Codegeneratie met XSLT

Object Types genereren met tabeldefinities

In Optimize 5 van 2009 wijdde ik een artikel aan Object Types. Daar merkte ik op dat het relatief simpel is om deze te genereren. Vaak gebruik je deze om data te verwerken in Pl/SQL code. En net als dat Toplink of ADF Business Components Java objecten mappen op tabellen, zijn ook Object Types vaak een reflectie van rijen in tabellen. Zou het daarom dan niet handig zijn om deze te genereren op basis van tabeldefinities?

En dan bij voorkeur op een flexibele manier, dus op een template-gebaseerde manier. Daardoor is het even zo eenvoudig om bijvoorbeeld Java-beans te genereren. Nu is het tamelijk makkelijk om de data-dictionary van Oracle uit te lezen voor tabel-, kolom- en primary key-definities. Met de XML-SQL functionaliteit vanaf Oracle 9i is het ook simpel om dit op te leveren in XML-formaat. En als je de definities in XML-formaat hebt, kun je eigenlijk alles genereren wat je wilt met ... XSLT.

Van tabel definities naar XML

De tabeldefinities zijn eenvoudig op te vragen uit de all_tables view van de data-dictionary (of de betreffende USERS_* of DBA_* varianten). Het is alleen zo dat we het resultaat in xml willen hebben. Voor dit soort toepassingen heeft Oracle XDB met het XMLtype en de XML-SQL functions gemaakt. Met deze functionaliteit ziet de query er als volgt uit:

```
select tbl.owner
     ,       tbl.table_name
     ,       xmlelement( "TableDefinition"
                       , xmlelement("Name", tbl.table_name)
                       ) xml
     from all_tables tbl
     where tbl.owner      = bTableOwner
     and   tbl.table_name = bTableName
     group by tbl.owner, tbl.table_name
  ;
```

Deze simpele query levert het volgende xml:

```
<TableDefinition><Name>HSD_EMPLOYEES</Name></TableDefinition>
```

De truuk is in het toevoegen van de 'xmlelement' function dat een xmltype levert, met de element-name als de eerste parameter met daaropvolgend een willekeurig aantal xmltype parameters. Of een referentie naar een kolom.
Vervolgens voegen we de kolomen van de tabel toe. Maar dan in een hiërarchisch dieper niveau. Dat is wat lastig om in een enkele query te doen. Maar ik houd er ook van om mijn queries op een nette manier te modulariseren.

```
select xmlelement( "Columns"
                         , xmlagg(xmlelement( "Column"

, xmlelement("ColName", col.column_name)

, xmlelement("ColId", col.column_id)

, xmlelement("DataType", col.data_type)

, xmlelement("DataLength", col.data_length)

, xmlelement("DataPrecision", col.data_precision)

, xmlelement("DataScale", col.data_scale)
                                                  )
                                    )
                         ) xml
    from all_tab_cols col
    where col.owner      = 'HDEMO65'
    and   col.table_name =  'HSD_EMPLOYEES';
```

Hier is een andere functie: de 'xmlagg'. Deze functie aggregeert meerdere rijen in een xmltype. Zo krijgen we een element 'Columns' met daarin per kolom een 'Column' element met corresponderende informatie. Daarbij moeten we primary key informatie toevoegen. Dit hebben we nodig om de parameters voor de constructor-methode in het object type te genereren.

```
select cnt.owner
       ,       cnt.table_name
       ,       cnt.constraint_name
       ,       xmlelement( "PrimaryKey"
                         , xmlelement("PrimaryKeyName", cnt.constraint_
name)
                         , xmlelement( "PrimaryKeyColumns"
                                     , xmlagg( xmlelement("Column"

, xmlelement("Name", cln.column_name)

, xmlelement("Position", cln.position)
```

CENTENNIUM BI OPLEIDINGEN

GENESEE ACADEMY World-Class Training

Data Vault Institute

Juni 2010 - Ouwehands Dierenpark, te Rhenen
On-site dagen op 10 & 11 juni 2010

Data Vault Modeling & Certification

2 dagen on-line training + 2 dagen on-site training en certificering + een gratis dag additionele on-line verdiepingsmodules!

Kijk voor meer informatie over deze en onze andere Business Intelligence cursussen op WWW.BI-OPLEIDINGEN.NL

Uw exclusieve kans om 'Certified Data Vault Data Modeler' te worden.

Altijd al willen weten hoe Data Vault kan bijdragen aan de realisatie van flexibele en kosteneffectieve datawarehouses met korte ontwikkeltijden? Hoe u een Data Vault ontwerpt die zich voortdurend aanpast aan veranderende businessbehoeften? Wat de kenmerken van een Data Vault zijn? En wat de concrete toegevoegde waarde is voor zowel business als IT?

Antwoord op deze en vele andere vragen rondom Data Vault krijgt u via dit vernieuwde, seminar. Het seminar bestaat uit twee on-site cursusdagen welke voorafgegaan worden door twee dagen aan on-line materiaal. De on-line onderdelen worden twee weken voor aanvang van de on-site cursusdagen aangeboden aan de deelnemers.

De on-line onderdelen stellen de deelnemers in staat om in hun eigen tempo de basis van Data Vault te leren, deze worden geheel en exclusief verzorgd door Dan Linstedt en/of Hans Hultgren. Inhoud, opzet en leservaring zijn vergelijkbaar met een on-site cursus. Tijdens de on-line delen wordt uw kennisopbouw getoetst middels een aantal korte on-line assessments, waardoor u verzekerd bent van optimale voorbereiding op de on-site dagen. De deelnemers hebben tevens tot twee weken na afloop van de on-site cursusdagen toegang tot de bijbehorende website en kunnen daar een aantal aanvullende en verdiepende modules volgen naar wens.

Deze vernieuwde opzet kent het grote voordeel dat u slechts twee dagen volledig vrij hoeft te maken voor het volgen van de on-site delen. De overige delen kunt u flexibel in uw eigen tijd en op uw eigen tempo volgen. Het seminar wordt afgesloten met het examen dat u in staat stelt om 'Certified Data Vault Data Modeler' te worden! Een unieke kans om uw onderscheidende waarde als professional te vergroten.

On-line Training
In uw eigen tempo v.a. twee weken voorafgaand aan het on-site gedeelte.

On-site Training & Certification
10 & 11 juni 2010

On-line verdiepingsmodules
Beschikbaar tot twee weken na afronding van het on-site gedeelte.

Uw investering voor Data Vaul Modeling & Certification: € 3.275,00 (vrij van BTW).

Meer informatie: www.bi-opleidingen.nl of neem contact op met Esther Sorel of Remco Broekmans: opleidingen@centennium.nl of telefonisch: 070-3120 370.

Benieuwd naar het on-line materiaal?
Ga naar www.datavaultacademy.com en registreer voor een gratis on-line proefles.

Inclusief gratis Centennium BI opleidingen vouchers t.w.v. maximaal € 1.120,=

advies
projecten
detachering

business intelligence opleidingen

```
, xmlelement("DataType", tcl.data_type)

, xmlelement("DataLength", tcl.data_length)

  , xmlelement("DataPrecision", tcl.data_precision)

, xmlelement("DataScale", tcl.data_scale)
                                                              )
                                                )
                                          )
                          ) xml
       from all_constraints cnt
       ,    all_cons_columns cln
       ,    all_tab_columns  tcl
       where cnt.constraint_type = 'P'
       and   cln.owner            = cnt.owner
       and   cln.table_name       = cnt.table_name
       and   cln.constraint_name = cnt.constraint_name
       and   tcl.table_name       = cln.table_name
       and   tcl.owner            = cln.owner
       and   tcl.column_name      = cln.column_name
       and   cnt.owner            = b_Owner
       and   cnt.table_name       =  b_TableName
       group by cnt.owner, cnt.table_name, cnt.constraint_name;
```

Dit lijkt me ook niet al te gecompliceerd. Wat je hiervoor moet weten, is dat we de constraints van type 'P' (primary key) met de gerelateerde kolommen nodig hebben. Voor deze kolommen hebben we de datatype-informatie weer nodig. Strikt genomen zou je kunnen zeggen dat dit al is opgenomen in het eerdere 'columns'-gedeelte. Maar het is voor de xslt's makkelijker om de informatie hier als nog op te nemen.
De queries heb ik in een paar functies in een package opgenomen. Dan ziet de overall-query er als volgt uit:

```
select tbl.owner
     ,      tbl.table_name
     ,      xmlelement( "TableDefinition"
                      , xmlelement("Name", tbl.table_name)

, xxx_table_definitions.TabColumns(tbl.owner, tbl.table_name)

, xxx_table_definitions.PKColumns(tbl.owner, tbl.table_name)
                      ) xml
     from all_tables tbl
     where tbl.owner      = bTableOwner
     and   tbl.table_name = bTableName
     group by tbl.owner, tbl.table_name
```

En dat levert een xml op zoals hieronder. Ik heb het voor het gemak met Jdeveloper geherformateerd:

```
<TableDefinition>
  <Name>HSD_EMPLOYEES</Name>
  <Columns>
    …
    <Column>
      <ColName>SALARY</ColName>
      <ColId>8</ColId>
      <DataType>NUMBER</DataType>
      <DataLength>22</DataLength>
      <DataPrecision>7</DataPrecision>
      <DataScale>2</DataScale>
    </Column>
    <Column>
      <ColName>NAME</ColName>
      <ColId>5</ColId>
      <DataType>VARCHAR2</DataType>
      <DataLength>10</DataLength>
      <DataPrecision></DataPrecision>
      <DataScale></DataScale>
    </Column>
    …
    <Column>
      <ColName>ID</ColName>
      <ColId>1</ColId>
      <DataType>NUMBER</DataType>
      <DataLength>22</DataLength>
      <DataPrecision>4</DataPrecision>
      <DataScale>0</DataScale>
    </Column>
  </Columns>
  <PrimaryKey>
    <PrimaryKeyName>HSD_EMP_PK</PrimaryKeyName>
    <PrimaryKeyColumns>
      <Column>
        <Name>ID</Name>
        <Position>1</Position>
        <DataType>NUMBER</DataType>
        <DataLength>22</DataLength>
        <DataPrecision>4</DataPrecision>
        <DataScale>0</DataScale>
      </Column>
    </PrimaryKeyColumns>
  </PrimaryKey>
</TableDefinition>
```

De package body die hieruit op te maken is, is te bekijken op de website van Optimize (http://www.optimize.nl/Het-Blad/Optimize/Extra)

XSLT voor generatie

Nu alle definities in XML zijn op te vragen, kan de codegeneratie beginnen. Alles wat je nodig hebt is een code-editor en een xml-parser die de tussentijdse transformaties kan doen om de stylesheets te testen. Zelf heb ik er een simpele Java-tool voor gemaakt, maar Xtrans vind ik ook wel een aardige. Later in het artikel leg ik uit hoe je hetzelfde in de database kunt doen.

Basistemplate

Ik ga hier geen complete XSLT cursus geven, maar alleen de basisprincipes. Hier is een basistemplate:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
  <!-- Root template -->
  <xsl:template match="/">


  </xsl:template>

</xsl:stylesheet>
```

Als je vanuit de 'New Gallery' een nieuw 'XSL Style Sheet' in jDeveloper laat aanmaken, krijg je dit. Natuurlijk doet het nog niet veel. Maar wat je ziet, is een heel simpel xml-document. En het start met een zogenaamd template met in een match attribute een xpath-expressie dat het root-element bevraagd van het

ABONNEES GRATIS TOEGANG TOT BI-EVENT

18 MEI 2010 - INTRES - HOEVELAKEN

BI-EVENT 2010

CONGRES MET INTERNATIONALE KEYNOTE SPREKERS BILL INMON, MARK GREAVES EN RICK VAN DER LANS

Uw dagvoorzitter is Hans Lamboo


Bill Inmon


Mark Greaves


Rick van der Lans


Hans Lamboo

19 MEI 2010 - INTRES - HOEVELAKEN

Congresspreker **Bill Inmon** presenteert de dag na het BI-event zijn eendaagse interactieve workshop

"THE TEXTUAL DATA WAREHOUSE"

Informatie en registratie: www.bi-event.nl en www.arrayseminars.nl

Array SEMINARS

DB/M

HOOFDSPONSOR: ORACLE

xml-document dat getransformeerd moet worden. Belangrijk voor ons is het toevoegen van een output style:

```
<xsl:output method="text" indent="no"/>
```

Dit geeft aan dat we als uitkomst van de transformation 'platte ascii tekst' verwachten, aangezien we 'free-formatted source' genereren (dus geen xml of html).

Het doorlopen van de XML

Er zijn in de basis twee manieren om in XSL door de bron-xml te 'wandelen' en op die manier je XSL op te bouwen om de output te genereren. De meest recht-toe-recht-aan methode is het gebruik van een 'xsl:for-each':

```
<xsl:for-each select="/ns1:SOAOrderBookingProcessRequest/
po:PurchaseOrder/po:OrderItems/po:Item">
```

Binnen dit xml-element kun je de elementen van de bron-xml bevragen die vallen binnen de xpath-selectie die in het 'select' attribuut is opgegeven. Daarvoor wordt het <xsl:value-of select="po:price"/> element gebruikt.

Het voordeel hiervan is dat er een 'order by' sortering opgegeven kan worden door toevoeging van een <xsl:sort > element als het eerste “child” element.

Dit is de manier zoals het wordt gegenereerd als het wordt ontwikkeld met de Transformation-tool in de BPEL en ESB designer van jDeveloper.

Maar voor het opbouwen van mijn codegenererende stylesheets geef ik de voorkeur aan een andere methode: het gebruik van sub-templates.

In de basisstylesheet zag je al een eerste template. Maar je kunt er zoveel toevoegen als je zelf wilt. Ik ben gewend minstens een template per 'functioneel niveau' in de bron xml toe te voegen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
 <xsl:output method="text" indent="no"/>
 <!-- Main template -->
 <xsl:template match="/">
  <xsl:apply-templates select="/TableDefinition" mode="ObjectType"/>
 </xsl:template>
 <!-- Template to create the object type -->
 <xsl:template name="ObjectType" match="/TableDefinition"
mode="ObjectType">
  <xsl:call-template name="CreateType"/>
  <xsl:call-template name="CloseType"/>
 </xsl:template>
 <xsl:template name="CreateType">
 <!-- Declaration of the Type -->
  <xsl:value-of select="concat('create or replace type ', Name)"/>
 </xsl:template>
 <!-- Close the Type -->
 <xsl:template name="CloseType">
  <xsl:text>
);</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

Wat je ziet is dat in het 'main template' het commando 'apply-templates' is toegevoegd met een Xpath expressie. Dit stelt dat elke template die voldoet aan de xpath expressie wordt uitgevoerd. In dit geval zal de xsl-processor een template genaamd 'ObjectType' vinden met een match-attribute dat overeenkomt met het select attribute van het 'apply-templates' commando.

Er is ook een derde en een vierde template die zorgen voor de declaratie en afsluiting van het Object Type. Deze templates worden aangeroepen met een 'xsl:call-template' statement en kan alleen tekst bevatten of elke complexe template structuur dat xpath-expressions bevat die uitgaan van de node welke geselecteerd is in het aanroepende template. In de CreateType template staat bijvoorbeeld een 'value-of' statement met een 'select'-attribute dat een concatenatie van 'create or replace type' en de naam van de tabel uitwerkt. 'Name' is een sub-node van '/TableDefinition' die is geselecteerd in het aanroepende template.

Bij het CreateType-template en de 'apply-templates' is ook een mode-attribute opgegeven. In dit geval heeft het weinig toegevoegde waarde. Maar met dit attribute is het mogelijk om de volgorde van uitvoer van de apply-templates te controleren. Er zijn een aantal regels die de xsl-processor volgt om te bepalen welk sub-template als eerste uit te voeren. Je kunt deze volgorde bereken, maar er is feitelijk weinig aan te beïnvloeden. Daarentegen kan het gecontroleerd worden door het opgeven van een modus waarin de processor de template moet toepassen. In dat geval zal het alleen die templates toepassen die behalve aan de xpath-expression ook voldoen aan de mode. Ik zal hier later een voorbeeld geven.

Variabelen

Variabelen blijken ook aardig handig te zijn. Eerder plaatste ik mijn tekstonderdelen in sub-templates die ik liet aanroepen. Dat werkt prima, maar variabelen zijn soms handiger om default-teksten en karakters te bevatten, zoals new-lines, quotes etc. Deze variabelen kunnen vervolgesn gebruikt worden in xpath expressies. Dus, om er maar eens een paar toe te voegen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
 <xsl:output method="text" indent="no"/>
<!-- Variables -->
<!-- Object type suffix -->
          <xsl:variable name="ObjectSuffix"><xsl:text>_type </
xsl:text></xsl:variable>
 <!-- Close the Type -->
 <xsl:variable name="CloseType">
  <xsl:text>
);</xsl:text>
 </xsl:variable>
 <!-- Object Heading -->
 <xsl:variable name="ObjectTypeHeading">
                   <xsl:text> as object
(
  -- Author  : Martien van den Akker, Darwin-IT Professionals
  -- Created : 8/15/2008 12:00:00 PM
  -- Purpose : Table to Object Mapping

  -- Attributes
</xsl:text>
    </xsl:variable>
…
```

```
 Zie onder <!-- Main template --> uit vorig voorbeeld.
…
</xsl:stylesheet>
```

Hier zie je dat ik de CloseType template heb getransformeerd in een variable en een ObjectSuffix en een ObjectTypeHeading variable heb toegevoegd. Hieraan wordt gerefereerd vanuit een Xpath-expressie door de naam toe te voegen met een dollar-teken. De ObjectSuffix voegde ik toe om zeker te stellen dat de object-name uniek is ten opzichte van de tabelnaam.

De output van de stylesheet is tot zo ver:

```
create or replace type HSD_EMPLOYEES_type  as object
(
  -- Author  : Martien van den Akker, Darwin-IT Professionals
  -- Created : 8/15/2008 12:00:00 PM
  -- Purpose : Table to Object Mapping

  -- Attributes
);
```

Misschien niet al te spectaculair, maar het geeft een indruk. Laten we er wat meer spectakel aan toe voegen.

Het gebruik van Modes: toevoegen van attributen

We willen dat het object type attributen heeft die de kolommen van de brontabel representeren.

Om dat te doen voegen we drie templates toe. Een die feitelijk de generatie van de attributen verzorgt. Maar we zullen de attributen moeten scheiden met een komma. We kunnen elke attribuutdeclaratie met een komma eindigen. Alleen de laatste moet er geen een krijgen. Mijn eigen programmeer-standaarden geven de voorkeur elke regel te prefixen met een komma. En dat maakt het ook iets makkelijker om te genereren. Ik gebruik een template die het eerste attribuut genereert en vervolgens een tweede die de rest verzorgt. Ik stuur de xsl-processing door de "mode" attributen en de xpath expression met de xpath-function 'position()'.

```
 <!-- Generate the Attributes -->
 <xsl:template name="Attributes">
  <xsl:apply-templates select="Columns/Column[1]"
mode="AttributeFirst"/>
  <xsl:apply-templates select="Columns/Column[position()>1]"
mode="AttributeRest"/>
 </xsl:template>
 <!-- Generate First of the Attributes -->
 <xsl:template name="AttributeFirst" match="Columns/Column"
mode="AttributeFirst">
  <xsl:value-of select="concat('  ',ColName, ' ' , DataType)"/>
  <xsl:call-template name="DataTypeDefinition"/>
 </xsl:template>
 <!-- Generate Rest of the Attributes -->
 <xsl:template name="AttributeRest" match="Columns/Column"
mode="AttributeRest">
  <xsl:value-of select="concat($NewLineComma, ColName, ' ' ,
DataType)"/>
  <xsl:call-template name="DataTypeDefinition"/>
 </xsl:template>
 <!-- DataType Definition -->
 <xsl:template name="DataTypeDefinition">
  <xsl:if test="DataType!='DATE'">
   <xsl:choose>
    <xsl:when test="string-length(DataPrecision)>0">
     <xsl:value-of select="concat('(', DataLength, ', ' ,DataPrecision,
')')"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="concat('(', DataLength, ')')"/>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:if>
 </xsl:template>
```

Het sturen van de xsl-processor kon ook makkelijk worden gedaan met een xsl:if constructie zoals in het 'DataTypeDefinition' is te zien. Daar wordt getest of er een grootte en een precisiedeel aan het datatype moet worden toegevoegd (voor varchars en numbers). Wanneer de attributen ook gesorteerd moeten worden, wordt een 'xsl:for-each' met een 'xsl:sort' gebruikt. Vervolgens kan met een slimme xsl:if worden getest of het de eerste loop-cyclus is of een latere. Naast een "xsl:if" vind je hier ook een voorbeeld van een 'xsl:choose'.

Het toevoegen van een constructor en een method

Een Object moet geinstantieerd kunnen worden. Om het object te instantieren volstaat het om gewoon een procedure te maken die een select doet en de resultaten doorgeeft aan de default constructor van het Object Type. Het is echter prettiger om een constructor te hebben gebaseerd op de primary key van de tabel en deze de betreffende rij te bevragen in de attributen. In de constructor kan de code veel eenvoudiger zijn en vanuit Object Orientatie Principes hoort het daar ook eigenlijk thuis. Je kunt het dan immers gebruiken waar vandaan het nodig is. Het creeëren van de declaratie van de constructor gaat op dezelfde manier als eerder met de attributen.

Een extra method ToXML is toegevoegd om een eenvoudige methode te hebben om de inhoud van het Object Type in een XMLType op te vragen. Dat is in elk geval handig voor testdoeleinden, maar kan functioneel ook praktisch zijn.

De complete xslt is te vinden op de website van Optimize (http://www.optimize.nl/Het-Blad/Optimize/Extra).

Het resultaat van de transformatie is als volgt:

```
create or replace type HSD_EMPLOYEES_type  as object
(
  -- Author  : Martien van den Akker, Darwin-IT Professionals
  -- Created : 8/15/2008 12:00:00 PM
  -- Purpose : Table to Object Mapping

  -- Attributes
  CREATED_BY VARCHAR2(30)
, CREATION_DATE DATE
, LAST_UPDATE_DATE DATE
, LAST_UPDATED_BY VARCHAR2(30)
, CIVIL_STATE VARCHAR2(1)
, BANK_ACCOUNT NUMBER(22, 15)
, E_MAIL VARCHAR2(30)
, TELEPHONE VARCHAR2(20)
, COUNTRY VARCHAR2(24)
, ZIP_CODE VARCHAR2(10)
, CITY VARCHAR2(24)
, STREET VARCHAR2(24)
, COMMISSION NUMBER(22, 7)
```

```
, SALARY NUMBER(22, 7)
, HIRE_DATE DATE
, JOB VARCHAR2(9)
, NAME VARCHAR2(10)
, EMP_ID NUMBER(22, 4)
, DEP_ID NUMBER(22, 2)
, ORACLE_USERNAME VARCHAR2(30)
, ID NUMBER(22, 4)
, constructor function HSD_EMPLOYEES_type ( p_ID NUMBER) return self as
result
, member function to_xml return sys.xmltype
);
```

De object type body

Zo kun je de object type body zelf (redelijk) eenvoudig maken. Deze is te vinden op de website van Optimize (http://www.optimize.nl/Het-Blad/Optimize/Extra).

Het uitvoeren van de XSL in de database

Eigenlijk zijn er twee methodes voor het uitvoeren van de de transformatie in de database. Wanneer de XSLT's in een CLOB-kolom in een tabel in de database zitten, kan het als volgt:

```
select xtb.xml.transform( xmltype(doc.xmldoc) ) from (
 select tbl.owner
     ,      tbl.table_name
     ,      xmlelement( "TableDefinition"
                      , xmlelement("Name", tbl.table_name)

, xxx_table_definitions.TabColumns(tbl.owner, tbl.table_name)

, xxx_table_definitions.PKColumns(tbl.owner, tbl.table_name)
                      ) xml
     from all_tables tbl
     where tbl.owner      = 'HDEMO65'
     and   tbl.table_name = 'HSD_EMPLOYEES'
     group by tbl.owner, tbl.table_name) xtb
     ,  xxx_xml_documents doc
where doc.docname='CreateObjectTypeBody';
```

Het nadeel hiervan is de mindere performance als gevolg van de verscheidene parses die meerdere keren gedaan worden. Dus geef ik er de voorkeur aan om het in Pl/SQL uit te werken met de dbms_xslprocessor package.

Eerst moet de xslt-stylesheet geparst worden. Onthoudt: een xsl-stylesheet is in zichzelf een xml-document. Dit kan in Pl/Sql op de volgende manier worden gedaan:

```
    retDoc xmldom.DOMDocument;
    parser xmlparser.Parser;
  BEGIN
    parser := xmlparser.newParser;
    xmlparser.parseCLOB(parser, xml);
    retDoc := xmlparser.getDocument(parser);
    xmlparser.freeparser(parser);
    return retDoc;
  END;
```

Vervolgens moet met het resulterende DOMDocument een XMLStylesheet worden gemaakt:

```
    xmldoc    xmldom.DOMDocument;
    newsheet xslprocessor.Stylesheet;
  BEGIN
    xmldoc   := xml.parse(doc); -- See the Pl/Sql code above
    newsheet := xslprocessor.newStylesheet(xmldoc, NULL);
    xmldom.freeDocument(xmldoc);
    RETURN newsheet;
```

En aan het eind de transformatie zelf, na het parsen van het bron-xml-document:

```
retval VARCHAR2(32767);
  BEGIN
    xmldoc := xml.parse(source); -- See the parsing code above
    retval := transform(xmldoc, style, params);
    xml.freeDocument(xmldoc);
    RETURN retval;
```

Wanneer de resulterende code een valide DDL statement is, zoals ons create type, dan kan het met een eenvoudige 'execute immediate retval;' meteen in de database worden aangemaakt.

Conclusies en opmerkingen

Zoals je ziet is het niet al te lastig om code te genereren met XSL en een xml-query op de database. De toepassingen zijn schier oneindig. Met wat klussen is het ook niet te lastig om bijvoorbeeld Pl/SQL-packages uiteen te rafelen (met bijvoorbeeld record-type declaraties,etc.) in XML om daar dan weer code mee te genereren. Wanneer de definities in een xml-file staan, kan het in alles worden getransformeerd. Zo kun je er bijvoorbeeld net zo gemakkelijk java beans uit genereren.

Er is overigens een belangrijk verschil tussen de de parser in de database (Oracle 10gR2) en de parser meegeleverd met jDeveloper. De java-xml-parser van jDeveloper transformeert de xml precies zoals ik verwacht en als getoond in de voorbeelden in dit artikel. De database-xml-parser, daarentegen, transformeert quotes, 'groter dan' karakters in '=>' code, etc. Naar zogenaamde 'excaped XML-characters' zoals '"', '>', etc. Ik heb nog niets gevonden om dat aan te sturen. Zelfs een XSL-attribute als 'disable-output-escaping="yes"' helpt niet. Dus voor de resulterende code van de database-parser moeten deze nog worden vervangen met het corresponderende karakter zoals:

```
function replaceEntities(pDoc in varchar2)
return varchar2
is
  lDoc varchar2(32767);
begin
  lDoc := replace(srcstr => pDoc, oldsub => '&amp;quot;', newsub => '"');
  lDoc := replace(srcstr => lDoc, oldsub => '&amp;apos;', newsub => '''');
  lDoc := replace(srcstr => lDoc, oldsub => '&amp;gt;'  , newsub => '>');
  return lDoc;
end;
```

Voor the xml-parsing en de xslt-transformation in de database moet ik nog vermelden dat ik de voorbeeldpackages van Steve Muench's boek 'Building Oracle XML Applications' heb gebruikt.

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

Sander Hoogendoorn presenteert:

SCHATTEN MET USE CASES

Middagseminar over het plannen van software development projecten

26 mei 2010 – Hotel Lapershoek Hilversum

In dit beknopte maar intensieve seminar toont Sander Hoogendoorn aan waarom schatten zo moeilijk is. Hij geeft een beknopt overzicht van veel toegepaste schattingstechnieken. Aansluitend laat de spreker zien hoe één van deze technieken, *smart use cases,* kunnen worden ingezet bij het maken van schattingen, zelfs al in een vroege fase van het project. De spreker laat zien hoe smart use cases zijn te identificeren, te modelleren en hoe ze gemakkelijk zijn in te schatten. Natuurlijk zijn ook smart use cases geen silver bullet, maar zij hebben zich desondanks als techniek succesvol bewezen in diverse typen projecten, waaronder .Net, Java, Sharepoint, servicegeoriënteerde ontwikkeling en zelfs business intelligence projecten en SAP implementaties. Tenslotte laat Sander zien hoe ook het plannen van projecten vergemakkelijkt aan de hand van smart use cases, met name in iteratieve, agile projecten.

Bestemd voor ú
Heeft u tijdens software development projecten te maken met het inschatten van tijd, complexiteit en omvang? Bent u betrokken bij software development? Dan mág u dit middagseminar niet missen!

PRAGMATISCH ONTWIKKELEN MET .NET

Best practices in .NET projecten

2 juni 2010 – Hotel Lapershoek Hilversum

De onderwerpen van dit seminar dragen direct bij aan het succesvol ontwikkelen van software. Tijdens het seminar bespreekt Sander Hoogendoorn, principal technology officer bij Capgemini en lid van de Visual Studio Advisory Board bij Microsoft, het toepassen van software architectuur, het opzetten van frameworks en het hanteren van de juist ontwerppatronen. Het seminar geeft veel praktijkvoorbeelden over het toepassen van dergelijke patronen in de dagelijkse praktijk, maar toont ook diverse zeer leerzame anti-voorbeelden. Het geeft de deelnemers een helder inzicht in de positieve bijdrage die deze technieken leveren aan projecten, het motiveert de deelnemers en biedt talrijke handvatten voor het verbeteren van de kwaliteit en onderhoudbaarheid van uw software ontwikkeling.

Bestemd voor ú
Bent u betrokken bij software development in .NET? Dan mag u dit seminar niet missen!

MELD U SNEL AAN OP WWW.ARRAYSEMINARS.NL!

Array SEMINARS