onder een opgegeven pad te vinden.

syntax: `INTEGER UNDER_PATH(resource_kolom, padnaam);`

De functie retourneert de waarde 1 als er paden voorkomen onder de opgegeven padnaam. Deze functie wordt meestal in de where clause van een SQL-statement gebruikt.

Bijvoorbeeld:

```
select path
from path_view
where under_path(res, '/home/cursisten/<gebruikersnaam>/x')=1;

PATH
--------------------------------------------------
/home/cursisten/nwg802/x/y
/home/cursisten/nwg802/x/y/voorbeeld.xml
```

We kunnen ook de maximale diepte van het pad onder het opgegeven pad specificeren. Deze parameter staat tussen de resource kolom en het pad in.

In dit voorbeeld zoeken we alleen de paden hooguit 1 niveau onder het opgegeven pad:

```
select path
from path_view
where under_path(res, 1, '/home/cursisten/<gebruikersnaam>')=1;

PATH
--------------------------------------------------------
/home/cursisten/nwg802/links
/home/cursisten/nwg802/voorbeeld.xml
/home/cursisten/nwg802/x
```

Het is ook mogelijk om een correlatie index mee te geven, voor het gebruik van deze functie in combinatie met de hulpfuncties depth en path. Deze variant wordt bij de betreffende hulpfuncties behandeld.

6.3.2 EQUALS_PATH

De functie equals_path lijkt op under_path, maar wordt gebruikt om naar exacte paden te zoeken.

Voorbeeld:

```
select path
from path_view
where equals_path(res,'/home/cursisten/<gebruikersnaam>/voorbeeld.xml')=1;

PATH
--------------------------------------------------------------------------------
/home/cursisten/nwg802/voorbeeld.xml
```

Het is ook mogelijk om te zoeken met
`...where path = '/home/cursisten/<gebruikersnaam>/voorbeeld.xml'`
Dit gebruik wordt echter afgeraden, vooral in combinatie met resource_view en any_path. Dit omdat het letterlijke pad in resource_view maar één keer wordt opgeslagen. Met behulp van equals_path() kunnen alle links toch terug gevonden worden.
Voorbeeld:

```
select any_path
from resource_view
where equals_path(res,'/home/cursisten/<gebruikersnaam>/voorbeeld.xml')=1;

ANY_PATH
--------------------------------------------------------------------------------
/home/cursisten/nwg802/voorbeeld.xml

select any_path
from resource_view
where any_path = '/home/cursisten/<gebruikersnaam>/voorbeeld.xml';

ANY_PATH
--------------------------------------------------------------------------------

0 rows selected
```

6.3.3 DEPTH

De functie depth is een hulpfunctie, die in combinatie met under_path kan worden gebruikt om de diepte onder het opgegeven pad te bepalen. Depth en under_path kunnen vaker voorkomen in een statement. Om te bepalen welke aanroep van depth behoort bij welke aanroep van under_path, geven we in beide functies een correlation index mee. Het gebruik daarvan is vergelijkbaar met het gebruik van aliassen in een join: een tabel krijgt een alias, zodat verderop in het statement naar die alias kan worden verwezen. Zo krijgt under_path een extra integer waarde mee, zodat naar die waarde in depth kan worden verwezen.

In het volgende voorbeeld is die correlation index vet gedrukt. We zoeken naar de paden onder <gebruikersnaam> van twee niveaus diep:

```
select path
from path_view
where under_path(res,'/home/cursisten/<gebruikersnaam>',1)=1
and depth(1)=2;
```

```
PATH
------------------------------------------------------------
/home/cursisten/nwg802/links/x
/home/cursisten/nwg802/x/y
/home/cursisten/nwg802/ziekenhuizen/afdelingen.xml
/home/cursisten/nwg802/ziekenhuizen/bezetting.xml
/home/cursisten/nwg802/ziekenhuizen/patienten.xml
/home/cursisten/nwg802/ziekenhuizen/stafleden.xml
/home/cursisten/nwg802/ziekenhuizen/ziekenhuizen.xml
```

Let op: indien een pad verwijst naar een bron waarvoor meerdere paden bestaan, dan kiest Oracle de depth van één van de paden.

In het volgende voorbeeld halen we de paden en dieptes op van de bronnen waarnaar meerdere paden bestaan. De RefCount is dan groter dan 1. We gebruiken XMLCast om die waarde naar number om te zetten:

```
select depth(1)||' '||path
   from path_view
   where xmlcast(xmlquery(
   'declare namespace ns = "http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
   //ns:RefCount' passing res returning content ) as number)>1
   and under_path(res,'/home/cursisten/<gebruikersnaam>',1)=1
   order by path;

DEPTH(1)||''||PATH
------------------------------------------------------------
1 /home/cursisten/nwg802/links/voorbeeld.xml
3 /home/cursisten/nwg802/links/x/y/voorbeeld.xml
1 /home/cursisten/nwg802/voorbeeld.xml
3 /home/cursisten/nwg802/x/y/voorbeeld.xml
```

Bij combinaties van meerdere aanroepen van underpath en depth, worden ook meerdere correlation indexen gebruikt. In het volgende voorbeeld worden paden naar bronnen getoond van 1 niveau diep onder /home/cursisten/<gebruikersnaam>/x, en onder /home/cursisten/<gebruikersnaam>/links:

```
select path
from path_view
where under_path(res,'/home/cursisten/<gebruikersnaam>/x',1)=1 and depth(1)=1
or    under_path(res,'/home/cursisten/<gebruikersnaam>/links',2)=1
and depth(2)=1;

PATH
------------------------------
/home/cursisten/nwg802/x/y
/home/cursisten/nwg802/links/x
```

6.3.4 PATH

De functie path is vergelijkbaar met de functie depth: ook deze functie wordt in combinatie met under_path gebruikt met een correlation index. De functie retourneert het deel van het pad dat valt onder het pad dat aan under_path is meegegeven. Ook voor de hulpfunctie path geldt, dat Oracle één van de alternatieve paden naar een bron kiest.

In de volgende voorbeelden worden de paden getoond van maximaal twee niveaus diep (tweede argument van under_path) onder home/cursisten/<gebruikersnaam>.

```
select rpad(path,60)||path(1)
from path_view
where under_path(res,2,'/home/cursisten/<gebruikersnaam>',1)=1
order by path;
```

```
RPAD(PATH,60)||PATH(1)
------------------------------------------------------------------------------------
/home/cursisten/nwg802/links                              links
/home/cursisten/nwg802/links/voorbeeld.xml                voorbeeld.xml
/home/cursisten/nwg802/links/x                            links/x
/home/cursisten/nwg802/voorbeeld.xml                      voorbeeld.xml
/home/cursisten/nwg802/x                                  x
/home/cursisten/nwg802/x/y                                x/y
/home/cursisten/nwg802/ziekenhuizen                       ziekenhuizen
/home/cursisten/nwg802/ziekenhuizen/afdelingen.xml        ziekenhuizen/afdelingen.xml
/home/cursisten/nwg802/ziekenhuizen/bezetting.xml         ziekenhuizen/bezetting.xml
/home/cursisten/nwg802/ziekenhuizen/patienten.xml         ziekenhuizen/patienten.xml
/home/cursisten/nwg802/ziekenhuizen/stafleden.xml         ziekenhuizen/stafleden.xml
/home/cursisten/nwg802/ziekenhuizen/ziekenhuizen.xml      ziekenhuizen/ziekenhuizen.xml
```

6.4 Bronnen en paden bewerken

Bronnen en paden in resource_view en path_view kunnen worden bewerkt. Voor de meeste bewerkingen wordt aangeraden de package dbms_xdb te gebruiken. Dit wordt in de volgende paragraaf behandeld. Het is echter ook mogelijk om rechtstreeks bewerkingen uit te voeren.

6.4.1 Bronnen en paden verwijderen

De eenvoudigste bewerking is het verwijderen van paden en bronnen. Uit path_view kunnen losse paden worden verwijderd. Een delete statement op resource_view haalt alle paden naar de betreffende bron weg. In beide gevallen geldt dat alleen (paden naar) bronnen en lege folders kunnen worden verwijderd. Het verwijderen van folders met inhoud kan alleen met de dbms_xdb package.
In het volgende voorbeeld wordt een pad uit path_view verwijderd.

```
delete from path_view
where equals_path(res,'/home/cursisten/<gebruikersnaam>/links/voorbeeld.xml')=1;

1 row deleted.
```

Er bestaat nog een pad naar deze bron: alleen de RefCount van de bron is verlaagd en het pad is verwijderd:

```
select xmlcast(xmlquery('declare namespace
ns="http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
                        //ns:RefCount' passing res returning content)
                as number) as "RefCount"
from path_view
where equals_path(res,'/home/cursisten/nwg802/voorbeeld.xml')=1;

RefCount
-----------
1
```

In het volgende voorbeeld wordt de bron zelf wel verwijderd. In dit geval gaat het om /x/y/voorbeeld.xml, waarnaar ook de link /links/x/y/voorbeeld.xml bestaat:

```
delete from resource_view
where equals_path(res,'/home/cursisten/<gebruikersnaam>/x/y/voorbeeld.xml')=1;

1 row deleted.
```

De volgende folder is niet leeg, en kan dus niet zomaar verwijderd worden:

```
SQL> delete from path_view
  2  where equals_path(res,'/home/cursisten/<gebruikersnaam>/links')=1;
delete from path_view
*
Error report:
SQL Error: ORA-31007: Attempted to delete non-empty container
/home/cursisten/nwg802//links
ORA-06512: at "XDB.XDB_PVTRIG_PKG", line 13
ORA-06512: at "XDB.XDB_PV_TRIG", line 9
ORA-04088: error during execution of trigger 'XDB.XDB_PV_TRIG'
31007. 00000 -  "Attempted to delete non-empty container %s/%s"
*Cause:    An attempt was made to delete a non-empty container in
           the XDB hierarchical resolver.
*Action:   Either perform a recursive deletion, or first delete
           the contents of the container.
```

6.4.2 Bronnen aanpassen

Het is mogelijk bronnen geheel of gedeeltelijk met een update te wijzigen. In het volgende voorbeeld wordt de DisplayName en het pad van voorbeeld.xml aangepast naar voorbeelden.xml.

```
update resource_view r
set r.res=updatexml(r.res,'/Resource/DisplayName/text()','voorbeelden.xml'),
any_path='/home/cursisten/<gebruikersnaam>/voorbeelden.xml'
where equals_path(r.res,'/home/cursisten/<gebruikersnaam>/voorbeeld.xml')=1;

1 row updated.
```

6.4.3 Bronnen toevoegen

We kunnen ook nieuwe bronnen toevoegen of bronnen kopiëren met insert statements. Hierbij geven we meestal een deel van de bron zelf op, en laten we de rest genereren door de triggers die intern afgaan als gevolg van het insert statement. Voorbeeld:

```
INSERT INTO resource_view VALUES(
xmltype('
  <Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResource.xsd
http://xmlns.oracle.com/xdb/XDBResource.xsd">
  <Author>'||user||'</Author>
  <Comment>Dit is een nieuwe bron</Comment>
  <Language>nl</Language>
  <Contents>
  <nieuw xmlns="">Hoera, een nieuwe bron!</nieuw>
  </Contents>
  </Resource>'), '/home/cursisten/'||lower(user)||'/nieuw.xml', NULL)
/
commit;
```

Hierin wordt een deel van de bron, en het pad opgegeven. De rest van de bron en het resid zullen worden gegenereerd. Nu kunnen we bijvoorbeeld ook de CreationDate en de resid ophalen, ook al zijn die kenmerken niet opgegeven bij het aanmaken van de bron:

```
select resid, extractvalue(res,'/Resource/CreationDate') creationdate
from resource_view
where equals_path(res,'/home/cursisten/<gebruikersnaam>/nieuw.xml')=1;

RESID                            CREATIONDATE
-------------------------------- -------------------------
7992D2A49417B365E040007F010004F3 30-NOV-09 10.15.07.260614000
```

We kunnen ook bronnen kopiëren:

```
insert into path_view
(res, path)
select res,'/home/cursisten/'||lower(user)||'/x/y/nieuw.xml'
from path_view
where equals_path(res, '/home/cursisten/'||lower(user)||'/nieuw.xml')=1
/
commit;
```

Hiermee wordt een kopie gemaakt, geen link. De resid en link kolommen worden weer door triggers gegenereerd. Dit gebeurt ook als de kolommen resid en link bij het kopiëren worden meegeselecteerd.

6.5 Bronnen bewerken met dbms_xdb

We zijn al eerder enkele procedures en functies uit de dbms_xdb package tegengekomen. De volgende tabel geeft een overzicht van enkele procedures en functies die gebruikt kunnen worden bij het bewerken van bronnen en folders.

functie/procedure	returns	argumenten	omschrijving
functie CreateResource	true indien succesvol	(path IN VARCHAR2, data IN VARCHAR2)	Maakt een nieuwe bron met de opgegeven string als de inhoud.
		(path IN VARCHAR2, data IN SYS.XMLTYPE)	Maakt een nieuwe bron met de opgegeven XMLType data als de inhoud.
		(path IN VARCHAR2, datarow IN REF SYS.XMLTYPE)	Maakt een bron op basis van een REF naar een bestaande XMLType rij. De Contents van de nieuwe bron verwijst naar die rij. De betreffende rij mag niet tegelijk de basis van een andere bron vormen.
		(path IN VARCHAR2, data IN CLOB)	Maakt een bron met de opgegeven CLOB als inhoud.
		(abspath IN VARCHAR2, data IN BFILE, csid IN NUMBER := 0)	Maakt een XMLDB bron op basis van BFILE.
		(abspath IN VARCHAR2, data IN BLOB, csid IN NUMBER := 0)	Maakt een XMLDB bron op basis van BLOB.
functie CreateFolder	true indien succesvol	(path IN VARCHAR2) Return waarde: TRUE if successful.	Maakt een folder aan in het opgegeven pad. Dit lukt alleen indien de parent-folder al bestaat.
procedure Link	n.v.t.	(srcpath VARCHAR2, linkfolder VARCHAR2, linkname VARCHAR2, linktype BINARY_INTEGER)	Maakt een link aan naar de opgegeven bron. Standaard wordt het een harde link. We kunnen er een zachte link van maken, door als vierde argument DBMS_XDB.LINK_TYPE_WEAK mee te geven.

functie/procedure	returns	argumenten	omschrijving
procedure DeleteResource	n.v.t.	(path IN VARCHAR2, delete_option PLS_INTEGER) Mogelijke delete options: DELETE_RESOURCE (default): verwijdert alleen de bron. Dit lukt niet als het een folder betreft die niet leeg is. DELETE_RECURSIVE: Verwijdert de bron inclusief eventuele onderliggende mappen of bronnen DELETE_FORCE: Verwijdert de bron, zelfs als het object dat het bevat invalid is. DELETE_RECURSIVE_FORCE: Verwijdert de bron inclusief eventuele onderliggende mappen of bronnen, ook als de bron of de onderliggende bronnen invalid zijn.	Verwijdert de bron op het opgegeven pad. De delete options zijn package constanten. Bijvoorbeeld: dbms_xdb.DeleteResource(<pad>, dbms_xdb.delete_recursive);

We zullen het aanmaken van een resource op basis van een REF hier nader toelichten. Met behulp van de functie MAKE_REF kunnen we een REF maken naar een objecttabel of naar een objectview. Het object-id dient daarvoor gebaseerd te zijn op de primary key. Dit is het geval voor de view xml_kantoren_vw:

```
create view xml_kantoren_vw of xmltype
with object id (XMLCast(
                  XMLQuery('/kantoor/@kantnr'
                              PASSING OBJECT_VALUE RETURNING CONTENT)
               as INTEGER)) AS ....
```

In het script view_resource.sql maken we hier gebruik van. Eerst wordt een nieuwe folder kantoren aangemaakt met behulp van de methode dbms_xdb.createfolder. Vervolgens wordt voor elk kantoornummer in de tabel kantoren een referentie opgehaald naar het betreffende kantoor in xml_kantoren_vw. Die referentie wordt daarna gebruikt om een bron kantoor<kantnr>.xml aan te maken.

```
declare
   root varchar2(50):='/home/cursisten/'||lower(user);
   xmlref ref xmltype;
   result boolean;
begin
-- aanmaken nieuwe folder
      result := dbms_xdb.createfolder  (root||'/kantoren');

-- voor elk kantoornummer in de tabel kantoren ...
      for rij in (select kantnr from ox_kantoren) loop

-- haal een referentie op naar het betreffende kantoor in xml_kantoren_vw
        select make_ref(xml_kantoren_vw, rij.kantnr)
        into xmlref
        from dual;

-- maak een bron aan op basis van de opgehaalde referentie
        result := dbms_xdb.createresource(
                  root||'/kantoren/kantoor' || rij.kantnr || '.xml'
                  , xmlref, false);
      end loop;
      commit;
end;
/
```

De parameterwaarde false in de aanroep naar createResource geeft aan dat het geen "sticky" verwijzing moet worden. Als de bron later verwijderd wordt, zullen daardoor de onderliggende rijen in de tabel kantoren niet worden verwijderd. Let op: vanwege compatibiliteit met voorgaande versies is de default waarde true.

We kunnen deze bronnen met WebDAV of FTP niet wijzigen. Wel geven deze bronnen de actuele inhoud van de onderliggende tabellen weer.

6.6 XLink en XInclude

Vanaf Oracle 11g release 1 worden XLink en XInclude binnen Oracle ondersteund. Met XLink kunnen we links aanmaken van het ene naar het andere document, vergelijkbaar met links in HTML pagina's. De ondersteuning hiervoor is nog beperkt, zowel binnen Oracle als in internet browsers.
XInclude maakt het mogelijk om documenten modulair op te bouwen. Hierbij verwijzen we binnen een hoofddocument naar subdocumenten. Bij het opvragen van het hoofddocument kunnen we automatisch ook de subdocumenten ophalen. We zullen hiervan een voorbeeld bespreken.

Binnen boek.xml wordt verwezen naar hoofdstuk.xml en titel.txt op de volgende manier:

```
<boek xmlns:xi="http://www.w3.org/2001/XInclude">
  <titel><xi:include href="titel.txt" parse="text" /></titel>
  <xi:include href="hoofdstuk.xml" />
</boek>
```

De bron zelf moet geldige XML zijn. Het subdocument kan ook van het type text zijn. Binnen de standaard XInclude is het ook mogelijk om XML-fragmenten als subdocumenten op te nemen, die van zichzelf geen XML-document zijn (dus geen XML met één root element). Binnen Oracle wordt dit niet ondersteund.
Om het gehele document op te halen, gebruiken we de functie xdburitype().
Naast de uri, geven we met de code 1 aan dat we de XInclude elementen eerst openen, voordat we het restultaat tonen. Met de code 0 wordt alleen het hoofddocument getoond.

```
select xdburitype(any_path, '1').getxml() as boek
from resource_view
where equals_path(res, '/home/cursisten/<gebruikersnaam>/boek/boek.xml') =1

BOEK
--------------------------------------------
<boek xmlns:xi="http://www.w3.org/2001/XInclude">
<titel>Voorbeeld van XInclude</titel>
<hoofdstuk>
Dit is het eerste en enige hoofdstuk van dit boek.
Hieraan zijn enige regels toegevoegd, om toch nog de indruk te wekken dat het iets om
het lijf heeft.
Maar schijn bedriegt: inhoudelijk wordt hier niets meer vermeld, dan reeds bij de
eerste regel is prijsgegeven.
U kunt dus met een schoon geweten stoppen met lezen, zelfs halverwege zo u wilt,
bijvoorbeeld hier, ook al is deze zin al bijna af. Met excuses voor de verloren tijd,
mocht u toch nog niet zijn afgehaakt, ook omdat het einde tamelijk abrupt en
onbevredigend
</hoofdstuk>
</boek>
```

7 XML Schema

7.1 Inleiding

XML Schema maakt het mogelijk XML documenten te valideren. In XMLDB wordt een XML Schema ook gebruikt om XMLType tabellen en kolommen relationeel op te kunnen slaan. In dit hoofdstuk zullen we een XML Schema via WebDAV aan de repository toevoegen. Met behulp daarvan zullen we vervolgens enkele XMLType documenten valideren.

7.2 Wat is XML Schema?

De XML Schema Recommendation is gemaakt door het World Wide Web Consortium (W3C) om de inhoud en de structuur van XML documenten te beschrijven. Een XML Schema is zelf ook een XML Document. De noodzaak voor XML Schema is ontstaan uit de beperkingen van Document Type Definition (DTD), welke eerder gangbaar was om XML Documenten mee te valideren. DTD's beschrijven de elementen die in een XML document voor kunnen komen, wat de subelementen zijn van andere elementen en de volgorde van de elementen. Voordelen van XML Schema ten opzichte van DTD zijn onder meer:

- Een XML Schema is zelf een XML Document, en kan dus door dezelfde XML applicatie worden gelezen als de XML Documenten die het beschrijft. DTD heeft een andere structuur.
- Een DTD kent maar één datatype (karakter strings). XML Schema kent vele datatypen, en stelt de gebruiker in staat zelf typen te definiëren.
- In een XML Schema kan exact worden vastgelegd hoe vaak een subelement binnen een element minimaal en maximaal voor mag komen. Een DTD kent in dit verband alleen 0, 1 en meer dan 1.
- In een XML Schema kunnen minimale en maximale waarden voor een element of attribuut worden vastgelegd. In een DTD kunnen alleen constante waarden worden vastgelegd.

7.3 XML Schema in XMLDB

In XMLDB kunnen XML Schema's worden gebruikt als basis voor tabellen en objecttypen. Bij het registreren van een XML Schema worden die tabellen en objecttypen automatisch aangemaakt. XML Documenten die aan zo'n schema voldoen kunnen we vervolgens eenvoudig in deze tabellen opslaan.

7.3.1 XML Schema registreren

We zullen in dit hoofdstuk het schema bestelling.xsd in de database registreren. Vereenvoudigd ziet dit schema er als volgt uit:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    version="1.0">

  <xsd:element name="bestelling" type="BestellingType"/>

  <xsd:complexType name="BestellingType" >
    <xsd:sequence>
      <xsd:element name="verzendNaar" type="AdresType" />
      <xsd:element name="rekeningNaar" type="AdresType"/>
      <xsd:element name="items" type="ItemType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="AdresType" >
    <xsd:sequence>
      <xsd:element name="naam" type="xsd:string" />
      <xsd:element name="straat" type="xsd:string" />
      <xsd:element name="postcode" type="xsd:string" />
      <xsd:element name="plaats" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ItemType" >
    <xsd:sequence>
      <xsd:element name="item" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="naam" type="xsd:string" />
            <xsd:element name="hoeveelheid" type="xsd:positiveInteger"/>
            <xsd:element name="prijs" type="xsd:decimal" />
            <xsd:element name="verzendDatum" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Een bestelling bevat dus onder meer twee adressen (verzendNaar en rekeningNaar) en een verzameling items. Het schema beschrijft ook de structuur van een adres en van een item. We kunnen dit schema registreren in de database.

7.3.1.1 Schema met default xdb-waarden

Het registreren van een schema doen we met behulp van de PL/SQL package dbms_xmlschema. Deze package heeft diverse procedures registerSchema(), die van elkaar verschillen door de manier waarop het schema als parameter wordt meegegeven. We zullen eerst laten zien hoe het schema als string kan worden doorgegeven:

```
declare
doc varchar2(3000):='<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    version="1.0">
<!-- hier staat de inhoud van het schema -->
 </xsd:schema>';
begin
  dbms_xmlschema.registerSchema('http://www.5hart.com/schema/bestelling.xsd', doc,
  enablehierarchy => dbms_xmlschema.ENABLE_HIERARCHY_NONE);
end;
/
```

We kunnen in de datadictionary view user_xml_schemas controleren of het registreren gelukt is.

```
select schema_url from user_xml_schemas;

SCHEMA_URL
------------------------------------------------------------
http://www.5hart.com/schema/bestelling.xsd
```

Het schema bestelling.xsd is geregistreerd op de locatie http://www.5hart.com/schema/bestelling.xsd. Hoewel het niet noodzakelijk is, is het wel gebruikelijk een URL op te geven als locatie. XMLDB gebruik deze URL als verwijzing naar het in de repository opgeslagen schema. Het is dus een soort link naar een interne representatie van het schema, en geen verwijzing naar een bestand op de webserver.

Het schema zelf staat ook in user_xml_schemas. XMLDB heeft aan het schema diverse eigen tags toegevoegd. We halen nu uit dit schema het element "bestelling" op, om te kijken hoe dat in de repository is opgeslagen. We gebruiken daarvoor de xmlquery() functie. Omdat "xsd:element" tot de namespace `http://www.w3.org/2001/XMLSchema` behoort, declareren we ook een variabele van het type namespace.:

```
select xmlquery('declare namespace xsd="http://www.w3.org/2001/XMLSchema"; (: :)
                //xsd:element[@name="bestelling"]' passing schema returning content)
                as bestelling
from user_xml_schemas
where schema_url like '%bestelling%';


BESTELLING
--------------------------------------------------------------------------------
<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="bestelling" type="BestellingType"
xmlns:oraxdb="http://xmlns.oracle.com/xdb" oraxdb:propNumber="4232" oraxdb:global="true"
oraxdb:SQLName="bestelling" oraxdb:SQLType="BestellingType659_T" oraxdb:SQLSchema="NWG802"
oraxdb:memType="258" oraxdb:defaultTable="bestelling660_TAB" oraxdb:defaultTableSchema="NWG802"
xmlns:oraxdb2="http://xmlns.oracle.com/xdb" oraxdb2:system="false"
xmlns:oraxdb3="http://xmlns.oracle.com/xdb" oraxdb3:mutable="false"
xmlns:oraxdb4="http://xmlns.oracle.com/xdb" oraxdb4:hidden="false"
xmlns:oraxdb5="http://xmlns.oracle.com/xdb" oraxdb5:baseProp="false" nillable="false"
abstract="false" xmlns:oraxdb6="http://xmlns.oracle.com/xdb" oraxdb6:SQLInline="false"
xmlns:oraxdb7="http://xmlns.oracle.com/xdb" oraxdb7:JavaInline="false"
xmlns:oraxdb8="http://xmlns.oracle.com/xdb" oraxdb8:MemInline="false"
xmlns:oraxdb9="http://xmlns.oracle.com/xdb" oraxdb9:maintainDOM="true" minOccurs="0"
xmlns:oraxdb10="http://xmlns.oracle.com/xdb" oraxdb10:isFolder="false"
xmlns:oraxdb11="http://xmlns.oracle.com/xdb" oraxdb11:maintainOrder="true"/>
```

Omdat dit element gedefinieerd is als xsd:element, behoren alle attributen zonder voorvoegsel, zoals name, nillable, en minOccurs ook tot dezelfde namespace `http://www.w3.org/2001/XMLSchema`. We zien ook dat diverse attributen in de namespace "`http://xmlns.oracle.com/xdb`" zijn toegevoegd. Hiermee wordt onder meer geregeld hoe het XML Schema in de database wordt opgeslagen. Zo blijkt in dit voorbeeld bijvoorbeeld dat een default tabel "bestelling660_TAB" is aangemaakt, van het type "BestellingType659_T".

De default tabel wordt gebruikt om XML Documenten in op te slaan die voldoen aan dit schema en die via WebDAV of FTP aan XMLDB worden aangeboden.

```
desc "bestelling660_TAB"

Name                                                      Null?    Type
--------------------------------------------------------- -------- ------------------
SYS_NC_ROWINFO$                                                     XMLTYPE()

desc "BestellingType659_T"

user type definition
-------------------------------------------------------------------------------
TYPE   "BestellingType659_T" AS OBJECT ("SYS_XDBPD$" "XDB"."XDB$RAW_LIST_T","datum"
VARCHAR2(4000 CHAR),"verzendNaar" "AdresType658_T","rekeningNaar"
"AdresType658_T","commentaar" VARCHAR2(4000 CHAR),"items" "ItemType654_T") NOT FINAL
INSTANTIABLE
```

Standaard worden SQL namen en typen afgeleid van de namen van de elementen. Deze namen zijn hoofdlettergevoelig. Dat betekent dat ze in SQL statements tussen dubbele quotes moeten worden gezet. Gelukkig kunnen we in een schema deze default waarden ook overschrijven. Dit zullen we in de volgende paragraaf behandelen.

We zullen nu het geregistreerde schema weer verwijderen. Daartoe moeten ook alle gegenereerde tabellen en kolommen worden verwijderd. Dit kan in één statement in PL/SQL met behulp van de optie DELETE_CASCADE van deleteSchema:

```
BEGIN
  DBMS_XMLSCHEMA.deleteSchema(
  SCHEMAURL => 'http://www.5hart.com/schema/bestelling.xsd',
  DELETE_OPTION => dbms_xmlschema.DELETE_CASCADE);
END;
```

7.3.1.2 Schema met overschreven xdb-waarden

We zullen nu een ander schema registreren, waarvan de defaultTable is gespecificeerd. Ook de SQLName en SQLType van de elementen overschrijven we in dit schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xdb="http://xmlns.oracle.com/xdb"
    version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      <author>
        <firstname>Marko</firstname>
        <lastname>Draisma</lastname>
      </author>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="bestelling" type="BestellingType"
               xdb:defaultTable="BESTELLINGEN"/>
  <xsd:element name="commentaar" type="xsd:string"
               xdb:SQLName="COMMENTAAR" xdb:defaultTable="" xdb:SQLType="VARCHAR2"/>
```

```
  <xsd:complexType name="BestellingType" xdb:SQLType="BESTELLING_TYPE">
    <xsd:sequence>
      <xsd:element name="verzendNaar" type="AdresType" minOccurs="1"
                   xdb:SQLName="VERZEND_NAAR"/>
      <xsd:element name="rekeningNaar" type="AdresType" minOccurs="0"
                   xdb:SQLName="REKENING_NAAR"/>
      <xsd:element ref="commentaar" minOccurs="0"
                   xdb:SQLName="COMMENTAAR"/>
      <xsd:element name="items" type="ItemType"
                   xdb:SQLName="ITEMS"/>
    </xsd:sequence>
    <xsd:attribute name="datum" type="stringKort"
                   xdb:SQLName="DATUM" xdb:SQLType="VARCHAR2"/>
    <xsd:attribute name="id" type="stringKort" xdb:SQLName="ID"
                   xdb:SQLType="VARCHAR2"/>
  </xsd:complexType>
  <xsd:complexType name="AdresType" xdb:SQLType="ADRES_TYPE">
     <xsd:sequence>
      <xsd:element name="naam" type="stringKort"
                   xdb:SQLName="NAAM" xdb:SQLType="VARCHAR2"/>
      <xsd:element name="straat" type="stringKort"
                   xdb:SQLName="STRAAT" xdb:SQLType="VARCHAR2"/>
      <xsd:element name="postcode" type="stringKort"
                   xdb:SQLName="POSTCODE" xdb:SQLType="VARCHAR2"/>
      <xsd:element name="plaats" type="stringKort"
                   xdb:SQLName="PLAATS" xdb:SQLType="VARCHAR2"/>
    </xsd:sequence>
  </xsd:complexType>
 <xsd:complexType name="ItemType" xdb:SQLType="ITEM_TYPE">
    <xsd:sequence>
      <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="naam" type="stringKort"
                         xdb:SQLName="NAAM" xdb:SQLType="VARCHAR2"/>
            <xsd:element name="hoeveelheid"
                         xdb:SQLName="HOEVEELHEID" xdb:SQLType="INTEGER">
              <xsd:simpleType>
                <xsd:restriction base="xsd:positiveInteger">
                  <xsd:maxExclusive value="100"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
            <xsd:element name="prijs" type="xsd:decimal"
                         xdb:SQLName="PRIJS" xdb:SQLType="NUMBER"/>
            <xsd:element ref="commentaar" minOccurs="0"/>
            <xsd:element name="verzendDatum" type="stringKort" minOccurs="0"
                         xdb:SQLName="VERZEND_DATUM"  xdb:SQLType="VARCHAR2"/>
          </xsd:sequence>
          <xsd:attribute name="code" type="codeType" use="required"
                         xdb:SQLName="CODE" xdb:SQLType="VARCHAR2"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="codeType" >
    <xsd:restriction base="stringKort">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
  </xsd:simpleType>
   <xsd:simpleType name="stringKort" >
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="0"/>
       <xsd:maxLength value="50"/>
     </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

In dit schema zijn diverse SQLType en SQLName attributen overschreven. Door ze allemaal upper case te maken kunnen ze in SQL ook zonder dubbele quotes worden benaderd. Verder zorgen we er in dit schema voor dat op basis van het element bestellingen een tabel BESTELLINGEN wordt aangemaakt.

Sla dit bestand met Save to FTP or WebDAV Server op. We zullen dit schema nu registreren. We gebruiken nu niet registerSchema, maar registerUri, zodat we kunnen verwijzen naar het bestand dat we aan de repository hebben toegevoegd.

```
begin
dbms_xmlschema.registeruri('http://www.5hart.com/schema/bestelling.xsd',
  '/home/cursisten/<gebruikersnaam>/bestellingen/bestelling.xsd',
    enablehierarchy => dbms_xmlschema.ENABLE_HIERARCHY_NONE);
end;
/
```

Nu is de tabel bestellingen aangemaakt:

```
desc bestellingen
```

```
desc bestellingen
Name                          Null     Type
----------------------------- -------- ------------------------------------
SYS_NC_ROWINFO$                        XMLTYPE()
```

We kunnen dit ook controleren in de datadictionary view USER_XML_TABLES:

```
select xmlschema, element_name
from user_xml_tables
where table_name='BESTELLINGEN';
```

```
XMLSCHEMA                                              ELEMENT_NAME
-------------------------------------------------------------------
http://www.5hart.com/schema/bestelling.xsd             bestelling
```

Het bestand bestelling1.xml voldoet aan het geregistreerde schema:

```
<?xml version="1.0"?>
<bestelling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.5hart.com/schema/bestelling.xsd"
datum="01-01-2004">
  <verzendNaar>
    <naam>P. Hendrix</naam>
    <straat>Smitstraat 4a</straat>
    <postcode>2435 XA</postcode>
    <plaats>Ten Burg</plaats>
  </verzendNaar>
  <items>
    <item code="123-DF">
      <naam>MP3 speler</naam>
      <hoeveelheid>1</hoeveelheid>
      <prijs>50.00</prijs>
      <verzendDatum>02-01-2004</verzendDatum>
    </item>
  </items>
</bestelling>
```

Het attribuut noNamespaceSchemaLocation uit de namespace http://www.w3.org/2001/XMLSchema-instance, in dit voorbeeld afgekort tot xsi, koppelt het geregistreerde schema aan deze XML instantie (bestelling). De attributen **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"** en **xsi:noNamespaceSchemaLocation=<schemalocatie>** dienen in het root element te worden opgenomen van het XML document dat aan het schema wordt gekoppeld.

Indien het XML document zelf ook tot een namespace behoort dan ziet het er anders uit. Stel bijvoorbeeld dat bestellingen dienen te behoren tot de namespace http://www.5hart.com/xml, dan had het er als volgt uit gezien:

```
<vh:bestelling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:vh="http://www.5hart.com/xml" xsi:schemaLocation=" http://www.5hart.com/xml
http://www.5hart.com/schema/bestelling.xsd">
...
</vh:bestelling>
```

We openen een XML bestand dat aan dit schema voeldoet en bewaren het in XMLDB. We kunnen daarna in SQL controleren of dit bestand ook aan de tabel BESTELLINGEN is toegevoegd.

```
select xmlquery('for $bestelling in /*
                 return $bestelling' passing object_value returning content)
as bestellingen
from bestellingen;


BESTELLINGEN
--------------------------------------------------------------------------------
<bestelling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.5hart.com/schema/bestelling.xsd" datum="01-
01-2004">
  <verzendNaar>
    <naam>P. Hendrix</naam>
    <straat>Smitstraat 4a</straat>
    <postcode>2435 XA</postcode>
    <plaats>Ten Burg</plaats>
  </verzendNaar>
  <items>
    <item code="123-DF">
      <naam>MP3 speler</naam>
      <hoeveelheid>1</hoeveelheid>
      <prijs>50</prijs>
      <verzendDatum>02-01-2004</verzendDatum>
    </item>
  </items>
</bestelling>
```

Nu is het nog maar één bestelling. Zodra het er meer worden is het handig om naar de inhoud van bestellingen te kunnen zoeken via het pad in XMLDB. In de repository wordt voor elke bestelling een referentie vastgelegd naar de betreffende rij in bestellingen.
Door een join te maken van resource_view en bestellingen kan de inhoud van een bestelling op een specifieke locatie in XMLDB worden opgezocht.

Bijvoorbeeld:

```
select xmlquery('for $bestelling in /*
   return $bestelling' passing b.object_value returning content) as bestellingen
from resource_view r, bestellingen b
where xmlcast(xmlquery('declare default element namespace
"http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
fn:data(/Resource/XMLRef)' passing r.res returning content) as ref xmltype)=ref(b)
and equals_path(res,'/home/cursisten/&1/bestellingen/bestelling1.xml')=1


BESTELLINGEN
--------------------------------------------------------------------------------
<bestelling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.5hart.com/schema/bestelling.xsd" datum="01-
01-2004">
  <verzendNaar>
    <naam>P. Hendrix</naam>
    <straat>Smitstraat 4a</straat>
    <postcode>2435 XA</postcode>
    <plaats>Ten Burg</plaats>
  </verzendNaar>
  <items>
    <item code="123-DF">
      <naam>MP3 speler</naam>
      <hoeveelheid>1</hoeveelheid>
      <prijs>50</prijs>
      <verzendDatum>02-01-2004</verzendDatum>
    </item>
  </items>
</bestelling>
```

Deze query behoeft wat extra uitleg.
Om te beginnen maken we hier gebruik van een default element namespace. Hiermee geven we aan dat zowel Resource als XMLRef tot de namespace http://xmlns.oracle.com/xdb/XDBResource.xsd behoren.
Verder wordt gebruik gemaakt van de XQuery functie fn:data(): hiermee halen we de "typed atomic value" op. Dit betekent dat niet het hele element, maar de tekst-waarde wordt opgehaald. Als het element volgens een XMLSchema een bepaald datatype heeft, dan wordt die waarde ook in dat datatype opgehaald. In dit geval hangt er een referentie-datatype aan het XMLRef element. De functie XMLCast() kan dat type vertalen naar het SQL datatype REF XMLType.
Het effect van deze query is dat we de bestelling in de tabel bestellingen vinden, die hoort bij het pad /home/cursisten/<gebruikersnaam>/bestellingen/bestelling1.xml in resource_view.

We maken van deze selectie een view aan:

```
create or replace view bestelling1 as
select b.object_value
  from resource_view r, bestellingen b
  where xmlcast(xmlquery('declare default element namespace
"http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
  fn:data(/Resource/XMLRef)' passing r.res returning content) as ref xmltype)=ref(b)
  and equals_path(res,'/home/medewerkers/&1/bestellingen/bestelling1.xml')=1
```

7.4 Valideren met XML Schema

Bij tabellen gebaseerd op een XML Schema wordt standaard alleen gecontroleerd of een nieuwe rij "past" in de object-relationele tabellen en typen die bij het schema horen. Dit betekent dat zolang de elementnamen en de SQL datatypen kloppen, we ook incorrecte

XML instanties aan de tabel kunnen toevoegen. Deze beperkte schema validatie gaat uit van de veronderstelling dat veel XML documenten al met een externe tool (bijvoorbeeld XMLBlueprint) gevalideerd zullen zijn voordat ze in de database worden vastgelegd.

Om met een XML Schema een XMLType document te valideren, zijn enkele functies en procedures beschikbaar. We zullen deze hier kort bespreken. Van tevoren zullen we bewust bestelling1.xml wijzigen, zodat deze niet meer voldoet aan het schema. Verander in XMLBLueprint van het item met de code 123-DF de hoeveelheid in 1000. Leg deze wijziging vast. Oracle voert daardoor een update uit in de tabel bestellingen.

```
select xmlquery('fn:data(//hoeveelheid)'
       passing object_value returning content) as hoeveelheid
from bestelling1;

HOEVEELHEID
------------
1000
```

7.4.1 XMLIsValid()

Met behulp van XMLIsValid() kan eenvoudig worden gecontroleerd of een XMLType document voldoet aan het schema. Deze functie geeft de waarde 1 terug als het een geldig document is. In alle andere gevallen, dus als het document niet geldig is, of als de geldigheid niet bepaald kan worden, retourneert XMLIsValid() de waarde 0.

Bijvoorbeeld:

```
select xmlisvalid(object_value)
from bestelling1;

XMLISVALID(OBJECT_VALUE)
------------------------
                       0
```

Deze functie is handig voor het gebruik in check constraints. In het volgende voorbeeld maken we een nieuwe tabel met een check constraint aan, gebaseerd op het schema voor bestellingen.

```
create table geldige_bestellingen of xmltype
 (check (XMLIsValid(object_value) = 1))
XMLSchema "http://www.5hart.com/schema/bestelling.xsd" element "bestelling";

CREATE TABLE succeeded.
```

Nu proberen we de aangepaste bestelling toe te voegen aan de tabel geldige_bestellingen. Dit zal niet lukken, omdat de aangepaste bestelling niet meer valid is.

```
insert into geldige_bestellingen
select * from bestelling1;

Error starting at line 1 in command:
insert into geldige_bestellingen
select * from bestelling1
Error report:
SQL Error: ORA-02290: check constraint (NWG802.SYS_C0013558) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

7.4.2 isSchemaValid()

De XMLType member functie isSchemaValid() lijkt op XMLIsValid(). Omdat het een member functie is, kan isSchemaValid() ook in PL/SQL gebruikt worden om de geldigheid van XMLType variabelen te controleren. In het volgende voorbeeld wordt het schema als parameter meegegeven.

```
create or replace function geldige_bestelling(bestelling xmltype) return number as
  xmldata xmltype := bestelling;
begin
  return xmldata.isschemavalid('http://www.5hart.com/schema/bestelling.xsd');
end;
/
```

Met deze functie kunnen we een bestelling controleren. Ook de functie isschemavalid() retourneert de waarde 1 als het document valid is. Als het document niet valid is, of als het validatieproces zelf niet lukt, retourneert de functie de waarde 0.

```
select case geldige_bestelling(
          xmlquery('/*' passing object_value returning content)
       )
       when 0 then 'invalid'
       when 1 then 'valid'
       end as waarde
from bestelling1;

WAARDE
-------
invalid
```

7.4.3 schemaValidate()

De member procedure schemaValidate() valideert het XML Document indien dat niet eerder is gebeurd. Deze procedure geeft een error indien het XML Document niet aan het schema voldoet. Als het document wel geldig is, dan krijgt het document de status VALIDATED.

De procedure schemaValidate() is geschikt voor gebruik binnen triggers. In het volgende voorbeeld maken we een trigger aan die een bestelling controleert voordat deze in de tabel bestellingen wordt opgeslagen:

```
CREATE OR REPLACE TRIGGER VALIDATE_BESTELLING
  before insert or update on BESTELLINGEN
  for each row
declare
  XMLDATA xmltype;
begin
  XMLDATA := :new.object_value;
  xmltype.schemavalidate(XMLDATA);
end;
/
```

Hierin staat `:new.object_value` voor de nieuw toe te voegen rij c.q. bestelling.

Om deze trigger te testen proberen we nogmaals de gewijzigde bestelling via Save to URL op te slaan. Het opslaan lukt niet, maar er verschijnt ook geen foutmelding.

De foutmelding wordt wel weergegeven als we een FTP programma gebruiken. We proberen dit nu uit. Wijzig eerst de hoeveelheid van de lokale versie van bestelling1.xml in 1000.

- Open een dos box (Start → run → cmd)
- Ga naar h:\ (type in h: → Enter)
- Ga naar h:\oraclexml (cd oraclexml)
- Open een FTP sessie met het commando ftp → Enter
- Type open <servernaam> 2100 → Enter
  (spatie tussen servernaam en poortnummer 2100)
- Wacht tot de gebruikersnaam kan worden ingetypt
- Geef uw gebruikersnaam op → Enter
- Geef uw wachtwoord op → Enter
- Ga naar uw XMLDB map: cd /home/cursisten/<gebruikersnaam>/bestellingen → Enter
- Verzend bestelling1.xml: send bestelling1.xml → Enter

Nu krijgen we wel een duidelijke foutmelding:

```
200 PORT Command successful
150 ASCII Data Connection
550- Error Response
ORA-00604: error occurred at recursive SQL level 1
ORA-31154: invalid XML document
ORA-19202: Error occurred in XML processing
LSX-00292: value "1000" is greater than maximum "100" (exclusive)
ORA-06512: at "SYS.XMLTYPE", line 354
ORA-06512: at "NWG802.VALIDATE_BESTELLING", line 6
ORA-04088: error during execution of trigger 'NWG802.VALIDATE_BESTELLING'
550 End Error Response
```

De melding bij Error 00292 toont de oorzaak van het probleem: de waarde "1000" is groter dan de maximale waarde "100".

7.4.4 isSchemaValidated()

De functie isSchemaValidated() is een member functie van XMLType. Deze functie retourneert de validatiestatus van het XMLType document: de waarde 1 als het document is gevalideerd, en anders 0. Zoals eerder vermeld kan de validatiestatus door de procedure schemaValidate() in het document worden vastgelegd.

7.4.5 setSchemaValidated()

Het kan zijn dat een document al gevalideerd is. De procedure schemaValidate() zal het document dan niet nogmaals valideren. Andersom zal schemaValidate() altijd proberen om een document te valideren die nog niet eerder gevalideerd is. Om dit gedrag te kunnen sturen kunnen we de procedure setSchemaValidated() gebruiken. Default zet deze procedure de status van een document op VALIDATED. We kunnen als parameter de waarde 0 (NOT VALIDATED) of 1 (VALIDATED) aan de procedure meegeven.
In het volgende voorbeeld zullen we dit aantonen.

Wijzig de code van de trigger als volgt:

```
CREATE OR REPLACE TRIGGER VALIDATE_BESTELLING
    before insert or update on BESTELLINGEN
    for each row
    declare
      XMLDATA xmltype;
    begin
       XMLDATA := :new.object_value;
       XMLDATA.setSchemaValidated();
       xmltype.schemavalidate(XMLDATA);
    end;
/
```

Maak de trigger opnieuw aan, wijzig de hoeveelheid van bestelling1 in 500 en sla de bestelling opnieuw op met Save to FTP or WebDAV Server.

Standaard krijgt elk gewijzigde XML Document de status NOT VALIDATED en zal schemaValidate() z'n werk doen. Nu wordt dit document echter gewoon opgeslagen, omdat de status van het document in PL/SQL op VALIDATED is gezet, en de procedure schemaValidate() dat dan niet opnieuw zal controleren.

7.5 Schema's genereren

Met behulp van de functie DBMS_XMLSCHEMA.GenerateSchema() kunnen schema's worden gegenereerd van bestaande object-typen. Dit is een bijzonder handig hulpmiddel voor Oracle ontwikkelaars die minder vertrouwd zijn met het schrijven van XML Schema's. De functie krijgt als parameters de naam van het database schema en de naam van het objecttype mee, en retourneert het gegenereerde XML Schema als XMLType.

De naamgeving van typen en kolommen bepaalt welke elementnamen het gegenereerde XML Schema zal beschrijven. In de volgende voorbeelden zijn die namen vet gedrukt:

```
create type punt_type as object(x number, y number);
/
create type lijn_type as object(p1 punt_type, p2 punt_type);
/
create type lijn_tab as varray(30) of lijn_type;
/
create type lijnen_type as object(lijn lijn_tab);
/
create type vorm is object(naam varchar2(30), lijnen lijnen_type);
/
select dbms_xmlschema.generateschema(user,'VORM') from dual;
```

```
DBMS_XMLSCHEMA.GENERATESCHEMA(USER,'VORM')
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xdb="http://xmlns.oracle.com/xdb"
xsi:schemaLocation="http://xmlns.oracle.com/xdb
http://xmlns.oracle.com/xdb/XDBSchema.xsd">
 <xsd:element name="VORM" type="VORMType" xdb:SQLType="VORM" xdb:SQLSchema="NWG802"/>
 <xsd:complexType name="VORMType" xdb:SQLType="VORM" xdb:SQLSchema="NWG802"
xdb:maintainDOM="false">
  <xsd:sequence>
   <xsd:element name="NAAM" xdb:SQLName="NAAM" xdb:SQLType="VARCHAR2">
    <xsd:simpleType>
     <xsd:restriction base="xsd:string">
      <xsd:maxLength value="30"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
   <xsd:element name="LIJNEN" type="LIJNEN_TYPEType" xdb:SQLName="LIJNEN"
xdb:SQLSchema="NWG802" xdb:SQLType="LIJNEN_TYPE"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="LIJNEN_TYPEType" xdb:SQLType="LIJNEN_TYPE"
xdb:SQLSchema="NWG802" xdb:maintainDOM="false">
  <xsd:sequence>
   <xsd:element name="LIJN" type="LIJN_TYPEType" maxOccurs="30" minOccurs="0"
xdb:SQLName="LIJN" xdb:SQLCollType="LIJN_TAB" xdb:SQLType="LIJN_TYPE"
xdb:SQLSchema="NWG802" xdb:SQLCollSchema="NWG802"/>
```

```
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="LIJN_TYPEType" xdb:SQLType="LIJN_TYPE" xdb:SQLSchema="NWG802"
xdb:maintainDOM="false">
  <xsd:sequence>
   <xsd:element name="P1" type="PUNT_TYPEType" xdb:SQLName="P1"
xdb:SQLSchema="NWG802" xdb:SQLType="PUNT_TYPE"/>
   <xsd:element name="P2" type="PUNT_TYPEType" xdb:SQLName="P2"
xdb:SQLSchema="NWG802" xdb:SQLType="PUNT_TYPE"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="PUNT_TYPEType" xdb:SQLType="PUNT_TYPE" xdb:SQLSchema="NWG802"
xdb:maintainDOM="false">
  <xsd:sequence>
   <xsd:element name="X" type="xsd:double" xdb:SQLName="X" xdb:SQLType="NUMBER"/>
   <xsd:element name="Y" type="xsd:double" xdb:SQLName="Y" xdb:SQLType="NUMBER"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

We kunnen dit schema als http://www.5hart.com/schema/vormen.xsd registreren. Daarbij moet de optie *gentypes* op false worden gezet: de objecttypen bestaan immers al:

```
declare
   doc xmltype;
begin
   select dbms_xmlschema.generateschema(user,'VORM') into doc from dual;
   dbms_xmlschema.registerschema('http://www.5hart.com/schema/vormen.xsd',
   doc, gentypes => false, enablehierarchy=>dbms_xmlschema.ENABLE_HIERARCHY_NONE);
end;
/
```

Er is nu een tabel aangemaakt, waarvan de naam bestaat uit de naam van het type, een gegenereerd nummer en _TAB als achtervoegsel. We kunnen daar een synoniem voor maken met een eenvoudiger naam:

```
select table_name
from user_xml_tables
where table_name like 'VORM%';

TABLE_NAME
------------------------------
VORM170_TAB

create synonym vormen for vorm170_tab;

Synonym created.
```

Het bestand vorm.xml voldoet aan het gegenereerde schema:

```
<VORM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.5hart.com/schema/vormen.xsd">
  <NAAM>hoek</NAAM>
  <LIJNEN>
    <LIJN>
      <P1>
         <X>0</X><Y>0</Y>
      </P1>
      <P2>
         <X>1</X><Y>0</Y>
      </P2>
    </LIJN>
```

```
    <LIJN>
      <P1>
        <X>1</X><Y>0</Y>
      </P1>
      <P2>
        <X>1</X><Y>2</Y>
      </P2>
    </LIJN>
  </LIJNEN>
</VORM>
```

Sla vorm.xml op in XMLDB. Hierdoor wordt een rij aan de tabel toegevoegd. We kunnen dit controleren door een selectie op het synoniem vormen uit te voeren:

```
select xmlquery('for $vorm in /* return $vorm'
         passing object_value returning content )
from vormen;
```

7.6 Schema evolutie

Soms is het nodig een geregistreerd schema aan te passen, bijvoorbeeld naar aanleiding van nieuwe regelgeving. Het is dan de vraag hoe dit het beste in XMLDB is toe te passen. De meest eenvoudige en drastische manier is het oude schema weg te gooien, inclusief alle afhankelijke tabellen, om vervolgens het nieuwe schema te registreren.
Als we oude gegevens willen bewaren zullen we subtieler te werk moeten gaan. Daarvoor bestaan twee opties:

copy-based schema evolutie	Hierbij worden de documenten die aan het nieuwe schema voldoen naar een tijdelijke locatie in de database gekopieerd. Het oude schema wordt verwijderd, het nieuwe geregistreerd, en daarna worden de documenten weer van hun tijdelijke locatie teruggeplaatst. Hierbij wordt gebruik gemaakt van de procedure DBMS_XMLSCHEMA.copyEvolve.
in-place schema evolutie	Dit is een snellere manier, zonder kopiëren van documenten, verwijderen van het oude schema, en terugplaatsen van de documenten. Deze manier is sinds Oracle 11g beschikbaar, gebruik makend van de procedure DBMS_XMLSCHEMA.inPlaceEvolve. Deze manier heeft als extra restricties dat het opslagmodel hetzelfde blijft (gestructureerd of binary XML), dat de volgorde van de elementen hetzelfde blijft en dat alle bestaande documenten ook al voldoen aan het nieuwe schema (eventueel na het toepassen van een XSLT-transformatie). Voorbeelden van wijzigingen aan een schema die via in-place schema evolution kunnen plaats vinden zijn het toevoegen van optionele elementen of attributen en het toevoegen van een complex of een simple type.

In de cursus zullen we deze onderwerpen niet nader behandelen. We verwijzen hiervoor naar de online documentatie.

8 PL/SQL packages

8.1 Inleiding

Als onderdeel van de XML Development kit (XDK) bestaan enkele PL/SQL packages om XML gegevens te verwerken. Daarvan behoren DBMS_XMLQuery en DBMS_XMLStore tot de XML SQL Utility (XSU). Met DBMS_XMLGen kunnen XML gegevens worden gegenereerd uit relationele data. DBMS_XMLStore is bedoeld om XML gegevens in relationele tabellen op te slaan, aan te passen of te verwijderen. Andere XDK packages worden in dit hoofdstuk korter behandeld.

8.2 DBMS_XMLGen

Met behulp van de package DBMS_XMLGen kunnen we een XMLType of een CLOB genereren die een XML document bevat. Dat gebeurt in de volgende stappen:

- Maak een "query context" aan door de functie XBMS_XMLGen.getCtx() aan te roepen en hieraan de query als CLOB of als VARCHAR2 mee te geven. Een query context kan worden gezien als een verzameling attributen, zoals de ROW tag naam en de ROWSET tag naam. De variabele die met behulp van getCtx() wordt aangemaakt, wordt de "context handle" genoemd: daarmee kan de query context worden gewijzigd.
- Bepaal eventueel de waarden van bind variabelen in de query met behulp van de DBMS_XMLGen.bind() functie.
- Bepaal optionele attributen van de context, zoals de ROW tag naam en de ROWSET tag naam.
- Fetch de XML als een CLOB met behulp van de getXML() functie, of als een XMLType met de getXMLType() functie.
- Sluit de context.

Onderstaande tabel geeft een overzicht van enkele procedures en functies uit deze package:

Methode	Omschrijving
CLOSECONTEXT	Sluit de query context
GETNUMROWSPROCESSED	Geeft het aantal opgehaalde rijen van de laatst uitgevoerde getXML() of getXMLType() functie.
GETXML	Genereert het XML document als CLOB
GETXMLTYPE	Genereert het XML document als XMLType
NEWCONTEXT	Maakt een query context aan en retourneert de context handle.
RESTARTQUERY	Voert de query nogmaals uit
SETBINDVALUE	Bepaalt de waarde van een bind variabele
SETNULLHANDLING	Geeft aan hoe null waarden worden behandeld
SETMAXROWS	Bepaalt het maximum aantal rijen dat wordt opgehaald
SETROWSETTAG	Bepaalt de tag naam van de dataset, default ROWSET.
SETROWTAG	Bepaalt de tag naam van een rij, default ROW.
SETSKIPROWS	Bepaalt het aantal rijen dat moet worden overgeslagen.
SETXSLT	Welke stylesheet moet worden toegepast op de gegenereerde XML.

Hiervan worden de methoden getNumRowsProcessed, setMaxRows() en setSkipRows() gebruikt om lange uitvoer in groepen rijen te kunnen tonen.

In het volgende voorbeeld maken we een functie werknemers_xmltype aan, die bij een opgegeven kantoornummer en functie de gegevens ophaalt van de betreffende werknemers:

```
create or replace function werknemers_xmltype
  (p_kantnr number, p_functie varchar2) return xmltype is

  genCtx DBMS_XMLGen.ctxType;
  resultaat XMLType;
begin
  genCtx := DBMS_XMLGen.newContext(
              'select * from ox_werknemers
              where kantnr = :KANTOOR
              and functie= :FUNCTIE');

  DBMS_XMLGen.setBindValue(genCtx,'KANTOOR',p_kantnr);
  DBMS_XMLGen.setBindValue(genCtx,'FUNCTIE',p_functie);
  DBMS_XMLGen.setNullHandling(genCtx,DBMS_XMLGen.EMPTY_TAG);
  resultaat := DBMS_XMLGen.getXMLType(genCtx);
  DBMS_XMLGen.closeContext(genCtx);

  return resultaat;
  end;
/
```

Bovenaan wordt de context bepaald. De query bevat twee bind variabelen in de where clause. Die krijgen een waarde met behulp van setBindValue().
Daaronder staat dat de null waarden als een lege tag moeten worden getoond: standaard wordt daarvoor geen tag gegenereerd.
Hierna wordt het resultaat van de query opgehaald via DBMS_XMLGen.getXMLType() en in een variabele gezet. Voordat we dit resultaat retourneren, sluiten we de context.
Probeer nu de volgende twee queries uit:

```
select werknemers_xmltype(30,'VERKOPER') from dual;
select werknemers_xmltype(10,'MANAGER') from dual;
```

Merk op dat bij de manager een leeg toeslag-element is gegenereerd:

```
WERKNEMERS_XMLTYPE(10,'MANAGER')
--------------------------------
<ROWSET>
 <ROW>
  <PERSNR>5810</PERSNR>
  <NAAM>HEUVEL</NAAM>
  <FUNCTIE>MANAGER</FUNCTIE>
  <MGR>6221</MGR>
  <SAL>3450</SAL>
  <TOESLAG/>
  <KANTNR>10</KANTNR>
 </ROW>
</ROWSET>
```

Het volgende voorbeeld laat zien hoe setSkipRows(), setMaxRows() en getNumRowsProcessed() gebruikt kunnen worden om de XML uitvoer op te delen in groepen van een bepaald aantal rijen. Dit is vooral handig voor webapplicaties. Het resultaat van een lange query kan daardoor verdeeld worden over meerdere pagina's.

```
create or replace package xmlgen_pck as
  teller number;
  genCtx dbms_xmlgen.ctxType;
  function toonxml(statement in varchar2, n in number) return xmltype;
end;
/

create or replace package body xmlgen_pck as
  function toonxml(statement in varchar2, n in number) return xmltype is
    resultaat xmltype;
  begin
    if genCtx is null then
      genCtx:=dbms_xmlgen.newcontext(statement);
      dbms_xmlgen.setmaxrows(genCtx, n);
    end if;
    dbms_xmlgen.setskiprows(genCtx,teller);
    resultaat := DBMS_XMLGen.getXMLType(genCtx);
    if dbms_xmlgen.getnumrowsprocessed(genCtx) = n then
      teller:=teller+n;
    else
      dbms_xmlgen.closecontext(genCtx);
      genCtx:=null;
    end if;
    return resultaat;
  end;
end;
/
```

De functie toonxml simuleert het bladeren door omvangrijke data, verspreid over meerdere pagina's. In dit voorbeeld kunnen we alleen vooruit bladeren. De functie toonxml werkt als volgt. Als de package variabele genCtx nog niet bestaat, wordt deze bij de eerste aanroep van de procedure gevuld, en wordt het maximaal aantal op te halen rijen op n gezet. Vervolgens worden teller rijen overgeslagen, waarna de volgende n rijen met getXMLType worden opgehaald. Indien minder dan n rijen zijn opgehaald, dan wordt de context gesloten en op null gezet.

```
select xmlgen_pck.toonxml('select naam from ox_patienten', 3) from dual;

XMLGEN_PCK.TOONXML('SELECTNAAMFROMOX_PATIENTEN',3)
--------------------------------------------------
<ROWSET>
 <ROW>
  <NAAM>Koopmans M.</NAAM>
 </ROW>
 <ROW>
  <NAAM>Schouten W.</NAAM>
 </ROW>
 <ROW>
  <NAAM>Elbers M.</NAAM>
 </ROW>
</ROWSET>
```

Herhaal dit statement en merk op dat dan de volgende drie rijen worden opgehaald.

8.3 dbms_xmlstore

DBMS_XMLStore stelt ons in staat om XML documenten in relationele tabellen op te slaan. We dienen daarvoor het XML document in een formaat aan te bieden die de package DBMS_XMLStore herkent:

- het root-element heeft de naam ROWSET;
- elke rij wordt omsloten met een element met de naam ROW;
- Elk tag naam binnen het ROW element komt overeen met de naam van een kolom.

De package dbms_xmlstore bevat de volgende methoden:

CLEARKEYCOLUMNLIST	Maakt de sleutelkolomlijst leeg
CLEARUPDATECOLUMNLIST	Maakt de update kolomlijst leeg
CLOSECONTEXT	Sluit de context
DELETEXML	Verwijdert de rijen uit de tabel, zoals gespecificeerd bij het aanmaken van de context, waarvan de sleutelkolom overeenkomt met die van het XML document
INSERTXML	Voegt het XML document toe aan de tabel, zoals gespecificeerd bij het aanmaken van de context
NEWCONTEXT	Maakt een save context voor een bepaalde tabel en retourneert de context handle
SETKEYCOLUMN	Deze methode voegt een sleutelkolom toe aan de sleutelkolomlijst
SETROWTAG	Geeft de naam van de tag op die de kolomwaarden omsluit. Standaard: ROW
SETUPDATECOLUMN	Voegt een kolom toe aan de update kolomlijst
UPDATEXML	Wijzigt de waarden van de kolommen zoals gespecificeerd in de update kolomlijst (default alle kolommen), van de rijen die overeenkomen met de sleutelkolomlijst, in de overeenkomende waarden binnen het XML document

In paragraaf 4.5 hebben we een stylesheet gemaakt waarmee we de gegevens in xml_kantoren naar het beoogde formaat kunnen transformeren. Het volgende voorbeeld laat dit zien voor kantoor 10:

```
select xmltransform(value(k), stylesheet)
from xml_kantoren_vw k, stylesheets s
where existsnode(value(k),'//kantoor[@kantnr="10"]')=1
and s.naam='werknemers.xsl';

XMLTRANSFORM(VALUE(K),STYLESHEET)
--------------------------------------------------------------------------------
<ROWSET>
  <ROW>
    <KANTNR>10</KANTNR>
    <PERSNR>5810</PERSNR>
    <MGR>6221</MGR>
    <NAAM>HEUVEL</NAAM>
    <SAL>3450</SAL>
    <FUNCTIE>MANAGER</FUNCTIE>
  </ROW>
```

```
  <ROW>
    <KANTNR>10</KANTNR>
    <PERSNR>6221</PERSNR>
    <NAAM>KRAAY</NAAM>
    <SAL>6000</SAL>
    <FUNCTIE>DIRECTEUR</FUNCTIE>
  </ROW>
  <ROW>
    <KANTNR>10</KANTNR>
    <PERSNR>8222</PERSNR>
    <MGR>5810</MGR>
    <NAAM>MANDERS</NAAM>
    <SAL>2300</SAL>
    <FUNCTIE>KLERK</FUNCTIE>
  </ROW>
</ROWSET>
```

We zullen deze stylesheet gebruiken in de volgende voorbeelden.

8.3.1 insertXml()

Met de functie insertXml() kunnen we rijen XML gegevens aan een relationele tabel toevoegen. In dit voorbeeld maken we een procedure aan die insertXml() aanroept:

```
create or replace procedure insert_Xmldoc(xmlDoc IN CLOB, tabelnaam IN VARCHAR2) is
   insCtx DBMS_XMLStore.ctxType;
   rijen number;
 begin
   if xmltype(xmldoc).existsnode('/ROWSET/ROW/*')=1 then
     insCtx := DBMS_XMLStore.newContext(tabelnaam);    -- maak een context handle
     rijen := DBMS_XMLStore.insertXML(insCtx,xmlDoc); -- insert het document
     DBMS_XMLStore.closeContext(insCtx);              -- sluit de context handle
   end if;
end;
```

Deze procedure controleert eerst of het ingevoerde document /ROWSET/ROW elementen bevat. Als de documentstructuur klopt, wordt met newContext() een insert context gecreëerd. Dit is vergelijkbaar met de query context die we bij dbms_xmlgen hebben behandeld. In dit geval wordt een tabelnaam aan de context meegegeven. De rijen worden vervolgens aan de tabel toegevoegd met insertXML(). Deze functie retourneert het aantal rijen dat is toegevoegd. Tot slot wordt de context handle gesloten.

We gaan deze procedure gebruiken om een lege tabel te vullen. We maken deze lege tabel eerst aan als een kopie van de tabel werknemers.

```
create table ox_werknemers_kopie
as select * from ox_werknemers
where 1=0;
```

Door de where clausule (where 1=0) blijft deze kopie leeg.

Nu zullen we deze tabel vullen. Voor elke rij in xml_kantoren_vw zullen we de getransformeerde versie aan de procedure insert_xmldoc meegeven:

```
begin
for rij in (select xmltransform(value(k), stylesheet).getCLOBVal() doc
   from xml_kantoren_vw k, stylesheets s
   where s.naam='werknemers.xsl') loop
   insert_xmldoc(rij.doc, 'OX_WERKNEMERS_KOPIE');
end loop;
end;
/

select * from ox_werknemers_kopie;
```

PERSNR	NAAM	FUNCTIE	MGR	SAL	TOESLAG	KANTNR
5810	HEUVEL	MANAGER	6221	3450		10
6221	KRAAY	DIRECTEUR		6000		10
8222	MANDERS	KLERK	5810	2300		10
3381	SMITS	KLERK	7902	2400		20
3930	PIETERS	MANAGER	6221	3975		20
5931	SANDERS	ANALIST	3930	4000		20
6681	ADELAAR	KLERK	5931	2100		20
7902	VERMEULEN	ANALIST	3930	3900		20
3462	ALKEMA	VERKOPER	4621	2600	300	30
3518	WALSTRA	VERKOPER	4621	2250	500	30
4510	VERGEER	VERKOPER	4621	2250	1400	30
4621	KLAASEN	MANAGER	6221	3850		30
6500	DROST	VERKOPER	4621	2500	0	30
7900	APPEL	KLERK	4621	1950		30

8.3.2 updateXML()

Met de functie updateXML() kunnen we rijen XML gegevens in een relationele tabel wijzigen. Deze functie lijkt in het gebruik op insertXml(). Bij updateXML() geven we ook aan welke kolommen geupdate moeten worden met setUpdateColumn(), en welke kolom de primary key bevat met setKeyColumn(). UpdateXML() gebruikt de primary key kolom in de where clausule van de update.

In dit voorbeeld zullen we gebruik maken van een XML bestand. Dit bestand bevat drie werknemers die 5 % meer gaan verdienen. Als extra wijziging zijn de functies van deze werknemers naar kleine letters omgezet. Sla dit bestand op in XMLDB. We kunnen in SQL naar de inhoud van dit bestand verwijzen met behulp van het type xdburitype:

```
select xdburitype('/home/cursisten/nwg802/werknemers.xml').getXml()
from dual;


XDBURITYPE('/HOME/CURSISTEN/NWG802/WERKNEMERS.XML').GETXML()
--------------------------------------------------------------------------------
<werknemers>
  <werknemer persnr="3381" mgr="7902">
    <naam>SMITS</naam>
    <sal>2520</sal>
    <functie>klerk</functie>
  </werknemer>
  <werknemer persnr="7902" mgr="3930">
    <naam>VERMEULEN</naam>
    <sal>4095</sal>
    <functie>analisten</functie>
  </werknemer>
  <werknemer persnr="7900" mgr="4621">
    <naam>APPEL</naam>
    <sal>2047.5</sal>
    <functie>klerk</functie>
  </werknemer>
</werknemers>
```

Met behulp van werknemers.xsl kunnen we deze omzetten naar de standaardvorm met ROWSET en ROW elementen.

```
select xdburitype('/home/cursisten/nwg802/werknemers.xml').getXml()
.transform(stylesheet)
from stylesheets
where naam='werknemers.xsl';

XDBURITYPE('/HOME/CURSISTEN/NWG802/WERKNEMERS.XML').GETXML().TRANSFORM(STYLESHEE
--------------------------------------------------------------------------------
<?xml version="1.0" encoding="utf-8"?>

<ROWSET>
  <ROW>
    <PERSNR>3381</PERSNR>
    <MGR>7902</MGR>
    <NAAM>SMITS</NAAM>
    <SAL>2520</SAL>
    <FUNCTIE>klerk</FUNCTIE>
  </ROW>
  <ROW>
    <PERSNR>7902</PERSNR>
    <MGR>3930</MGR>
    <NAAM>VERMEULEN</NAAM>
    <SAL>4095</SAL>
    <FUNCTIE>analist</FUNCTIE>
  </ROW>
  <ROW>
    <PERSNR>7900</PERSNR>
    <MGR>4621</MGR>
    <NAAM>APPEL</NAAM>
    <SAL>2047.5</SAL>
    <FUNCTIE>klerk</FUNCTIE>
  </ROW>
</ROWSET>
```

Deze stylesheet gebruiken we in het volgende voorbeeld:

```
accept gebruikersnaam char prompt 'wat is uw gebruikersnaam? '

declare
   xmldoc xmltype;
   xsldoc xmltype;
   updCtx DBMS_XMLStore.ctxType;
   rijen number;
   gebruikersnaam varchar2(20) := lower('&gebruikersnaam');
 begin
   select xdburitype('/home/cursisten/'||gebruikersnaam||'/werknemers.xml').getXml()
   into xmldoc from dual;                                    -- zet werknemers.xml
                                                             -- in xmldoc
   select stylesheet into xsldoc
   from stylesheets
   where naam='werknemers.xsl';                              -- zet stylesheet in xsldoc

   xmldoc:=xmldoc.transform(xsldoc);                         -- transformeer xmldoc

   if xmldoc.existsnode('/ROWSET/ROW/*')=1 then
     updCtx := DBMS_XMLStore.newContext('OX_WERKNEMERS_KOPIE');
     DBMS_XMLStore.setKeyColumn(updCtx,'PERSNR');            -- persnr is de primary key
     DBMS_XMLStore.setUpdateColumn(updCtx,'SAL');            -- sal wordt gewijzigd
     DBMS_XMLStore.setUpdateColumn(updCtx,'FUNCTIE');        -- functie wordt gewijzigd
     rijen := DBMS_XMLStore.updateXML(updCtx,xmlDoc);        -- update de tabel
     DBMS_XMLStore.closeContext(updCtx);
   end if;
end;
/

Wat is uw gebruikersnaam? nwg802

anonymous block completed

select * from ox_werknemers_kopie
order by functie desc;
```

PERSNR	NAAM	FUNCTIE	MGR	SAL	TOESLAG	KANTNR
3381	SMITS	**klerk**	7902	**2520**		20
7900	APPEL	**klerk**	4621	**2048**		30
7902	VERMEULEN	**analist**	3930	**4095**		20
3462	ALKEMA	VERKOPER	4621	2600	300	30
4510	VERGEER	VERKOPER	4621	2250	1400	30
3518	WALSTRA	VERKOPER	4621	2250	500	30
6500	DROST	VERKOPER	4621	2500	0	30
4621	KLAASEN	MANAGER	6221	3850		30
5810	HEUVEL	MANAGER	6221	3450		10
3930	PIETERS	MANAGER	6221	3975		20
8222	MANDERS	KLERK	5810	2300		10
6681	ADELAAR	KLERK	5931	2100		20
6221	KRAAY	DIRECTEUR		6000		10
5931	SANDERS	ANALIST	3930	4000		20

8.3.3 deleteXml()

Met behulp van deleteXML() kunnen we rijen uit een tabel verwijderen. Met setKeyColumn geven we aan welke rijen verwijderd worden.

In het volgende voorbeeld verwijderen we de werknemers die voorkomen in een xml bestand uit de tabel ox_werknemers_kopie.

```
select count(*) from ox_werknemers_kopie;

  COUNT(*)
----------
        14

declare
   xmldoc xmltype;
   xsldoc xmltype;
   delCtx DBMS_XMLStore.ctxType;
   rijen number;
   gebruikersnaam varchar2(20) := 'nwg802';

 begin
   select xdburitype('/home/cursisten/'||gebruikersnaam||'/werknemers2.xml').getXml()
   into xmldoc from dual;                                  -- zet werknemers2.xml
                                                           -- in xmldoc
   select stylesheet into xsldoc
   from stylesheets
   where naam='werknemers.xsl';                            -- zet stylesheet in xsldoc

   xmldoc:=xmldoc.transform(xsldoc);                       -- transformeer xmldoc

   if xmldoc.existsnode('/ROWSET/ROW/*')=1 then
     delCtx := DBMS_XMLStore.newContext('OX_WERKNEMERS_KOPIE');
     DBMS_XMLStore.setKeyColumn(delCtx,'PERSNR');          -- persnr is de primary key
     rijen := DBMS_XMLStore.deleteXML(delCtx,xmlDoc);      -- verwijder de rijen
     DBMS_XMLStore.closeContext(delCtx);
   end if;
end;
/

anonymous block completed.

select count(*) from ox_werknemers_kopie;

  COUNT(*)
----------
        11
```

8.4 Andere packages

De XDK bevat nog enkele andere geavanceerde packages. In deze paragraaf worden ze genoemd om een indruk te krijgen van de mogelijkheden ervan. Voor meer informatie verwijzen we naar de online help documenten.

8.4.1 dbms_xmlsave

Deze package is vergelijkbaar met dbms_xmlstore. Dbms_xmlstore is in C geprogrammeerd en gecompileerd in de PL/SQL kernel, terwijl dbms_xmlsave een PL/SQL interface is op een Java package. DBMS_XMLStore is daardoor, en door een beter gebruik van SAX om XML te parsen, sneller dan dbms_xmlsave. Daar staat tegenover dat dbms_xmlsave uitgebreider is.

8.4.2 dbms_xmlquery

Deze package is vergelijkbaar met dbms_xmlgen. De verschillen lijken op die van dbms_xmlstore en dbms_xmlsave: dbms_xmlgen is in C geprogrammeerd en gecompileerd in de PL/SQL kernel en is daardoor sneller; dbms_xmlquery is geprogrammeerd in Java en is uitgebreider.

8.4.3 dbms_xmldom

Het Document Object Model (DOM) is een "Application Programming Interface" (API) voor HTML en XML documenten. Deze standaard is voor PL/SQL vertaald in een zeer uitgebreide package waarmee een XML document als boomstructuur kan worden benaderd. Deze package maakt het mogelijk om nagenoeg elk deel van deze boomstructuur te veranderen, verwijderen of toe te voegen. Omdat DOM een objectgeoriënteerde API is, en PL/SQL een procedurele taal is, komt de dbms_xmldom package niet geheel overeen met de DOM specificatie.

8.4.4 dbms_xmlparser

Gebruik deze package om XML documenten in te lezen. Als resultaat van het parsen retourneert de getDocument() functie een DOM Document, die verder verwerkt kan worden met de dbms_xmldom package.

8.4.5 dbms_xslprocessor

Met behulp van de package dbms_xslprocessor kunnen we (delen van) XML documenten naar andere XML documenten transformeren met behulp van XSLT stylesheets of losse XPATH expressies.

8.4.6 dbms_xdb

De dbms_xdb package wordt gebruikt om de XDB repository te onderhouden. Hiermee kunnen we onder meer bronnen aanmaken of verwijderen; rechten over deze bronnen verlenen aan gebruikers; en de configuratie ophalen, wijzigen of verversen. In hoofdstuk 5 is behandeld hoe deze package kan worden gebruikt om bronnen te onderhouden.

Appendix A Tabellen

tabel ox_kantoren

beschrijving

```
KANTNR          NUMBER(2)       NOT NULL
NAAM            VARCHAR2(15)
PLAATS          VARCHAR2(10)
```

inhoud

KANTNR	NAAM	PLAATS
10	BOEKHOUDING	AMSTERDAM
20	ONDERZOEK	UTRECHT
30	VERKOOP	DEN HAAG
40	PRODUCTIE	ARNHEM

tabel ox_werknemers

beschrijving

```
PERSNR          NUMBER(4)       NOT NULL
NAAM            VARCHAR2(10)
FUNCTIE         VARCHAR2(9)
MGR             NUMBER(4)
SAL             NUMBER(5)
TOESLAG         NUMBER(5)
KANTNR          NUMBER(2)
```

inhoud

PERSNR	NAAM	FUNCTIE	MGR	SAL	TOESLAG	KANTNR
3381	SMITS	KLERK	7902	2400		20
3462	ALKEMA	VERKOPER	4621	2600	300	30
3518	WALSTRA	VERKOPER	4621	2250	500	30
3930	PIETERS	MANAGER	6221	3975		20
4510	VERGEER	VERKOPER	4621	2250	1400	30
4621	KLAASEN	MANAGER	6221	3850		30
5810	HEUVEL	MANAGER	6221	3450		10
5931	SANDERS	ANALIST	3930	4000		20
6221	KRAAY	DIRECTEUR		6000		10
6500	DROST	VERKOPER	4621	2500	0	30
6681	ADELAAR	KLERK	5931	2100		20
7900	APPEL	KLERK	4621	1950		30
7902	VERMEULEN	ANALIST	3930	3900		20
8222	MANDERS	KLERK	5810	2300		10

tabel ox_ziekenhuizen

beschrijving

```
ZIEKHNR         NUMBER(2)       NOT NULL
NAAM            VARCHAR2(15)
PLAATS          VARCHAR2(15)
TELEFOON        VARCHAR2(12)
TOTBED          NUMBER(4)
```

inhoud

Appendix A Tabellen

ZIEKHNR	NAAM	PLAATS	TELEFOON	TOTBED
10	AMC	Amsterdam	020-6532617	502
15	Diaconessen	Utrecht	030-2646362	587
20	Antonius	Nieuwegein	030-6045632	412
25	Zuiderzee	Lelystad	0320-278391	845

tabel ox_afdelingen

beschrijving

ZIEKHNR	NUMBER(2)	NOT NULL
AFDNR	NUMBER(2)	NOT NULL
NAAM	VARCHAR2(15)	
TOTBED	NUMBER(4)	

inhoud

ZIEKHNR	AFDNR	NAAM	TOTBED
10	3	Intensive Care	21
10	6	Psychiatrie	67
15	1	Hersteloord	13
15	3	Intensive Care	10
15	4	Hartafdeling	53
20	1	Hersteloord	10
20	2	Kinder	34
20	6	Psychiatrie	118
25	2	Kinder	24
25	4	Hartafdeling	55

tabel ox_stafleden

beschrijving

ZIEKHNR	NUMBER(2)	NOT NULL
AFDNR	NUMBER(2)	NOT NULL
PERSNR	NUMBER(4)	NOT NULL
NAAM	VARCHAR2(15)	
FUNCTIE	VARCHAR2(15)	
DIENST	VARCHAR2(6)	
SAL	NUMBER(5)	

inhoud

ZIEKHNR	AFDNR	PERSNR	NAAM	FUNCTIE	DIENST	SAL
10	6	3526	Dinter B.	Verpleegster	A	17400
20	6	8574	Beek G.	Zaalknecht	N	12600
10	6	3198	Hursman J.	Zaalknecht	A	13500
15	4	2342	Keyzer W.	Assistent	A	18300
20	1	5342	Coolen R.	Verpleegster	A	16300
20	1	8632	Riksen G.	Verpleegster	D	20200
20	2	3257	Mensink C.	Assistent	N	17000
20	6	2315	Horst D.	Verpleegster	N	18300
25	2	9835	Fleskes H.	Verpleegster	A	19400
25	4	6543	Arends R.	Assistent	D	17000

Appendix A Tabellen

tabel ox_patienten

beschrijving

```
PATNR           NUMBER(5)       NOT NULL
NAAM            VARCHAR2(15)
PLAATS          VARCHAR2(15)
GEBDAT          DATE
MV              VARCHAR2(1)
ZIEKFNR         NUMBER(7)
```

inhoud

PATNR	NAAM	PLAATS	GEBDAT	M	ZIEKFNR
11321	Koopmans M.	Utrecht	11-DEC-66	M	3542764
12816	Schouten W.	Den Haag	23-APR-73	V	7466384
19381	Elbers M.	Amsterdam	01-JAN-76	V	9753728
25218	Dekker B.	Utrecht	05-NOV-54	M	8466355
30940	Lammers T.	Arnhem	12-APR-43	V	3452718
38911	Jong H.	Nijmegen	12-JAN-82	M	4656238
39410	Manders G.	Den Bosch	11-DEC-70	M	2794710
45630	Ravenhorst P.	Eindhoven	04-FEB-48	M	9872513
48220	Feenstra A.	Breda	27-FEB-77	V	3529976
50333	Horst E.	Utrecht	12-APR-64	M	1232988

tabel ox_bezetting

beschrijving

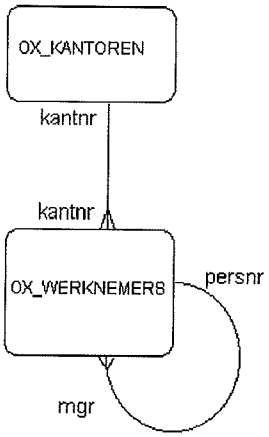
```
PATNR           NUMBER(5)       NOT NULL
ZIEKHNR         NUMBER(2)       NOT NULL
AFDNR           NUMBER(2)       NOT NULL
BEDNR           NUMBER(2)       NOT NULL
```

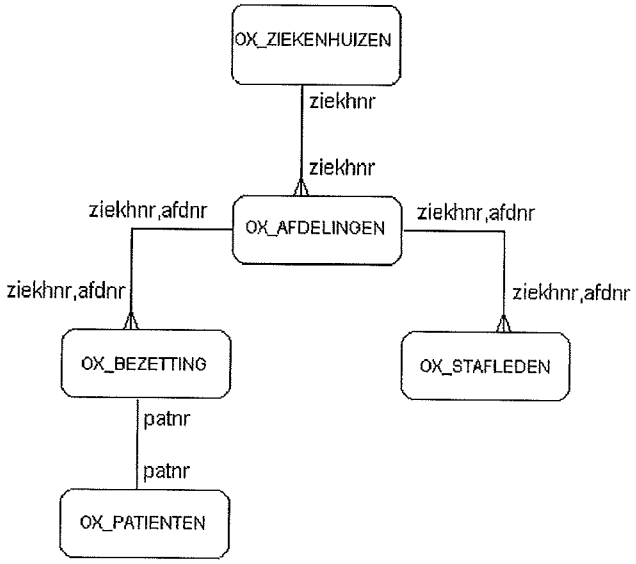
inhoud

PATNR	ZIEKHNR	AFDNR	BEDNR
11321	10	3	1
12816	10	3	2
19381	10	3	3
25218	15	4	1
30940	15	4	2
38911	20	6	1
39410	20	6	2
45630	20	6	3
48220	20	2	1
50333	25	4	1

Appendix A Tabellen

Relaties tussen de tabellen





Appendix B Opdrachten

Opdrachten hoofdstuk 2

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml2.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml2.002.

1. Maak met behulp van WebDAV in explorer een nieuwe folder "opdrachten" binnen uw eigen folder.

2. Wijzig de inhoud van voorbeeld.xml zonder deze wijziging op te slaan. Bewaar het gewijzigde bestand als voorbeeld2.xml in de nieuwe folder met behulp van save to FTP or WebDAV Server in XMLBlueprint.

3. Toon de inhoud van voorbeeld2.xml op binnen SQL Developer.

Appendix B Opdrachten

Appendix B Opdrachten

Opdrachten hoofdstuk 3

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml3.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml3.002.

1. Genereer XML-output van de patiënten uit Utrecht met behulp van SQL/XML functies. De inhoud dient er als volgt uit te zien (pretty-print is optioneel):

```
patienten
--------------------------------------------
<patienten>
  <patient patnr="11321" ziekfnr="3542764">
    <naam>Koopmans M.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>11-DEC-66</gebdat>
    <mv>M</mv>
  </patient>
  <patient patnr="25218" ziekfnr="8466355">
    <naam>Dekker B.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>05-NOV-54</gebdat>
    <mv>M</mv>
  </patient>
  <patient patnr="50333" ziekfnr="1232988">
    <naam>Horst E.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>12-APR-64</gebdat>
    <mv>M</mv>
  </patient>
</patienten>
```

   Tip: maak gebruik van de functies XMLElement, XMLAttributes, XMLAgg en XMLForest

2. Haal uit de tabellen afdelingen en stafleden van afdeling 1 in ziekenhuis 20 de namen en nummers op van de afdeling en van de stafleden die er werken. Het resultaat dient er als volgt uit te zien (Tip: gebruik geneste cursors in een XMLType constructor. Pretty-print is optioneel):

```
AFDELING
------------------------------
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <NAAM>Hersteloord</NAAM>
  <AFDNR>1</AFDNR>
  <STAFLEDEN>
   <STAFLEDEN_ROW>
    <NAAM>Riksen G.</NAAM>
    <PERSNR>8632</PERSNR>
   </STAFLEDEN_ROW>
   <STAFLEDEN_ROW>
    <NAAM>Coolen R.</NAAM>
    <PERSNR>5342</PERSNR>
   </STAFLEDEN_ROW>
  </STAFLEDEN>
 </ROW>
</ROWSET>
```

3. **optioneel:** Haal van elke functie het gemiddelde salaris, het maximale salaris, het minimale salaris en het aantal werknemers op. De uitvoer van elke functie dient er als volgt uit te zien:

```
<functie naam="ANALIST">
  <column name="gemiddeld salaris">3950</column>
  <column name="maximaal salaris">4000</column>
  <column name="minimaal salaris">3900</column>
  <column name="aantal werknemers">2</column>
</functie>
```

4. **optioneel**: Zorg dat dezelfde gegevens als van opgave 2 als volgt worden weergegeven, door gebruik te maken van SQL/XML functies

```
afdelingen
--------------------------------------
<afdelingen>
  <afdeling afdnr="1">
    <stafleden>
      <staflid persnr="8632">
        <naam>Riksen G.</naam>
      </staflid>
      <staflid persnr="5342">
        <naam>Coolen R.</naam>
      </staflid>
    </stafleden>
  </afdeling>
</afdelingen>
```

Appendix B Opdrachten

Opdrachten hoofdstuk 4

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml4.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml4.002.

1. In uw home directory staat xml_ziekenhuizen_vw.sql, waarmee de xmltype view xml_ziekenhuizen_vw kan worden gemaakt. Voer dit script uit en bekijk de structuur van deze view.

2. Toon van xml_ziekenhuizen_vw alle stafleden uit de avonddienst met een salaris groter dan 18000. Maak gebruik van xmlquery met een XPath expressie.

3. Toon het staflid met persnr 2342. Voeg daarin een extra element voornaam toe, met als waarde Wietze.

4. Maak een tabel xml_ziekenhuizen aan op basis van xml_ziekenhuizen_vw.

   Optioneel:

5. Voeg daaraan met behulp van SQL*Loader de gegevens toe uit het bestand ziekenhuis30.xml.

6. Toon van het ziekenhuis met ziekhnr 25 alle afdelingen, zonder de stafleden.
   Tip: maak gebruik van deleteXML(), met daarbinnen XMLQuery(). Kijk eventueel naar een vergelijkbaar voorbeeld bij updateXML(). Uitvoer:

```
AFDELINGEN
----------------------------------------
<afdelingen>
   <afdeling afdnr="4">
      <naam>Hartafdeling</naam>
      <totbed>55</totbed>
      <patienten>
         <patient patnr="50333">
            <naam>Horst E.</naam>
            <plaats>Utrecht</plaats>
            <gebdat>1964-04-12</gebdat>
            <mv>M</mv>
            <ziekfnr>1232988</ziekfnr>
         </patient>
      </patienten>
   </afdeling>
   <afdeling afdnr="2">
      <naam>Kinder</naam>
      <totbed>24</totbed>
      <patienten/>
   </afdeling>
</afdelingen>
```

Appendix B Opdrachten

Appendix B Opdrachten

Opdrachten hoofdstuk 5

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml5.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml5.002.

1. a. Geef op basis van een XQuery view op de tabel OX_STAFLEDEN alle functies op alfabetische volgorde. De uitvoer dient er als volgt uit te zien (zonder pretty-print):

```
<functies>
   <functie naam="Assistent"></functie>
   <functie naam="Verpleegster"></functie>
   <functie naam="Zaalknecht"></functie>
</functies>
```

   b. **(Optioneel)** Geef ook de namen van de stafleden binnen de de betreffende functie-elementen:

```
<functies>
   <functie naam="Assistent">
      <naam>Arends R.</naam>
      <naam>Keyzer W.</naam>
      <naam>Mensink C.</naam>
   </functie>
   ....
</functies>
```

2. a. Zet via WebDAV het bestand patienten.xml in XDB in de map ziekenhuizen. Bekijk de structuur van dit bestand.
   b. Toon alle gegevens van de patiënten met behulp van XMLTable(), alsof het een relationele tabel is (zonder XML-opmaak, in rijen en kolommen), op volgorde van geboortedatum.

3. Toon van de XML-tabel xml_ziekenhuizen de namen van de ziekenhuizen, de afdelingen en de stafleden. Toon daarbij ook de afdelingen zonder stafleden. Maak gebruik van XMLTable().

4. **(Optioneel)** Toon uit xml_ziekenhuizen_vw alle stafleden uit de avonddienst met een salaris groter dan 18000, op volgorde van naam.
   Tip: maak gebruik van XMLAgg() in een inline view om alle ziekenhuizen samen te benaderen.

Appendix B Opdrachten

Appendix B Opdrachten

Opdrachten hoofdstuk 6 (optioneel)

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml6.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml6.002.

1. Maak in de folder ziekenhuizen nieuwe bronnen aan met de naam ziekenhuis<ziekhnr>.xml . Deze bronnen verwijzen naar de rijen in xml_ziekenhuizen_vw.

2. Maak een folder hospitaals aan als link naar de folder ziekenhuizen.

3. Maak de view folder_view die alle kolommen van path_view bevat van de folders onder /home/cursisten/<gebruikersnaam>

4. Onder de map /sys staan bronnen die door XML DB intern worden gebruikt. Haal alle relatieve paden onder /sys op met een diepte van 5.

5. Verwijder de map links inclusief inhoud met behulp van dbms_xdb.

Appendix B Opdrachten

Appendix B Opdrachten

Opdrachten hoofdstuk 7

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxml7.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxml7.002.

In deze opdracht zult u een XML Schema genereren voor kantoren op basis van objecttypen. Dit schema zal een andere structuur beschrijven dan we kennen van xml_kantoren en xml_kantoren_vw. Uit objecttypen kunnen we namelijk geen attributen genereren.

1. Maak de objecttypen aan door het script create_types.sql te draaien.

2. Bekijk de inhoud van het script create_types en bepaal op basis van welk type het schema moet worden gegenereerd.

3. Genereer het schema en registreer dit onder de url http://www.5hart.com/schema/kantoren.xsd.

4. Bepaal welke default tabel is aangemaakt. Maak voor deze tabel het synoniem xsd_kantoren aan. Tip: tabelnamen zijn hoofdlettergevoelig.

   De volgende opgaven zijn optioneel:

5. Pas de bestanden h:\oraclexml\kantoor10.xml en h:\oraclexml\kantoor20.xml aan, zodat deze gebruik maken van het geregistreerde schema.

6. Probeer deze bestanden via WebDaV of FTP toe te voegen aan de submap opdrachten. Probeer eventuele fouten in de bestanden zelf te verbeteren.

7. Controleer of de bestanden zijn toegevoegd aan xsd_kantoren door de naam van elk kantoor te tonen.

Appendix B Opdrachten

Appendix B Opdrachten

Opdrachten hoofdstuk 8

Voor zover opdrachten in SQL Developer worden gemaakt, geef dan eenmalig het commando "set echo on". Bewaar de uitvoer als h:\oxm8.001. Indien opdrachten een volgende dag worden afgemaakt, bewaar dan de uitvoer onder een volgend volgnummer, bijvoorbeeld h:\oxm8.002.

We kunnen op een XML view een instead of trigger zetten, zodat ingevoerde xml-gegevens automatisch in de onderliggende relationele tabellen worden verwerkt. Wij doen dit voor de volgende view (dit staat in create_xml_werkn_vw.sql):

```
  create or replace view xml_werknemers_vw of xmltype
with object id (XMLCast(XMLQuery('/ROW/PERSNR/text()'
                 passing object_value returning content) as integer))
as select column_value
from xmltable('for $rij in
               ora:view("OX_WERKNEMERS")/ROW
               return $rij');
```

1. Voer dit statement uit en controleer de inhoud van deze view.

2. Maak vervolgens de insteadof trigger aan. Een deel van de code staat alvast in create_xml_werkn_trg.sql.

3. Voeg werkn1.xml en werkn2.xml via WebDaV of FTP toe aan uw map opdrachten binnen XML DB. Voeg vervolgens werkn1.xml toe aan xml_werknemers_vw en controleer daarna de inhoud van zowel de view als de onderliggende tabel.

4. Haal de functie en het salaris van de werknemer met persnr 8222 op. Wijzig de gegevens van deze werknemer vervolgens met behulp van werkn2.xml. Controleer de view en de onderliggende tabel daarna opnieuw.

Appendix B Opdrachten

XML voor Oracle ontwikkelaars 11*g*

Uitwerkingen

Versie 3.2 juli-2014

Appendix A Uitwerkingen

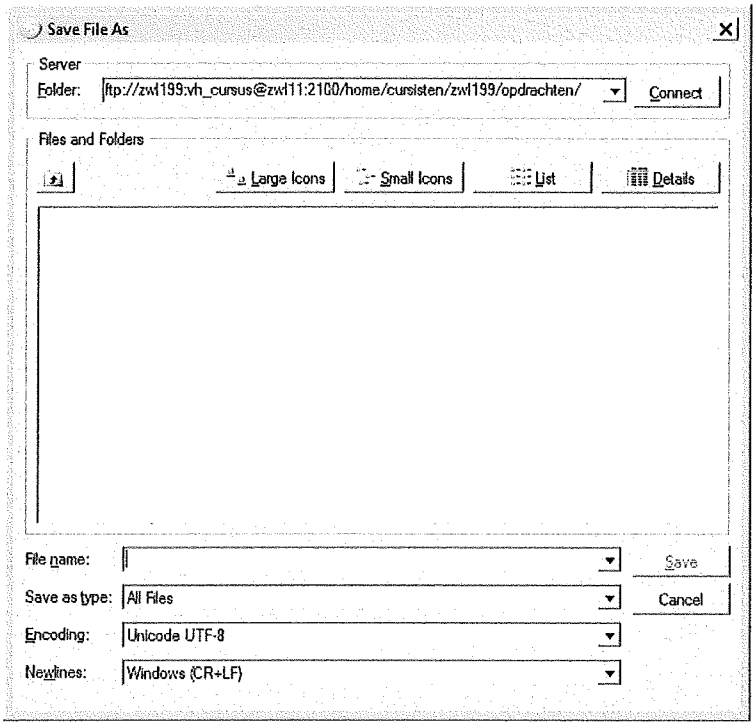
Opdrachten hoofdstuk 2

1. Maak met behulp van FTP in explorer een nieuwe folder "opdrachten" binnen uw eigen folder.

   Open uw eigen webmap, klik met de rechter muisknop ergens binnen deze map en kies nieuw -> map. Geef deze de naam "opdrachten".

2. Wijzig de inhoud van voorbeeld.xml zonder deze wijziging op te slaan. Bewaar het gewijzigde bestand als voorbeeld2.xml in de nieuwe folder met behulp van save to FTP or WebDAV Server in XMLBlueprint.

   Wijzig de tekst, en klik op File> Save to FTP or WebDAV Server. Dubbelklik in het dialoogvenster op de map "opdrachten" en geef bij File name de naam voorbeeld2.xml. Kies dan Save.



3. Toon de inhoud van voorbeeld2.xml binnen SQL Developer.

```
select xdburitype('/home/cursisten/nwg802/opdrachten/voorbeeld2.xml').getXml()
from dual;

XDBURITYPE('/HOME/CURSISTEN/NWG802/OPDRACHTEN/VOORBEELD2.XML').GETXML()
--------------------------------------------------------------------------------
<voorbeelden>
  <voorbeeld nr="1">
    Dit is een voorbeeld in de map opdrachten.
  </voorbeeld>
  <voorbeeld nr="2">
    Dit is een gewijzigd voorbeeld.
  </voorbeeld>
</voorbeelden>
```

Appendix A Uitwerkingen

Opdrachten hoofdstuk 3

1. Genereer XML-output van de patiënten uit Utrecht met behulp van SQL/XML functies. De uitvoer dient er als volgt uit te zien (pretty print is optioneel):

```
patienten
------------------------------------------------
<patienten>
  <patient patnr="11321" ziekfnr="3542764">
    <naam>Koopmans M.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>11-DEC-66</gebdat>
    <mv>M</mv>
  </patient>
  <patient patnr="25218" ziekfnr="8466355">
    <naam>Dekker B.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>05-NOV-54</gebdat>
    <mv>M</mv>
  </patient>
  <patient patnr="50333" ziekfnr="1232988">
    <naam>Horst E.</naam>
    <plaats>Utrecht</plaats>
    <gebdat>12-APR-64</gebdat>
    <mv>M</mv>
  </patient>
</patienten>
```

uitwerking:

```
select xmlelement("patienten",xmlagg(
       xmlelement("patient",
       xmlattributes(patnr as "patnr", ziekfnr as "ziekfnr"),
       xmlforest(naam "naam", plaats "plaats", gebdat "gebdat", mv "mv")
       ))) as "patienten"
from patienten
where plaats = 'Utrecht';
```

Of ingesprongen:

```
select xmlserialize(document
xmlelement("patienten",xmlagg(
       xmlelement("patient",
       xmlattributes(patnr as "patnr", ziekfnr as "ziekfnr"),
       xmlforest(naam "naam", plaats "plaats", gebdat "gebdat", mv "mv")
       )))
       as clob indent)
       as "patienten"
from patienten
where plaats = 'Utrecht';
```

Appendix A Uitwerkingen

2. Haal uit de tabellen afdelingen en stafleden van afdeling 1 in ziekenhuis 20 de namen en nummers op van de afdeling en van de stafleden die er werken. Het resultaat dient er als volgt uit te zien:

```
AFDELING
------------------------------
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <NAAM>Hersteloord</NAAM>
  <AFDNR>1</AFDNR>
  <STAFLEDEN>
   <STAFLEDEN_ROW>
    <NAAM>Riksen G.</NAAM>
    <PERSNR>8632</PERSNR>
   </STAFLEDEN_ROW>
   <STAFLEDEN_ROW>
    <NAAM>Coolen R.</NAAM>
    <PERSNR>5342</PERSNR>
   </STAFLEDEN_ROW>
  </STAFLEDEN>
 </ROW>
</ROWSET>
```

Tip: maak gebruik van geneste cursors in een XMLType constructor.

```
select xmltype(cursor(select a.naam, a.afdnr,
                  cursor(select s.naam, s.persnr from stafleden s
                        where ziekhnr=a.ziekhnr
                        and afdnr=a.afdnr) as stafleden
                from afdelingen a
                where ziekhnr = 20
                and afdnr=1)) as afdeling
from dual
/
```

3. **optioneel:** Haal van elke functie het gemiddelde salaris, het maximale salaris, het minimale salaris en het aantal werknemers op. De uitvoer van elke functie dient er als volgt uit te zien:

```
<functie naam="ANALIST">
  <column name="gemiddeld salaris">3950</column>
  <column name="maximaal salaris">4000</column>
  <column name="minimaal salaris">3900</column>
  <column name="aantal werknemers">2</column>
</functie>
```

```
select
xmlelement("functie",
           xmlattributes(functie as "naam"),
           xmlcolattval(round(avg(sal),2) as "gemiddeld salaris"),
           xmlcolattval(max(sal) as "maximaal salaris"),
           xmlcolattval(min(sal) as "minimaal salaris"),
           xmlcolattval(count(1) as "aantal werknemers")
         ).extract('/*') as "functies"
from werknemers
group by functie;
```

4. **optioneel:** Zorg dat dezelfde gegevens als van opgave 2 als volgt worden weergegeven, door gebruik te maken van SQL/XML functies

```
afdelingen
--------------------------------------
<afdelingen>
  <afdeling afdnr="1">
    <stafleden>
      <staflid persnr="8632">
        <naam>Riksen G.</naam>
      </staflid>
      <staflid persnr="5342">
        <naam>Coolen R.</naam>
      </staflid>
    </stafleden>
  </afdeling>
</afdelingen>

select
xmlelement ("afdelingen",
     xmlagg (xmlelement ("afdeling", xmlattributes (afdnr as "afdnr"),
                xmlelement ("stafleden",
                   xmlagg( xmlelement ("staflid",
                             xmlattributes (persnr as "persnr"),
                             xmlelement ("naam",naam)
                           )
                    )
                )
      )
).extract ('/*')
from stafleden s where ziekhnr = 20 and afdnr =1
group by ziekhnr, afdnr
/
```

Appendix A Uitwerkingen

Appendix A Uitwerkingen

Opdrachten hoofdstuk 4

1. In uw home directory staat xml_ziekenhuizen_vw.sql, waarmee de xmltype view xml_ziekenhuizen_vw kan worden gemaakt. Voer dit script uit en bekijk de structuur van deze view.

   Voor het bekijken van de structuur kunnen we op het data-tabblad naast een rij op de drie puntjes klikken, waarna het XML Data window wordt getoond.

2. Toon van xml_ziekenhuizen_vw alle stafleden uit de avonddienst met een salaris groter dan 18000.

```
select xmlquery('//staflid[sal>18000][dienst="A"]'
                passing object_value returning content) as stafleden
from xml_ziekenhuizen_vw
where xmlexists('//staflid[sal>18000][dienst="A"]' passing object_value);

STAFLEDEN
----------------------------
<staflid persnr="2342">
   <naam>Keyzer W.</naam>
   <functie>Assistent</functie>
   <dienst>A</dienst>
   <sal>18300</sal>
</staflid>
<staflid persnr="9835">
   <naam>Fleskes H.</naam>
   <functie>Verpleegster</functie>
   <dienst>A</dienst>
   <sal>19400</sal>
</staflid>
```

3. Toon het staflid met persnr 2342. Voeg daarin een extra element voornaam toe, met als waarde Wietze.

```
select insertchildxmlafter(xmlquery('//staflid[@persnr=2342]' passing
    object_value returning content),
   'staflid','naam',xmlelement("voornaam",'Wietze'))
from xml_ziekenhuizen_vw
where xmlexists('//staflid[@persnr=2342]' passing object_value);

STAFLID
----------------------
<staflid persnr="2342">
   <naam>Keyzer W.</naam>
   <voornaam>Wietze</voornaam>
   <functie>Assistent</functie>
   <dienst>A</dienst>
   <sal>18300</sal>
</staflid>
```

Appendix A Uitwerkingen

4. Maak een tabel xml_ziekenhuizen aan op basis van xml_ziekenhuizen_vw.

```
create table xml_ziekenhuizen of xmltype
as select object_value from xml_ziekenhuizen_vw;
```

5. Voeg daaraan met behulp van SQL*Loader de gegevens toe uit het bestand ziekenhuis30.xml.

controlfile:

```
load data
infile *
replace
into table xml_ziekenhuizen APPEND xmltype(xmldata)
fields terminated by ","
(lob_file FILLER char,
 xmldata LOBFILE(lob_file) TERMINATED BY EOF
 )
begindata
h:\oraclexml\ziekenhuis30.xml
```

Aanroepen SQL*Loader:

```
H:\oraclexml>sqlldr <gebruikersnaam>/vh_cursus@nwg11 opdracht.ctl
log=opdracht.log
```

6. Toon van het ziekenhuis met ziekhnr 25 alle afdelingen, zonder de stafleden.
Tip: maak gebruik van deleteXML(), met daarbinnen XMLQuery(). Kijk eventueel naar een vergelijkbaar voorbeeld bij updateXML().

```
select deletexml(
          xmlquery('//afdelingen' passing object_value returning content)
          , '//stafleden') as afdelingen
from xml_ziekenhuizen_vw
where xmlexists('//ziekenhuis[@ziekhnr=25]' passing object_value)

AFDELINGEN
----------------------------------------
<afdelingen>
   <afdeling afdnr="4">
      <naam>Hartafdeling</naam>
      <totbed>55</totbed>
      <patienten>
         <patient patnr="50333">
            <naam>Horst E.</naam>
            <plaats>Utrecht</plaats>
            <gebdat>1964-04-12</gebdat>
            <mv>M</mv>
            <ziekfnr>1232988</ziekfnr>
         </patient>
      </patienten>
   </afdeling>
   <afdeling afdnr="2">
      <naam>Kinder</naam>
      <totbed>24</totbed>
      <patienten/>
   </afdeling>
</afdelingen>
```

Appendix A Uitwerkingen

Opdrachten hoofdstuk 5

1. a. Geef op basis van een XQuery view op de tabel OX_STAFLEDEN alle functies op alfabetische volgorde. De uitvoer dient er als volgt uit te zien (zonder pretty-print):

```
<functies>
   <functie naam="Assistent"></functie>
   <functie naam="Verpleegster"></functie>
   <functie naam="Zaalknecht"></functie>
</functies>
```

b. Geef ook de namen van de stafleden binnen de de betreffende functie-elementen:

```
<functies>
   <functie naam="Assistent">
      <naam>Arends R.</naam>
      <naam>Keyzer W.</naam>
      <naam>Mensink C.</naam>
   </functie>
   ....
</functies>
```

```
select xmlquery(
'<functies>{
     let $stafleden := ora:view("OX_STAFLEDEN")/ROW
     for $functie in distinct-values($stafleden/FUNCTIE)
     order by $functie
     return <functie naam="{$functie}">{
            for $naam in $stafleden[FUNCTIE=$functie]/NAAM
            order by $naam
            return <naam>{$naam/text()}</naam>
      }</functie>
  }</functies>'  returning content) as functies
from dual;
```

2. a. Zet via FTP het bestand patienten.xml in XDB in de map ziekenhuizen. Bekijk de structuur van dit bestand
b. Toon alle gegevens van de patiënten met behulp van XMLTable, alsof het een relationele tabel is (zonder XML-opmaak, in rijen en kolommen).

a. Open het bestand bijvoorbeeld in XMLBlueprint, en gebruik "Save to FTP or WebDAV server" om het bestand in XML DB op te slaan. Kies daarbij bovenaan het pad, en onder de filenaam.

b.
```
select * from xmltable('
  for $rij in
doc("/home/cursisten/<gebruikersnaam>/ziekenhuizen/patienten.xml")//patient
  return $rij'
  columns patnr number path '@patnr',
          ziekfnr number path '@ziekfnr',
          naam varchar2(20) path 'naam',
          plaats varchar2(20) path 'plaats',
          gebdat date path 'gebdat',
          mv varchar2(1) path 'mv')
```

3. Toon van de tabel xml_ziekenhuizen de namen van de ziekenhuizen, de afdelingen en de stafleden. Toon daarbij ook de afdelingen zonder stafleden. Maak gebruik van XMLTable().

```
select zie.ziekenhuis, afd.afdeling, sta.staflid
from xml_ziekenhuizen z,
    xmltable('/ziekenhuis' passing z.object_value
             columns ziekenhuis varchar2(20) path 'naam',
                     afdxml xmltype path 'afdelingen/afdeling') zie,
    xmltable('/afdeling' passing afdxml
             columns afdeling varchar2(20) path 'naam',
                     staxml xmltype path 'stafleden/staflid')  afd,
    xmltable('/staflid' passing staxml
             columns staflid varchar2(20) path 'naam') (+) sta
```

4. Toon van xml_ziekenhuizen_vw alle stafleden uit de avonddienst met een salaris groter dan 18000, op volgorde van naam.

```
select xmlquery('for $staflid in //staflid[dienst="A"][sal>18000]/naam
                 order by $staflid
                 return $staflid' passing xmldata returning content)
from (select xmlagg(object_value) xmldata
      from xml_ziekenhuizen_vw)
```

Appendix A Uitwerkingen

Opdrachten hoofdstuk 6

1. Maak in de folder ziekenhuizen nieuwe bronnen aan met de naam ziekenhuis<ziekhnr>.xml . Deze bronnen verwijzen naar de rijen in xml_ziekenhuizen_vw.

```
declare
  data ref xmltype;
  dummy boolean;
begin
 for rij in (select ziekhnr from ox_ziekenhuizen) loop
   select make_ref(xml_ziekenhuizen_vw, rij.ziekhnr)
   into data from dual;
    dummy:= dbms_xdb.createResource
    ('/home/cursisten/nwg802/ziekenhuizen/ziekenhuis'||rij.ziekhnr||'.xml'
    , data, false);
  end loop;
 end;
 /

commit;
```

2. Maak een folder /home/cursisten/<gebruikersnaam>/hospitaals aan als link naar de folder ziekenhuizen.

```
begin
  dbms_xdb.link('/home/cursisten/nwg802/ziekenhuizen',
  '/home/cursisten/nwg802','hospitaals');
end;
```

3. Maak de view folder_view die alle kolommen van path_view bevat van de folders onder /home/cursisten/<gebruikersnaam>.

```
create or replace view folder_view
as select * from path_view
where xmlexists('declare namespace ns =
"http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
/ns:Resource[@Container = xs:boolean("true")]'
passing res)
and under_path(res, '/home/cursisten/nwg802')=1

create or replace view succeeded.

select path from folder_view;

/home/cursisten/nwg802/boek
/home/cursisten/nwg802/ziekenhuizen
/home/cursisten/nwg802/hospitaals
/home/cursisten/nwg802/kantoren
/home/cursisten/nwg802/links
/home/cursisten/nwg802/links/x
/home/cursisten/nwg802/links/x/y
/home/cursisten/nwg802/stylesheets
/home/cursisten/nwg802/x
/home/cursisten/nwg802/x/y
```

4. Onder de map /sys staan bronnen die door XML DB intern worden gebruikt. Haal alle relatieve paden onder /sys op met een diepte van 5.

```
select path(1) from path_view
where under_path(res,'/sys',1)=1
and depth(1)=5;

PATH(1)
--------------------------------------------------------
schemas/PUBLIC/www.opengis.net/gml/feature.xsd
schemas/PUBLIC/www.opengis.net/gml/geometry.xsd
schemas/PUBLIC/www.w3.org/1999/csx.xlink.xsd
schemas/PUBLIC/www.w3.org/1999/xlink
schemas/PUBLIC/www.w3.org/1999/xlink.xsd
schemas/PUBLIC/www.w3.org/2001/XInclude.xsd
schemas/PUBLIC/www.w3.org/2001/csx.XInclude.xsd
schemas/PUBLIC/www.w3.org/2001/csx.xml.xsd
schemas/PUBLIC/www.w3.org/2001/xml.xsd
schemas/PUBLIC/xmlns.oracle.com/ord/dicom
schemas/PUBLIC/xmlns.oracle.com/ord/meta
schemas/PUBLIC/xmlns.oracle.com/rlmgr/rclsprop.xsd
schemas/PUBLIC/xmlns.oracle.com/rlmgr/rulecond.xsd
schemas/PUBLIC/xmlns.oracle.com/spatial/georaster
schemas/PUBLIC/xmlns.oracle.com/streams/schemas
schemas/PUBLIC/xmlns.oracle.com/xdb/XDBFolderListing.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/XDBResConfig.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/XDBResource.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/XDBSchema.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/XDBStandard.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/acl.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/csx.xmltr.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/dav.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/log
schemas/PUBLIC/xmlns.oracle.com/xdb/stats.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/xdbconfig.xsd
schemas/PUBLIC/xmlns.oracle.com/xdb/xmltr.xsd
schemas/PUBLIC/xmlns.oracle.com/xs/aclids.xsd
schemas/PUBLIC/xmlns.oracle.com/xs/dataSecurity.xsd
schemas/PUBLIC/xmlns.oracle.com/xs/principal.xsd
schemas/PUBLIC/xmlns.oracle.com/xs/roleset.xsd
schemas/PUBLIC/xmlns.oracle.com/xs/securityclass.xsd
```

Verwijder de map links inclusief inhoud met behulp van dbms_xdb.

```
begin
  dbms_xdb.deleteresource('/home/cursisten/nwg802/links',
  dbms_xdb.DELETE_RECURSIVE);
end;

commit;
```

Appendix A Uitwerkingen

Opdrachten hoofdstuk 7

In deze opdracht zult u een XML Schema genereren voor kantoren op basis van objecttypen. Dit schema zal een andere structuur beschrijven dan we kennen van xml_kantoren en xml_kantoren_vw. Uit objecttypen kunnen we namelijk geen attributen genereren.

1. Maak de objecttypen aan door het script create_types.sql te draaien.

   De inhoud van create_types.sql is:

```
create or replace type "werknemer_type" as object ("persnr" number, "mgr" number,
          "naam" varchar2(10), "sal" number,
          "toeslag" number, "functie" varchar2(9));
/

create or replace type "werknemer_tab" as varray(50) of "werknemer_type";
/

create or replace type "werknemers_type" as object ("werknemer" "werknemer_tab");
/

create or replace type "kantoor" as object ("kantnr" number, "naam" varchar2(15),
                          "plaats" varchar2(10), "werknemers"
"werknemers_type");
/
```

2. Bekijk de inhoud van het script create_types en bepaal op basis van welk type het schema moet worden gegenereerd.

   Het overkoepelende type is "kantoor"

3. Genereer het schema en registreer dit onder de url http://www.5hart.com/schema/kantoren.xsd.

```
declare
  xmldata xmltype;
begin
  select dbms_xmlschema.generateschema(user, 'kantoor') into xmldata
  from dual;
  dbms_xmlschema.registerschema(
      'http://www.5hart.com/schema/kantoren.xsd',
      xmldata,  gentypes => false);
end;
/
```

4. Bepaal welke default tabel is aangemaakt. Maak voor deze tabel het synoniem xsd_kantoren aan.

```
select table_name
from user_xml_tables
where table_name like 'kantoor%';

create synonym xsd_kantoren for <tabelnaam>;
```

5. Pas de bestanden kantoor10.xml en kantoor20.xml aan, zodat deze gebruik maken van het geregistreerde schema.

   Binnen elk kantoor element worden de volgende attributen toegevoegd:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.5hart.com/schema/kantoren.xsd"
```

6. Probeer deze bestanden via FTP of WebDaV toe te voegen aan de submap opdrachten. Probeer eventuele fouten in de bestanden zelf te verbeteren.

   Kantoor20.xml bevat een fout: het element regio komt niet voor in het schema. Deze dient te worden verwijderd.

7. Controleer of de bestanden zijn toegevoegd aan xsd_kantoren.

```
select extract(value(k),'/kantoor/naam')
from xsd_kantoren k;

EXTRACT(VALUE(K),'/KANTOOR/NAAM')
--------------------------------------------------------
<naam>BOEKHOUDING</naam>
<naam>ONDERZOEK</naam>
```

Appendix A Uitwerkingen

Opdrachten hoofdstuk 8

We kunnen op een XML view een instead of trigger zetten, zodat ingevoerde xml-gegevens automatisch in de onderliggende relationele tabellen worden verwerkt. Wij doen dit voor de volgende view (dit staat in create_xml_werkn_vw.sql):

```
create or replace view xml_werknemers_vw of xmltype
with object id (XMLCast(XMLQuery('/ROW/PERSNR/text()'
                passing object_value returning content) as integer))
as select column_value
from xmltable('for $rij in
               ora:view("OX_WERKNEMERS")/ROW
               return $rij');
```

1. Voer dit statement uit en controleer de inhoud van deze view

2. Maak vervolgens de insteadof trigger aan. Een deel van de code staat alvast in create_xml_werkn_trg.sql.

```
create or replace trigger xml_werkn_vw_trg
  instead of insert or update on xml_werknemers_vw
  for each row
declare
   ctx DBMS_XMLStore.ctxType;
   rijen number;
   xmldoc xmltype := :new.sys_nc_rowinfo$;          -- de nieuwe rij
 begin
   ctx := DBMS_XMLStore.newContext('OX_WERKNEMERS'); -- maak een context handle
   if inserting then
       rijen := DBMS_XMLStore.insertXML(ctx,xmlDoc); -- insert het document
   elsif updating then
       DBMS_XMLStore.setkeycolumn(ctx,'PERSNR');
       rijen := DBMS_XMLStore.updateXML(ctx,xmlDoc); -- update het document
   end if;
   DBMS_XMLStore.closeContext(ctx);                 -- sluit de context handle
end;
/
```

3. Voeg werkn1.xml en werkn2.xml via WebDaV of FTP toe aan uw map opdrachten binnen XML DB. Voeg vervolgens werkn1.xml toe aan xml_werknemers_vw, en controleer daarna de inhoud van zowel de view als de onderliggende tabel.

   Via XMLBlueprint opslaan als:

```
/home/cursisten/<gebruikersnaam>/opdrachten/werkn1.xml
/home/cursisten/<gebruikersnaam>/opdrachten/werkn2.xml

insert into xml_werknemers_vw
select xdburitype(
  '/home/cursisten/<gebruikersnaam>/opdrachten/werkn1.xml').getXml()
from dual;

select extract(object_value,'/*') from xml_werknemers_vw;
select * from ox_werknemers;
```

4. Haal de functie en het salaris van de werknemer met persnr 8222 op. Wijzig de gegevens van deze werknemer vervolgens met behulp van werkn2.xml. Controleer de view en de onderliggende tabel daarna opnieuw.

```
select functie, sal from ox_werknemers where persnr=8222;

FUNCTIE   SAL
--------- ----------------------
KLERK     2300


update xml_werknemers_vw
set object_value =
  xdburitype('/home/cursisten/<gebruikersnaam>/opdrachten/werkn2.xml').getXml()
where xmlexists('/ROW[PERSNR=8222]' passing object_value);


FUNCTIE   SAL
--------- ----------------------
ANALIST   2350
```