DBMS_XMLDOM

XML Document Object Model API

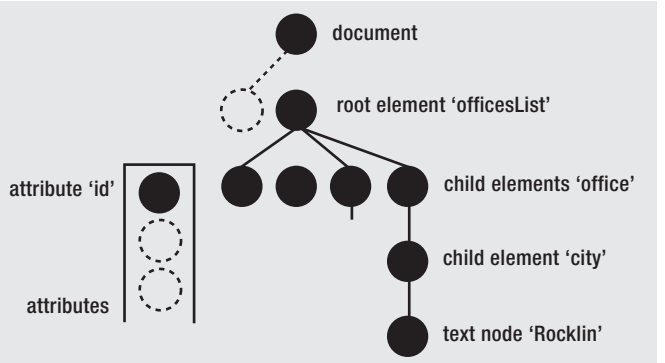
Dit is het vierde artikel in een reeks van artikelen over de XML ondersteuning in de Oracle database. In deze reeks introduceren Erwin Groenendal en Marc Vahsen in detail, en aan de hand van veel voorbeelden, de mogelijkheden en toepassingen van XML technologie op het Oracle platform. Aangenomen wordt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

In het eerste artikel is het gebruik van SQL/XML functies voor het genereren van XML data vanuit SQL behandeld, en in het tweede artikel ("XMLType, Native XML datatype in de database") het opslaan, bevragen en manipuleren van XML-data. De Oracle XDB Repository ("filesysteem in de database") was het onderwerp van het derde artikel. In dit artikel wordt de DBMS_XMLDOM package behandeld, Oracle's PL/SQL implementatie van de Document Object Model (DOM) API. Met behulp van deze API kunnen XML-documenten opgebouwd worden die niet goed met behulp van SQL/XML samengesteld kunnen worden. Er zullen nog artikelen volgen over het editen van XML-documenten, werken vanuit Java met Oracle XDB en de introductie van XQuery in release van 2 van Oracle 10g.

Document Object Model

Het Document Object Model (DOM) is een model voor het (exact) beschrijven van HTML en XML-documenten. Dit, door de W3C organisatie beheerde, model is het beste uit te leggen aan de hand van een voorbeeld. Hiervoor wordt het onderstaande XML-document gebruikt. In figuur 1 is de DOM instantie van dit document weergegeven.

```
<officesList>
          <office id="1">
                    <city>Costa Mesa</city>
          </office>
          <office id="2">
                    <city>El Segundo</city>
          </office>
          <office id="3">
                    <city>San Rafael</city>
          </office>
          <office id="4">
                    <city>Rocklin</city>
          </office>
</officesList>
<< einde kader met computercode >>
```



Figuur 1. Voorbeeld van een DOM-instantie. Met deze W3C-standaard kunnen HTML- en XML-documenten exact beschreven worden

De DOM instantie bestaat uit een hiërarchie van "nodes", de bolletjes in bovenstaand plaatje. De root node representeert het document zelfs. Deze node bestaat altijd, zelf als het document geen inhoud heeft, en wordt in XPath aangeduid met '/'. Onder de document node bevindt zich de node voor het root element 'officesList'. Vóór het root element zouden zich ook nog andere nodes kunnen bevinden. In het voorbeeld is dit niet het geval, maar nodes voor de XML-declaratie (`<?xml version="1.0"?>`) of een processing instruction voor het specificeren van een XSLT stylesheet transformatie zouden zich vóór het root element bevinden (op de plaats van de gestippelde node in bovenstaande plaatje). Onder het root element bevinden zich de nodes voor de 'office' child elementen, die ieder weer een child node hebben voor element 'city'. Het element 'city' heeft in het XML-document de naam van de stad als content. Deze tekst wordt in de DOM instantie als een aparte node gerepre-

senteerd. De element node 'city' bevat zelf dan ook geen waarde.
Bij child elementen is de volgorde van belang. De child elementen van 'officesList' bevinden zich dan ook in een zogenaamde "node list". Bij attributen is de volgorde niet van belang. Attributen worden op naam aangewezen (en niet op positie). De attributen van een element bevinden zich in een "named node map". In het voorbeeld bevat de named node map van een 'office' element maar één attribuut. Maar, zoals de gestippelde bolletjes al aangeven in het plaatje, kunnen dit er dus meerdere zijn. Een attribuut node bevat wel zelf de waarde. Er is dus geen aparte node voor de waarde van een attribuut.

Document Object Model API

De Document Object Model (DOM) API is een door de W3C gedefinieerde API (interface) voor het opbouwen, doorlopen en wijzigen van een DOM instantie. Van deze API bestaan 3 "levels", met telkens uitgebreidere functionaliteit. DBMS_XMLDOM is Oracle's PL/SQL implementatie van (level 1) van de DOM API.

DHTML

De DOM API is de basis voor DHTML. Met DHTML kan de structuur en inhoud van een HTML document programmatisch, dynamisch – vandaar de 'D' in DHTML – gewijzigd worden. Deze wijzigingen worden uitgevoerd via een implementatie van de DOM API. Bij HTML pagina's die in een browser getoond worden, kan gebruik worden gemaakt van een DOM API geïmplementeerd in JavaScript. Het is dan bijvoorbeeld mogelijk om dynamisch extra input velden in een pagina te plaatsen wanneer een gebruiker een checkbox aanvinkt.

XML

Voor XML wordt de DOM API gebruikt voor:

- het programmatisch construeren van XML-documenten,
- het onttrekken van informatie uit XML-documenten en
- het wijzigen van XML-documenten.

In dit artikel worden uitgebreide codevoorbeelden geven van deze drie toepassingen.

DBMS_XMLDOM

De DBMS_XMLDOM package wordt "vergezeld" door DBMS_XMLPARSER voor het parsen van XML-documenten en DBMS_XSLPROCESSOR voor het uitvoeren van XSLT transformaties. Deze drie packages maakten eerst deel uit van de XML Developer's Kit en waren toen gebaseerd op Java Stored Procedures. Vanaf release 2 van Oracle9i zijn deze packages gebaseerd op C, meegecompileerd met de kernel van Oracle en een stuk sneller.

Documentatie

De DBMS_XMLDOM package is bij versie 9i van de database gedocumenteerd in de "XML API Reference – XDK and Oracle XDB". Bij 10g is de documentatie van deze package verplaatst naar de "PL/SQL Supplied Packages and Types Reference". De package zelf is overigens niet gewijzigd.
In de documentatie wordt wel eens naar DBMS_XMLDOM verwezen als een XMLType API. Dit is niet correct; DBMS_XMLDOM maakt geen gebruik van XMLType, maar van een eigen datatype voor het representeren van een XML-document.

Simple API for XML

Voor het werken met hele grote XML-documenten is een waarschuwing op zijn plaats. Een DOM instantie wordt namelijk in zijn geheel in memory geplaatst. Hoewel er wel implementaties zijn waarbij slechts dat deel van de DOM instantie dat daadwerkelijk benaderd wordt in memory wordt geladen ("lazy DOM"), is voor het werken met hele grote XML-documenten de Simple API for XML (SAX) aan te raden. De SAX wordt ondersteund in de XML Developer's Kit (XDK) voor Java, C en C++, maar niet in de database zelf.

Construeren van XML-documenten

Met het onderstaande codevoorbeeld wordt het volgende XML-document opgebouwd:

```
<office>
  <city>Costa Mesa</city>
</office>
<< einde kader met computercode >>

<< begin kader met computercode >>
declare

   l_document        dbms_xmldom.DOMDocument;
   l_root_element    dbms_xmldom.DOMElement;
   l_element         dbms_xmldom.DOMElement;
   l_text_node       dbms_xmldom.DOMText;
   l_node            dbms_xmldom.DOMNode;

   l_buffer          varchar2(1000);

begin

   -- 1: create a new XML document

   l_document := dbms_xmldom.newDOMDocument;

   -- 2: create root element 'office'

   l_root_element := dbms_xmldom.createElement(l_document, 'office');

   -- 3: append root element (node) as a child node to the document
(node)
```

```
   l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_document)
             ,                       dbms_xmldom.makeNode(l_root_
element)
             );

   -- 4: create element 'city'

   l_element := dbms_xmldom.createElement(l_document, 'city');

   -- 5: append element (node) as a child node to the root element
(node)

   l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_root_ele-
ment)
             ,                       dbms_xmldom.makeNode(l_element)
             );

   -- 6: create text node

   l_text_node := dbms_xmldom.createTextNode(l_document, 'Costa Mesa');

   -- 7: append text node as a child node to the 'city' element (node)

   l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_element)
             ,                       dbms_xmldom.makeNode(l_text_
node)
             );

   -- show the constructed XML document

   dbms_xmldom.writeToBuffer(l_document, l_buffer);
   dbms_output.put_line(l_buffer);

end;
```

- Stap 1- Met behulp van 'newDOMDocument' wordt er een nieuw document aangemaakt. In feite wordt met deze stap een node van het sub-type 'document' aangemaakt.
- Stap 2- Hier wordt de element node 'office' aangemaakt. Het aanmaken van nodes gebeurt altijd in de context van het document (met uitzondering van de 'document' node zelf). De eerste parameter van 'createElement' is dan ook (een pointer naar) het document.
- Stap 3- De element node 'office' moet gekoppeld worden aan de document node (en wordt daardoor het root element van het document). Dit wordt gedaan met 'appendChild'. Omdat deze functie nodes als parameters heeft, moeten de document node en de element node met 'makeNode' gecast worden naar hun super-type 'node'.

Aanmaken element 'city':

- Stap 4- Koppelen element 'city' aan (parent) element 'office'.
- Stap 5- Het element 'city' heeft als inhoud de naam van de stad. Hiervoor moet een aparte node worden aangemaakt van het type 'text'. Dit wordt gedaan met 'createTextNode'.
- Stap 6- De text node wordt gekoppeld aan de parent element node 'city'.

Tenslotte wordt het opgebouwde XML-document getoond in SQL*Plus.

Attributen

In het vorige voorbeeld zijn alleen elementen gebruikt. Hieronder wordt dit codevoorbeeld uitgebreid met het toevoegen van attributen. Het XML-document ziet er dan als volgt uit:

```
<office id="1">
  <city>Costa Mesa</city>
</office>
```

```
   ...
   l_attributes     dbms_xmldom.DOMNamedNodeMap;
   l_attribute      dbms_xmldom.DOMAttr;

begin

   ...

   -- 2.1: get attributes of element 'office'

   l_attributes :=   dbms_xmldom.getAttributes(dbms_xmldom.makeNode(l_
root_element));

   -- 2.2: create attribute 'id'

   l_attribute := dbms_xmldom.createAttribute(l_document, 'id');

   -- 2.3: set value of attribute

   dbms_xmldom.setValue(l_attribute, to_char(1));

   -- 2.4: "place" attribute in attributes of element 'office'

   l_node := dbms_xmldom.setNamedItem(l_attributes
             ,                        dbms_xmldom.makeNode(l_attri-
bute)
             );

   ...

end;
```

- Stap 2.1- Eerst worden de attributen van het zojuist aangemaakte element 'office' opgehaald. In feite wordt er een pointer verkregen naar een nog lege named node map waarin de attribute nodes geplaatst kunnen worden.
- Stap 2.2- Aanmaken attribuut 'id'.
- Stap 2.3- Toekennen van een waarde aan het attribuut. Attribuutwaarden in de DOM API zijn altijd van het datatype 'string'.
- Stap 2.4- Het attribuut wordt in de named node map

geplaatst. Deze actie is vergelijkbaar met koppelen van een child element aan een parent element. Als deze acties niet uitgevoerd worden zal het attribuut of element geen deel uitmaken van het XML-document.

Modulair gebruik DOM API

Stel dat het onderstaande XML-document opgebouwd moet worden, waarin een lijst (collection) van kantoren is opgenomen.

```
<officesList>
<office id="1">
<city>Costa Mesa</city>
  </office>
<office id="2">
<city>El Segundo</city>
</office>
<office id="3">
<city>San Rafael</city>
</office>
<office id="4">
<city>Rocklin</city>
</office>
</officesList
```

Voor het construeren van het 'office' element kan op basis van het eerder gegeven voorbeeld een stored function gemaakt worden. Deze functie heeft, naast parameters voor de id en de naam van het kantoor, een parameter waarin de pointer naar het document meegegeven wordt. Elementen en attributen moeten namelijk, zoals eerder aangegeven is, in de context van een document aangemaakt worden. Dit document moet hetzelfde document zijn als waarin de overige elementen aangemaakt gaan worden. De stored function retourneert een volledig 'office' element.

```
create or replace
function get_office
(         p_document    in    dbms_xmldom.DOMDocument
,         p_id          in    number
,         p_name        in    varchar2
)
return dbms_xmldom.DOMElement
is

    l_root_element    dbms_xmldom.DOMElement;
    l_element         dbms_xmldom.DOMElement;
    l_text_node       dbms_xmldom.DOMText;
    l_node            dbms_xmldom.DOMNode;

    l_attributes      dbms_xmldom.DOMNamedNodeMap;
    l_attribute       dbms_xmldom.DOMAttr;

begin

    -- create root element 'office'

    l_root_element := dbms_xmldom.createElement(p_document, 'office');

    -- create element 'city'

    l_element := dbms_xmldom.createElement(p_document, 'city');

    -- append element (node) as a child node to the root element (node)

    l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_root_ele-
ment)
              ,                         dbms_xmldom.makeNode(l_element)
              );

    -- get attributes of element 'office'

    l_attributes := dbms_xmldom.getAttributes(dbms_xmldom.makeNode(l_
root_element));

    -- create attribute 'id'

    l_attribute := dbms_xmldom.createAttribute(p_document, 'id');

    -- set value of attribute

    dbms_xmldom.setValue(l_attribute, to_char(p_id));

    -- "place" attribute in attributes of element 'office'

    l_node := dbms_xmldom.setNamedItem(l_attributes
              ,                          dbms_xmldom.makeNode(l_attri-
bute)
              );

    -- create text node

    l_text_node := dbms_xmldom.createTextNode(p_document, p_name);

    -- append text node as a child node to the 'city' element (node)

    l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_element)
              ,                         dbms_xmldom.makeNode(l_text_
node)
              );

    -- return the constructed element

    return l_root_element;

end;
```

De bovenstaande stored function kan als volgt gebruikt worden:

```
declare

    cursor c_offices
    is
    select id
    ,      name
    from   acme_offices;
```

```
    l_document        dbms_xmldom.DOMDocument;
    l_root_element    dbms_xmldom.DOMElement;
    l_element         dbms_xmldom.DOMElement;
    l_text_node       dbms_xmldom.DOMText;
    l_node            dbms_xmldom.DOMNode;

    l_attributes      dbms_xmldom.DOMNamedNodeMap;
    l_attribute       dbms_xmldom.DOMAttr;

    l_buffer          varchar2(1000);

begin

    -- 1: create a new XML document

    l_document := dbms_xmldom.newDOMDocument;

    -- 2: create root element 'officesList'

    l_root_element := dbms_xmldom.createElement(l_document, 'office-
sList');

    -- 3: append root element (node) as a child node to the document
(node)

    l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_document)
              ,                          dbms_xmldom.makeNode(l_root_
element)
              );

    -- 4: add 'office' elements

    for r_office in c_offices loop

       -- 4.1: get element 'office'

       l_element := get_office(l_document, r_office.id, r_office.name);

       -- 4.2: append element (node) as a child node to the 'office-
sList' element (node)

       l_node := dbms_xmldom.appendChild(dbms_xmldom.makeNode(l_root_
element)
                 ,                         dbms_xmldom.makeNode(l_ele-
ment)
                 );

    end loop;

    -- show the constructed XML document

    dbms_xmldom.writeToBuffer(l_document, l_buffer);
    dbms_output.put_line(substr(l_buffer, 1, 255));

end;
```

- Stap 1- Aanmaken nieuw document.
- Stap 2- Aanmaken element 'officesList'.
- Stap 3- Koppelen element 'officesList' aan document node. Hiermee wordt 'officesList' het root element.
- Stap 4- Toevoegen van de met de cursor geselecteerde kantoren als child elementen van 'officesList'.
- Stap 4.1- Construeren 'office' element met behulp van de eerder beschreven stored function 'get_office'.
- Stap 4.2- Koppelen van het 'office' element aan parent 'officesList'.

Tenslotte wordt het opgebouwde XML-document getoond in SQL*Plus.

Onttrekken van informatie

Naast het opbouwen van een XML-document is de tweede belangrijke toepassing van de DOM API het onttrekken van informatie uit een bestaande XML-document. Om dit te kunnen doen zal een XML-document eerst "ingelezen" moeten worden om een instantie van een DOM instantie van het document te verkrijgen. Dit kan door de file in te lezen en te parsen met behulp van DBMS_XMLPARSER. Hierbij kan dan tegelijkertijd een validatie tegen een DTD plaatsvinden. Maar er kan ook gebruik worden gemaakt van XMLType. DBMS_XMLDOM biedt namelijk de mogelijkheid om een DOM instantie te maken van een XMLType waarde. Met het onderstaande statement wordt een XML document in de Oracle XDB Repository omgezet in een DOM instantie.

```
l_document := dbms_xmldom.newDOMDocument(xdburitype('/public/temp/cali-
fornia_offices.xml').getxml());
```

Overigens is er, helaas, geen XMLType constructor die een XMLType waarde aanmaakt op basis van een DOM instantie. Bij het ontrekken van informatie uit een XML-document is het van belang om vast te stellen of de structuur van het XML-document bekend is. Als de structuur bekend is, bijvoorbeeld omdat het document gevalideerd, is kunnen er aannames gemaakt worden bij het "doorlopen" van het XML-document. Indien de structuur onbekend is, is er meer sprake van het "verkennen" van het document. Net als bij het opbouwen van een XML-document wordt bij het onttrekken van informatie ook veel gebruik gemaakt van hele eenvoudige operaties. Bijvoorbeeld 'getChildNodes' voor het verkrijgen van een node list met de child nodes van een bepaalde node of 'getValue' voor het opvragen van de waarde van een text-node of attribuut. Daarnaast bestaan er krachtigere functies zoals 'getElementsByTagName' en kan gebruik worden gemaakt van DBMS_XSLPROCESSOR voor het selecteren van nodes via XPath expressies.
In het codevoorbeeld op de volgende pagina wordt uit het 'officesList' XML-document voor iedere kantoor de id en de naam onttrokken.

```
declare

    l_document        dbms_xmldom.DOMDocument;
    l_root_element    dbms_xmldom.DOMElement;
    l_child_nodes     dbms_xmldom.DOMNodeList;
    l_office_node     dbms_xmldom.DOMNode;
    l_node            dbms_xmldom.DOMNode;
    l_text_node       dbms_xmldom.DOMText;
    l_attributes      dbms_xmldom.DOMNamedNodeMap;

begin

    -- 1: create new DOM document based on document in XDB repository

    l_document := dbms_xmldom.newDOMDocument(xdburitype('/public/temp/
california_offices.xml').getxml());

    -- 2: get the root element ('officesList')

    l_root_element := dbms_xmldom.getDocumentElement(l_document);

    -- 3: get child nodes of root element

    l_child_nodes := dbms_xmldom.getChildNodes(dbms_xmldom.makeNode(l_
root_element));

    -- 4: for each child ...

    for i in 0..(dbms_xmldom.getLength(l_child_nodes) - 1) loop

       -- 4.1: get the node (element 'office')

       l_office_node := dbms_xmldom.item(l_child_nodes, i);

       -- 4.2: get attributes of element 'office'

       l_attributes := dbms_xmldom.getAttributes(l_office_node);

       -- 4.3: get attribute 'id'

       l_node := dbms_xmldom.getNamedItem(l_attributes, 'id');

       -- 4.4: get value of attribute

       dbms_output.put_line(dbms_xmldom.getNodeValue(l_node));

       -- 4.5: get the first child node (element 'city')

       l_node := dbms_xmldom.getFirstChild(l_office_node);

       -- 4.6: get the text node (the first child node)

       l_text_node := dbms_xmldom.makeText(dbms_xmldom.getFirstChild(l_
node));

       dbms_output.put_line(dbms_xmldom.getNodeValue(dbms_xmldom.
makeNode(l_text_node)));

    end loop;

end;
```

- Stap 1- Maak een nieuw document (DOM instantie) op basis van een XML-document in de Oracle XDB Repository.
- Stap 2- Hier wordt met behulp van 'getDocumentElement' een pointer naar de root node verkregen. Het gebruik van deze functie is aan te raden boven het gebruik van 'getFirst-Child' om de element node onder de document node te bepalen. Indien het document namelijk een XML declaration zou bevatten (<?xml version="1.0"?>) of een processing instruction voor het specificeren van een XSLT stylesheet transformatie, dan is de root node niet de eerste child node van de document node. De functie 'getDocumentElement' geeft altijd de root node terug.
- Stap 3- Haal child nodes op van root element 'officesList'.
- Stap 4- Doorloop de child elementen. De child elementen bevinden zich in een node list. De index van deze lijst start bij 0. Het aantal elementen in de lijst kan bepaald worden met 'getLength'.
- Stap 4.1- Verkrijg een pointer naar het "zoveelste" child element 'office'.
- Stap 4.2- Verkrijg een pointer naar het attribuut 'id'.
- Stap 4.3- Verkrijg een pointer naar de attributen (named node map).
- Stap 4.4- Vraag de waarde uit van dit attribuut en toon deze in SQL*Plus.
- Stap 4.5- Verkrijg een pointer naar de element child node 'city'.
- Stap 4.6- Verkrijg een pointer naar de text child node van element 'city', bepaal de waarde en toon deze in SQL*Plus.

Het resultaat van het voorgaande codevoorbeeld is:

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

Wijzigen XML-documenten

Bij het wijzigen van XML-documenten worden de DOM-operaties voor het onttrekken van informatie uit XML-documenten en het opbouwen van een XML-document min of meer gecombineerd. Daarnaast worden operaties zoals 'removeChild' voor het verwijderen van een child element en 'removeNamedItem' voor het verwijderen van een attribuut vaak gebruikt bij het wijzigen van een XML-document. Stel dat het, inmiddels bekende, 'officesList' XML-document gewijzigd moet worden naar het onderstaande format. Hierin wordt het element 'city' niet meer gebruikt.

```
<officesList>
<office id="1">Costa Mesa</office>
<office id="2">ElSegundo</office>
<office id="3">San Rafael</office>
<office id="4">Rocklin</office>
</officesList>
```

In het volgende codevoorbeeld wordt het XML-document gewijzigd door de text node onder 'city' te verplaatsen naar element 'office'.

```
declare

   l_document        dbms_xmldom.DOMDocument;
   l_root_element    dbms_xmldom.DOMElement;
   l_child_nodes     dbms_xmldom.DOMNodeList;
   l_office_node     dbms_xmldom.DOMNode;
   l_city_node       dbms_xmldom.DOMNode;
   l_node            dbms_xmldom.DOMNode;
   l_text_node       dbms_xmldom.DOMText;

   l_buffer          varchar2(1000);

begin

   -- 1: create new DOM document based on document in XDB repository

   l_document := dbms_xmldom.newDOMDocument(xdburitype('/public/temp/
california_offices.xml').getxml());

   -- 2: get the root element ('officesList')

   l_root_element := dbms_xmldom.getDocumentElement(l_document);

   -- 3: get child nodes of root element

   l_child_nodes := dbms_xmldom.getChildNodes(dbms_xmldom.makeNode(l_
root_element));

   -- 4: for each child ...

   for i in 0..(dbms_xmldom.getLength(l_child_nodes) - 1) loop

      -- 4.1: get the node (element 'office')

      l_office_node := dbms_xmldom.item(l_child_nodes, i);

      -- 4.2: get the first child node (element 'city')

      l_city_node := dbms_xmldom.getFirstChild(l_office_node);

      -- 4.3: get the text node (the first child node)

      l_text_node := dbms_xmldom.makeText(dbms_xmldom.getFirstChild(l_
city_node));

      -- 4.4: move text node

      l_node := dbms_xmldom.appendChild(l_office_node, dbms_xmldom.
makeNode(l_text_node));

      -- 4.5: delete 'city' element

      l_node := dbms_xmldom.removeChild(l_office_node, l_city_node);

   end loop;

   -- show the modified XML document


   dbms_xmldom.writeToBuffer(l_document, l_buffer);
   dbms_output.put_line(l_buffer);

end;
```

- Stap 1- Maak een nieuw document (DOM instantie) op basis van een XML-document in de Oracle XDB Repository.
- Stap 2- Verkrijg een pointer naar het root element.
- Stap 3- Haal child nodes op van het root element.
- Stap 4- Verplaats voor ieder child element de text node onder element 'city' naar element 'office'.
- Stap 4.1- Verkrijg een pointer naar het "zoveelste" child element 'office'.
- Stap 4.2- Verkrijg een pointer naar de element child node 'city'.
- Stap 4.3- Verkrijg een pointer naar de text child node van element 'city'.
- Stap 4.4- Verplaats de text node.
- Stap 4.5-Verwijder het child element 'city'.

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