ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 5 - NUMMER 4, DECEMBER 2008

OPEN CURSOR

"Nieuw in DB2 versie 9", in het bijzonder DB2 for z/OS, blijft een boeiend terrein voor zowel experimenteren als lectuur. En vooral ook voor pogingen om tussen de bomen van al die (technische) lectuur de essentie, de concepten terug te vinden. Dat is wat we u in het verleden en ook in dit nummer van "Exploring DB2" proberen te bieden. In het bijzonder focussen we opnieuw op "DB2 en XML"; dit nummer wordt om die reden uitzonderlijk eveneens verzonden naar mensen met eerder een XML- dan een DB2-achtergrond. Welkom bij "Exploring DB2 (& XML)"! Bij deze "bos en bomen"-poging vergeten we overigens niet dat velen onder ons nog met versie 8 of zelfs 7 van DB2 (moeten) werken. Om die reden heeft ABIS trouwens een nieuw initiatief in het leven geroepen: ***Exploring DB2 - live!*** *– een reeks van zes avondsessies over DB2. U vindt meer informatie op de bijgevoegde flyer, of op onze website.*

Het ABIS DB2-team.

IN DIT NUMMER:

- Een vervolg op het artikel in het vorige nummer over nieuwe mogelijkheden in DB2 versie 9. Nu een onderwerp dat ook voor versies 7 en 8 van DB2 van toepassing is: *Auto-rebind*.
- Nieuw in DB2 versie 9, maar wellicht nog niet zo goed begrepen, dus wat duiding over C*lone tables*.
- En een nieuw artikel in de reeks over "DB2 en XML", met name over XSLT, de "transform"-component van de XML-familie, in *XSLT genereert SQL*.
- En zoals gewoonlijk, de *cursusplanning* (op p.16), deze maal inclusief ons aanbod van XML-cursussen.

4

CLOSE CURSOR

In een volgend nummer leest u meer over XQuery, vertellen we u welke nieuwe EXPLAIN-mogelijkheden versie 9 (maar ook versie 8) van DB2 for z/OS biedt, en vindt u de winnaars van onze prijsvraag (zie p. 15).

Hopelijk tot ziens op één van onze "Exploring DB2 - live!" avondsessies, of tot in het volgende nummer!

Auto rebind *Steven Scheldeman (ABIS)*

In de vorige editie van Exploring DB2 stond er een artikel gewijd aan de mogelijkheden (van DB2 9 for z/OS) om bij een REBIND het oude ACCESS PATH te bewaren, om het in geval van eventuele problemen te kunnen herstellen. Dit brengt ons bij het onderwerp van dit artikel: waar het in die bijdrage ging over een manuele, expliciete REBIND, willen we het hier hebben over een REBIND die in feite zonder expliciete toestemming wordt uitgevoerd door DB2.

AUTO REBIND bestaat sinds versie 5 van DB2 for z/OS, en is in de eerste plaats bedoeld om bepaalde ingrijpende aanpassingen aan de database (b.v. een DROP/CREATE TABLE) toch transparant te maken voor reeds geBINDe applicaties.

Om te beginnen kan auto-rebind op systeemniveau gedeactiveerd worden: of er een AUTO REBIND kan plaatsvinden, hangt immers ook af van de gekozen optie voor de ABIND (Auto BIND) parameter op het DB2-installatiepaneel DSNTIPO. De default waarde hiervoor is YES; deze zparm kan trouwens online aangepast worden (paneel DSNTIPB). Andere waarden voor ABIND zijn NO en COEXIST (zie verderop).

Veronderstel dat ABIND actief is. In welke situaties vindt er dan effectief een AUTO REBIND plaats, en voor welke objecten? Wel, dit gebeurt op het ogenblik dat een PLAN of PACKAGE op het punt staat uitgevoerd te worden, maar door DB2 als 'invalid' is gemarkeerd. Kort samengevat is de enige reden voor DB2 om dit te doen, het feit dat het huidige ACCESS PATH niet meer kan werken, omdat het gebruik maakt(e) van ondertussen ongeldig geworden objecten of autorisaties.

Hier volgt een meer gedetailleerde opsomming van redenen voor een “invalidatie” van een PLAN of PACKAGE, en dit zowel in versie 8 als in versie 9:

- een (ander) PACKAGE is verwijderd, en het bewuste object (PLAN of PACKAGE) is hiervan afhankelijk
- het object is afhankelijk van een verwijderde TABLE, INDEX of VIEW, of van een tabel die van naam is veranderd of waarvan een gebruikte kolom van datatype, lengte of van naam is veranderd
- de OWNER van het object heeft niet langer de autorisatie om een TABLE, INDEX, VIEW, PACKAGE of PLAN te gebruiken dat voor dit object effectief nodig is
- een USER DEFINED FUNCTION (UDF) waarvan het object afhankelijk is, wordt gewijzigd
- de autorisatie om een STORED PROCEDURE uit te voeren is de OWNER van het object ontnomen, en het object roept die STORED PROCEDURE op met 'CALL <procedure-naam>'
- een TABLE waarvan het object afhankelijk is, wordt aangepast door er een TIME-, TIMESTAMP- of DATE-kolom aan toe te voegen

- een onderliggende TABLE krijgt een zichzelf refererende CONSTRAINT toegevoegd, of een CONSTRAINT met een 'delete'-regel van SET NULL of CASCADE
- de limietwaarden van een PARTITIONED INDEX waarvan het object afhankelijk is, worden gewijzigd
- de definitie van een INDEX waarvan het object afhankelijk is, wordt gewijzigd van NOT PADDED naar PADDED of omgekeerd
- omwille van een gewijzigde kolom in een onderliggende TABLE, kan een view waarvan het object afhankelijk is, niet langer door DB2 gegenereerd worden
- een “CREATED TEMPORARY TABLE” waar het object van afhankelijk is, wordt gewijzigd door er een kolom aan toe te voegen

Zelfs al is de oorzaak voor het als ‘invalid’ te markeren van een plan of package ondertussen verdwenen, toch blijft het object gemarkeerd als ongeldig. Dit kunnen we in de catalog-tabellen SYSIBM.SYSPLAN en SYSIBM.SYSPACKAGE terugvinden: in de kolom VALID staat genoteerd (met “Y” of “N”) of een bepaald PLAN of PACKAGE nog geldig is.

Een AUTO REBIND zal trouwens falen indien de oorzaak voor het invalideren nog bestaat (behalve als die oorzaak een INDEX was): het hele AUTO REBIND-concept gaat er dus van uit dat een dergelijke oorzaak van (zeer) tijdelijke duur is. Voorbeelden hiervan zijn: een TABLE wordt geDROPt en dadelijk daarna opnieuw gecreëerd; een kolom wordt geRENAMEd en dadelijk daarna wordt een nieuwe kolom met dezelfde (oude) naam toegevoegd; een onderliggend PACKAGE werd geFREEd maar is ondertussen opnieuw aangemaakt.

AUTO REBIND gebeurt dus *niet* wanneer b.v. een nieuwe index gecreëerd wordt, die door een manuele REBIND zou gebruikt worden. En evenmin wanneer aan een gebruikte tabel een numerieke of tekstkolom wordt toegevoegd, of wanneer de autorisaties voor het object zelf worden gewijzigd.

In zeer specifieke gevallen zal DB2 toch een AUTO REBIND doorvoeren van een PLAN of PACKAGE dat niet als 'invalid' geboekt staat. Indien de ABIND parameter (DSNTIPO-scherm) op 'YES' staat, gebeurt dit in de volgende gevallen:

- het object werd geBIND in een recentere versie van DB2 dan de versie waarop het effectief gedraaid wordt. Dit kan voorvallen in een DATA SHARING-omgeving of indien een DB2 subsysteem is moeten terugvallen op een eerdere versie van DB2
- het object werd geBIND vóór versie 4 van DB2. Elk dergelijk PLAN of PACKAGE dat gedraaid wordt op versie 9 van DB2, zal het lijdend voorwerp van een AUTO REBIND worden.
- het object is locatie-afhankelijk en wordt gedraaid op een andere locatie dan die waarop het geBIND werd. Deze situatie kan zich voordoen wanneer delen van een DATA SHARING GROUP gedefinieerd zijn als locatie afhankelijk, en een PACKAGE draait op een ander deel dan het deel waarop het oorspronkelijk geBIND werd.

Indien de ABIND-parameter (DSNTIPO-scherm) op COEXIST staat, gebeurt een AUTO REBIND van een niet-invalidated plan of package in het volgende geval:

- het subsysteem waarop een PLAN of PACKAGE draait, maakt deel uit van een DATA SHARING GROUP, en het werd geBIND op de huidige versie van DB2, maar wordt gedraaid op een voorgaande versie.

Dit is het enige verschil tussen ABIND=YES en ABIND=COEXIST.

Ik wil hier onmiddellijk aan toevoegen, dat indien de ABIND-parameter op 'NO' staat, DB2 een foutmelding genereert telkens een PLAN of PACKAGE een REBIND nodig heeft, maar - omwille van de ABIND-waarde - dit niet automatisch kan geschieden.

Enkele bijkomende opmerkingen:

- De gebruikte opties voor de parameters bij een AUTO REBIND zijn dezelfde als die welke gebruikt werden bij de meest recente (RE)BIND.
- Indien een PACKAGE (in versie 9) kopieën heeft ten gevolge van een (RE)BIND met de PLANMGMT(BASIC) of PLANMGMT(EXTENDED) optie, of omdat de subsysteem-parameter PLANMGMT op BASIC of EXTENDED staat, dan zullen die kopieën niet aangepast worden door een AUTO REBIND. Men blijft dus de mogelijkheid hebben om met een SWITCH een eerder PACKAGE terug te activeren. Alleen de huidige versie wordt door AUTO REBIND vervangen.
- Mocht een AUTO REBIND niet lukken - om om het even welke reden - dan zal dat PLAN of PACKAGE als onuitvoerbaar ('inoperative') worden bestempeld door DB2. We kunnen in de catalog-tabellen SYSIBM.SYSPLAN en SYSIBM.SYSPACKAGE de kolom OPERATIVE nakijken, om te zien welke PLANs of PACKAGEs als onuitvoerbaar zijn gemarkeerd.
- Ook in het geval van een AUTO REBIND zal het genereerde ACCESS PATH gedocumenteerd worden, indien de optie EXPLAIN(YES) voorkwam op de voorgaande (RE)BIND, en indien bovendien het ABEXP-veld op het installatiescherm DSNTIPO (of zparm ABEXP in DSN6SPRM) op YES staat. Doet er zich echter een EXPLAIN fout voor (een andere dan “PLAN_TABLE not found”) dan zal de AUTO REBIND ook falen.

Tot slot wil ik ook even melden dat de SQL Communication Area (SQLCA) niet beschikbaar is tijdens een AUTO REBIND. Dit wil zeggen dat niet alle boodschappen doorgegeven worden. Wil je zorgen dat applicaties toch alle mogelijke meldingen ontvangen, vermijd dan de AUTO REBIND door het uitvoeren van een manuele REBIND PLAN of REBIND PACKAGE na elk van de bovengenoemde wijzigingen.

Clone tables

Peter Vanroose (ABIS)

Eén van de structurele nieuwigheden in versie 9 van DB2 for z/OS is de mogelijkheid om tabellen te “klonen”. De term “clone tables” is misschien wat ongelukkig gekozen, en zaait mogelijk verwarring of wekt verkeerde verwachtingen. Daarom een kort woordje uitleg over wat je wel en niet mag verwachten van deze nieuwigheid.

Het klonen is in het leven geroepen voor applicaties die de inhoud van een tabel volledig moeten vervangen (“refresh”), terwijl de tabel ondertussen voortdurend beschikbaar blijft. Een typische context waarin dit nuttig kan zijn, is b.v. Data Warehousing, alhoewel we daar eerder aan Materialised Query Tables en het refresh-statement denken. Veel applicaties moeten “massieve” data-ververs-operaties uitvoeren; dit kan zowel gaan over zeer veel INSERTs, of DELETEs, of zelfs UPDATEs. Net die langdurigheid van de data-refresh vormen het probleem, omwille van locking en dus data-onbeschikbaarheid.

Een alternatief kan het LOAD-utility zijn, maar ook daar stellen zich mogelijk locking-problemen. Wat we met klonen dus krijgen is een soort online LOAD REPLACE, alleen beter. De constante beschikbaarheid van de data lijkt een contradictie: tijdens het overschrijven van de oude data is noch die oude data, noch de nieuwe data in z'n geheel toegankelijk. Dus: ofwel is de tabel langere tijd onbeschikbaar, ofwel is de zichtbare data een mix van oud en nieuw, en dus inconsistent.

Toch is het concept van een “vervangen van oud door nieuw met een vingerknip” implementeerbaar: denk aan “versioning”. Dit gaat als volgt: neem rustig de tijd om de nieuwe data klaar te zetten, *naast* de oude data. Ondertussen blijft de oude data gewoon toegankelijk. “Switch” dan oude en nieuwe data (of anders gezegd: laat de gebruikers “plots” de nieuwe versie van de data zien). Dit idee werd trouwens al in DB2 v8 for z/OS gebruikt voor de FASTSWITCH-optie van REORG. Dus eigenlijk niets nieuws onder de zon. Alleen de specifieke set-up en de SQL-syntax zijn nieuw. Want een ander belangrijk aspect van de implementatie m.b.v. een kloontabel is het feit dat dit via SQL gebeurt (dus b.v. vanuit een applicatie), daar waar REORG en LOAD REPLACE utilities zijn.

Een bestaande tabel wordt gekloond m.b.v. het SQL-statement

```
ALTER TABLE basistabelnaam ADD CLONE kloontabelnaam
```

Een kloontabel wordt dus niet aangemaakt via CREATE TABLE, maar afgezien daarvan gedraagt hij zich als een gewone tabel met een vrij te kiezen naam, uiteraard met exact dezelfde structuur (kolomnamen, datatypes, NOT NULL-attributen, check constraints en indexen) als de basistabel. Zelfs BEFORE-triggers worden overgenomen. Na creatie is de kloontabel uiteraard leeg: de data-inhoud is nu net het enige wat de basistabel en de kloontabel niet delen! Ook autorisaties worden niet overgenomen, en views op de basistabel worden niet gedupliceerd.

Na creatie gedragen de basistabel en de kloontabel zich als twee totaal onafhankelijke objecten. Beide kunnen dus, onafhankelijk van elkaar, gelezen en geschreven worden. Zo kan b.v. de kloontabel geladen worden met nieuwe gegevens, terwijl de basistabel toegankelijk blijft (met de oude gegevens).

Wanneer de kloontabel alle gegevens bevat die u wenst in te laden in de basistabel, kunnen basistabel en kloontabel geswitcht worden. Hiervoor wordt het nieuwe SQL-statement "EXCHANGE" gebruikt. Vanaf dat ogenblik gebeuren alle verwijzingen naar de kloontabel i.p.v. naar de basistabel, of beter gezegd: beide tabellen behouden hun naam, autorisaties, enz., maar hun data is verwisseld. Dit is mogelijk omdat de kloontabel in dezelfde tablespace zit als de basistabel, maar in een andere VSAM-dataset. De bestandsnaam is normaal gesproken van de vorm catname.DSNDBx.dbname.spname.I0001.A001. De bestandsnaam voor de kloontabel is identiek, behalve dat het vijfde deel van de naam I0002 is i.p.v. I0001. Het enige wat het EXCHANGE-statement dus doet, is het omwisselen van de VSAM-datasets.

In de catalog is de kloontabel herkenbaar doordat zijn TYPE (in SYSIBM.SYSTABLES) de waarde "C" heeft. Bovendien bevat de kolomcombinatie (TBCREATOR,TBNAME) de naam van de "andere" tabel. Tabellen die een kloontabel hebben, kunnen dus gevonden worden door de volgende query:

```
SELECT creator || '.' || name AS base_table
FROM   sysibm.systables
WHERE  type = 'T' AND tbname <> ' '
```

Voeg ", tbcreator||'.'||tbname AS clone_table" toe op de SELECT-lijn als je ook de naam van de kloontabel wil kennen. (In versies van DB2 vóór versie 8 zou "creator" (en "tbcreator") vervangen moeten worden door "RTRIM(creator)", omdat deze kolom pas sinds versie 8 het datatype VARCHAR heeft; in versie 9 is dat dus niet nodig.)

Ook SYSIBM.SYSTABLESPACE houdt bij wanneer er een kloontabel aanwezig is: de nieuwe kolom CLONE kan namelijk hetzij 'Y' hetzij 'N' zijn. Bovendien zal ook de indexspace van alle gekloonde indexen de waarde CLONE='Y' hebben.

Er zijn enkele beperkingen op het gebruik van klonen. Zo moet, om een kloontabel te mogen aanmaken, de basistabel in een universal tablespace zitten (naar keuze partitioned "by growth" of "by range"), en mag niet betrokken zijn in referentiële integriteit, dus geen foreign keys van of naar de tabel. Er zijn nog verschillende andere beperkingen, die meestal voor zich spreken. Zo kan een kloontabel niet opnieuw gekloond worden, kan de basistabel geen MQT, temporary table of catalog-tabel zijn, of mogen er geen AFTER-triggers op gedefinieerd zijn.

Ook na het aanmaken van een kloontabel blijven deze beperkingen bestaan: Zolang een tabel een kloon heeft, kunnen er geen AFTER-triggers of foreign keys op gedefinieerd worden, en kan er geen ALTER TABLE of RENAME TABLE meer op uitgevoerd worden (behalve dan ALTER TABLE basistabelnaam DROP CLONE). Anderzijds zullen alle BEFORE-triggers, check constraints en indexen die op de basis-

tabel gecreëerd (of verwijderd) worden, automatisch ook op de kloontabel gecreëerd (of verwijderd) worden. Nieuwe indexen op een niet-lege kloontabel worden automatisch in een rebuild-pending state geplaatst.

De “RI”-beperking (geen foreign keys) is niet helemaal logisch: een FK op een tabel én t.z.t. op z’n kloon zou geen probleem mogen zijn. Ook een NOT ENFORCED foreign key zou geen probleem mogen zijn. Kleine aanpassingen dus van het CLONE-model die wellicht in een volgende versie van DB2 zullen gebeuren.

Het (snel) omwisselen van de gegevens tussen basistabel en kloontabel gebeurt dus met het EXCHANGE-statement. De syntax is als volgt:

```
EXCHANGE DATA BETWEEN TABLE basistabelnaam AND kloontabelnaam
```

Er is geen nieuw type autorisatie voorzien voor het EXCHANGE-commando: er moet INSERT en DELETE-autorisatie zijn voor zowel de basistabel als de kloontabel.

Elke EXCHANGE kan uiteraard geCOMMIT of geROLLBACKed worden; kleine beperking hierbij: EXCHANGE moet in een “eigen” unit-of-work zitten, d.w.z.: er moet een expliciete COMMIT gebeuren vóór en na elke EXCHANGE.

Het uitvoeren van het EXCHANGE-statement gebeurt zeer snel: het enige wat echt gebeurt is het wijzigen van de namen van de twee VSAM-datasets. Op het ogenblik van de EXCHANGE wordt in de catalog een rij weggeschreven naar SYSIBM.SYSCOPY met STYPE='E', zodat latere RECOVER-acties weet zouden hebben van de bestandsnaamomwisseling. Verder worden de real-time-statistieken voor de basistabel gemarkeerd als ongeldig. Andere neveneffecten zijn er niet: alle PLANs en PACKAGEs blijven geldig, en de RUNSTATS-statistieken en de dynamic statement cache worden niet gewijzigd.

Beide tabellen, basis en kloon, hebben dezelfde tablespace-naam. Dit geldt ook voor elk van de indexen op de basistabel. Dit kan voor verwarring zorgen wanneer we naar slechts één van beide willen verwijzen, b.v. bij een UNLOAD. Daarom hebben de meeste utilities en commands er een keyword CLONE bijgekregen: zonder dit keyword verwijzen we naar de basistabel (of anders gezegd: naar het bestand met I0001 of J0001 in de naam), met CLONE verwijzen we naar de kloontabel (dus het I0002- of J0002-bestand). Dit geldt voor de commando's -START DATABASE en -STOP DATABASE en voor alle utilities. Behalve voor RUNSTATS: een kloontabel kan zelf geen statistieken hebben, enkel de basistabel!

XSLT genereert SQL

Peter Vanroose (ABIS)

In de vorige bijdragen over XML en DB2 hebt u kennis gemaakt met de manier waarop XML op een “native” manier binnen DB2 kan opgeslagen en vooral ook ondervraagd worden. Hiertoe bleek XPath een zeer belangrijk hulpmiddel, met name om te specificeren welk deel van de XML-boom we precies bedoelen. XPath is dus zo een beetje de XML-tegenhanger van SQL-predicaten in WHERE-condities.

In deze bijdrage maakt u kennis met een andere belangrijke component van XML, nl. XSLT, of voluit “Extensible Stylesheet Language Transformations”. XSLT is een XML-standaard voor het converteren van de inhoud van een XML-document naar een ander formaat (XML, HTML, tekst, ...). Een XSL-“stylesheet”-document is enerzijds een *well-formed* XML-document, dat bovendien aan de specificaties van XSLT voldoet, maar anderzijds eigenlijk een “algoritme”, een script of transformatie-programma dus, dat een (ander) XML-brondocument als input verwacht en als output (een deel van) de informatie uit dat XML-document op een andere manier laat zien.

DB2 9.5 for LUW heeft een scalaire functie `xsltransform` die een XSL-stylesheet kan “uitvoeren”. DB2 9 for z/OS ondersteunt XSLT nog niet. Toch kan XSLT zeer nuttig zijn binnen een DB2-systeem waar XML gebruikt wordt. Deze bijdrage probeert u hiervan te overtuigen door één specifieke toepassing van XSLT stapsgewijs uit te werken, namelijk het transformeren van meta-informatie over een XML-document (een XML-schema dus) naar meta-informatie over de SQL-gerelateerde informatie in dat document, met name de tabelstructuur die nodig is om die data op te slaan.

XSLT “uitvoeren” kan dan weliswaar niet binnen DB2, toch is het relatief eenvoudig om een XSL-stylesheet te laten “werken” op een XML-bronbestand om zo de output te verkrijgen. Elke web-browser bevat tegenwoordig XSLT, o.a. om XML-pagina's naar HTML te converteren, maar er zijn eveneens andere applicaties (zoals Altova, XALAN, SAXON) en command-line-tools die XSLT ondersteunen. Zo biedt b.v. Microsoft de XSLT-functionaliteit aan in een afzonderlijke windows-executable: `msxsl.exe`. Dit programma kan gedownload worden van de Microsoft-download-pagina's.

Voor details: zie *Bijlage 1* op p. 17.

Probleemstelling

Stel dat we een XML-document aangeleverd krijgen, met daarin data die we later mogelijk willen ondervragen vanuit DB2. We kunnen het document (voorlopig of definitief) opslaan in een xml-kolom van een DB2-tabel, maar op termijn zullen we wellicht XPath-expressies willen gebruiken in combinatie met één van de XML-functies van DB2 zoals XMLEXISTS of XMLQUERY. Een alternatief voor het opslaan van de XML-gegevens in DB2 is het “*shredden*” van deze gegevens, d.w.z. het interpreteren van de gegevens in de tekst-nodes van de XML-boom, en deze gegevens dadelijk op te slaan in velden van een DB2-tabel. Hierbij gaat mogelijk informatie verloren, want een tabel kent geen volgorde en geen hiërarchie. Maar in veel gevallen heeft de

XML-data een “platte” structuur en is de volgorde niet relevant. In dergelijke gevallen is het inderdaad zinvoller om de data “ge-de-XML-iseerd” of verknipt (“shredded”) op te slaan, zodat het latere ondervragen van de data efficiënter zal kunnen gebeuren, zonder de XML-overhead.

Een eerste hindernis bij het automatisch interpreteren van een XML-bestand, zelfs al heeft die een redelijk “platte” en dus verknipbare structuur, is de veelheid aan manieren waarop dezelfde data in XML kan voorgesteld worden. Precies daarvoor werden XML Schemas in het leven geroepen: een schema beschrijft de structuur waaraan een bepaald XML-document moet beantwoorden om “verwerkbaar” te zijn. De leverancier van ons XML-document (en van toekomstige documenten met een gelijkaardige structuur) moet ons dus ook een XML Schema aanleveren dat die structuur beschrijft. Of nog beter misschien: wij stellen zelf een XML Schema op en eisen van onze leveranciers dat hun XML-documenten hieraan moeten beantwoorden. In beide gevallen kunnen we, door het bekijken van het Schema, te weten komen waar in de XML-boom welke gegevens kunnen gevonden worden, zodat ze op basis van deze meta-informatie automatisch kunnen geëxtraheerd worden.

XSLT kan ons hierbij op twee niveau's helpen: enerzijds om een willekeurig, geldig XML-document te converteren naar een reeks INSERT-statements, de geshredde data dus, maar ook voor het converteren van het XML Schema naar een CREATE TABLE-statement waarin we de meta-data terugvinden, met name informatie over datatypes, maximale lengte van tekstvelden, NOT NULL-restricties, check constraints, enz. Het eerstgenoemde XSL-stylesheet zal zeer specifiek zijn voor een bepaald type document, maar voor de tweede toepassing kan er in principe één generiek XSL-stylesheet geschreven worden dat een willekeurig XML Schema kan “interpreteren” door de overeenkomstige SQL-gebaseerde meta-data te produceren! Bovendien kan er ook een generiek XSL-stylesheet geschreven worden dat vanuit een XML Schema een XSLT-bestand genereert van het eerste type, nl. één dat documenten die aan het Schema beantwoorden kan converteren naar een lijst van INSERT-statements die compatibel zijn met de tabel-declaratie gegenereerd door het eerste XSL-style-sheet!

Laten we even kijken hoe we een dergelijk “generiek” XSL-stylesheet kunnen samenstellen.

XSLT

De basisstructuur van een XSL-stylesheet is de volgende:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="XPath-expressie">
  TEKST (+ INSTRUCTIES)
 </xsl:template>
</xsl:stylesheet>
```

Voor elke gevonden match, d.w.z. voor elke element-node die voldoet aan de gegeven XPath-expressie, wordt TEKST als output gegenereerd. Deze “tekst” kan uiteraard een aantal “xsl”-instructies

bevatten als plaatshouders voor kopieën van (stukken van) het input-document. Stel b.v. dat we voor elk matchend knooppunt enkel z'n tekstuele inhoud willen uitprinten, dan gebruiken we als TEKST:

```
<xsl:value-of select="."/>
```

De entry "<xsl:value-of>" kopieert namelijk alle tekstinhoud van (de eerste node die voldoet aan) de XPath-expressie opgegeven in het "select"-attribuut. Hierbij fungeert de "template match" als referentie-node, wat verklaart waarom "." inderdaad de tekstinhoud van de referentienode oplevert.

Een tweede nuttige ingrediënt in XSLT is "<xsl:for-each>", een constructie die toelaat over een lijst van nodes te itereren. Deze lijst wordt uiteraard eveneens gespecificeerd m.b.v. een XPath-expressie. Zo produceert het onderstaande XSL-stylesheet voor elke node met de naam "Company" een lijst van de attributen genaamd "name" van al diens child nodes met de naam Employee:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="//Company">
<xsl:for-eachselect="./Employee"><xsl:value-ofselect="@name"/></xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

Binnen een "xsl:for-each" is de context-node uiteraard achtereenvolgens elk van de nodes in de nodelist, "./Employee" in dit geval, d.w.z. alle child-nodes van een "Company"-node die de naam "Employee" hebben. De "./" is hier trouwens overbodig.

De output zal per "Company"-node bestaan uit een herhaling van een spatie gevolgd door de waarde van een "name"-attribuut, dit alles gevolgd door een end-of-line.

Een XML Schema als input

Terug naar de oorspronkelijke vraag: het opstellen van een generiek XSL-stylesheet dat, uitgaande van het XML Schema voor XML-data met een "platte" structuur, een "CREATE TABLE"-statement produceert met de juiste kolomnamen, datatypes, enz.

Om alles voldoende concreet te houden, concentreren we ons op het volgende XML-bestand als voorbeeld:

```
<?xml version="1.0" encoding="UTF-8"?>
<Companies>
 <Company nr="3"><name>ABIS</name><street>Diestsevest</street><strno>32</strno>
                 <town>Leuven</town></Company>
 <Company nr="9"><name>Technisoft</name><tel>02977-22456</tel>
                 <town>Rotterdam</town></Company>
</Companies>
```

Een "Company" heeft een naam ("name"-subnode), een nummer ("nr"-attribuut), en optioneel een aantal andere gegevens zoals "tel", "town", enz. Een lijst van dergelijke nodes vormt samen de "Companies"-node. Dit is inderdaad een voldoende "platte" structuur om rechtstreeks op een tabel te mappen: een tabel is namelijk een "lijst" van

rijen; elke rij (“Company”) heeft een aantal velden (de subnodes en/of de attributen van de “Company”-node). Optionele velden die niet aanwezig zijn, worden door een NULL voorgesteld in de tabel.

Het volgende XML Schema beschrijft dat een geldig XML-bestand inderdaad precies één node “Companies” moet hebben, met daaronder een lijst van “Company”-nodes, die elk verplicht een attribuut “nr” moeten hebben, een verplichte subnode “name”, optionele nodes “tel”,“street”,“strno” en verplichte subnode “town” (in die volgorde):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Company" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="name" type="xs:string"/>
       <xs:element name="tel" type="xs:string" minOccurs="0"/>
       <xs:element name="street" type="xs:string" minOccurs="0"/>
       <xs:element name="strno" type="xs:integer" minOccurs="0"/>
       <xs:element name="town" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="nr" type="xs:integer" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Een XML Schema met een dergelijke volledig geneste structuur wordt een “*Russian Doll design*” genoemd. Alternatieve designs komen verderop aan bod.

De defaults voor minOccurs en maxOccurs zijn beide 1: elke elementnode die in het Schema vermeld wordt, moet normaal gesproken exact éénmaal voorkomen. Dit verklaart de aanwezigheid van minOccurs=0 voor “optionele” entries, en van maxOccurs="unbounded" voor een lijst van willekeurige lengte, een tabel dus.

Een eerste poging

Het volgende, relatief eenvoudige XSL-stylesheet matcht op een node met naam “xs:element” die een attribuut “maxOccurs” heeft met als waarde “unbounded”. Diens overgrootvader-element zal als “name”-attribuut de naam van de tabel bevatten, en dat is het eerste wat we nodig hebben om een CREATE TABLE-statement te genereren.

Daarna worden alle nodes met naam “xs:element” doorlopen (via xsl:for-each) die nog eens drie niveau's dieper zitten.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xsl:output method="text" omit-xml-declaration="yes"/>
<xsl:templatematch="/xs:schema/xs:element/xs:complexType/xs:sequence/xs:element
  [@maxOccurs='unbounded']">
 CREATE TABLE <xsl:value-of select="../../../@name"/> (
```

```
  <xsl:for-each select="xs:complexType/xs:sequence/xs:element">
   <xsl:value-of select="@name"/> VARCHAR(255)
    <xsl:if test="not(@minOccurs='0')"> NOT NULL</xsl:if>,
  </xsl:for-each>
  <xsl:for-each select="xs:complexType/xs:attribute">
 <xsl:value-of select="@name"/> INT <xsl:if test="@use='required'"> NOT NULL</xsl:if>,
  </xsl:for-each>);
 </xsl:template>
</xsl:stylesheet>
```

Bemerk de constructie "xsl:if", die toelaat om conditioneel bepaalde output (in dit geval: "NOT NULL") te genereren. De conditie wordt meegegeven via het "test"-attribuut van xsl:if. Bemerk ook de tweede xsl:for-each, die alle attributen (in het voorbeeld: "nr") doorloopt. Ten slotte valt ook het xsl:output element op: de gegenereerde output is geen XML, dus willen we geen XML header-declaratie laten genereren.

Dit XSL-stylesheet genereert de volgende output, wanneer losgelaten op bovenstaande XML Schema:

```
CREATE TABLE Companies (
   name VARCHAR(255) NOT NULL,
   tel VARCHAR(255),
   street VARCHAR(255),
   strno VARCHAR(255),
   town VARCHAR(255) NOT NULL,
   nr INT NOT NULL,
);
```

Belangrijkste fout in deze output is uiteraard de laatste komma. Om die te vermijden, moet er binnen de xsl:for-each een test komen die verifieert of er al dan niet een volgende node op komst is in de xsl:for-each.

Iets eenvoudiger is het, te verifiëren of de huidige iteratie binnen een xsl:for-each de eerste is, nl. met de conditie <xsl:if test="position() = 1">. Plaats dus een komma vóór (i.p.v. na) elke entry, behalve vóór de eerste:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xsl:output method="text" omit-xml-declaration="yes"/>
<xsl:templatematch="/xs:schema/xs:element/xs:complexType/xs:sequence/xs:element
  [@maxOccurs='unbounded']">
 CREATE TABLE <xsl:value-of select="../../../@name"/> (
  <xsl:for-each select="xs:complexType/xs:sequence/xs:element">
   <xsl:if test="position() &gt; 1">
  , </xsl:if><xsl:value-of select="@name"/> VARCHAR(255)
   <xsl:if test="not(@minOccurs='0')"> NOT NULL</xsl:if>
  </xsl:for-each>
  <xsl:for-each select="xs:complexType/xs:attribute">
  , <xsl:value-of select="@name"/> INT
   <xsl:if test="@use='required'"> NOT NULL</xsl:if>
  </xsl:for-each>
 );
 </xsl:template>
</xsl:stylesheet>
```

Datatypes en andere verfijningen

Nog niet bevredigend is uiteraard het feit dat alle velden het datatype "VARCHAR(255)" meekrijgen. Dit is echter zo ongeveer de enige zinvolle vertaling van het datatype dat in XML Schema wordt gespecificeerd, met name "xs:string". Voor numerieke datatypes zoals "xs:integer" kan (en moet) er uiteraard een aangepast SQL-datatype gegenereerd worden. Ook voor tekst-velden kan het in sommige gevallen beter, nl. als er een maximale lengte opgegeven wordt. Stel b.v. dat het XML Schema de volgende specificatie bevat voor telefoonnummer:

```
<xs:element name="tel" minOccurs="0"><xs:simpleType>
<xs:restriction base="xs:string"><xs:maxLength value="16"/></xs:restriction>
</xs:simpleType></xs:element>
```

Op basis hiervan kan het SQL-datatype aangepast worden naar hetzij CHAR(16), hetzij VARCHAR(16). Een "hetzij...hetzij"-constructie wordt in XSLT gemaakt met <xsl:choose> gecombineerd met meerdere <xsl:when>. Vervang dus in het XSL-stylesheet dat we tothiertoe hebben, de uitdrukking "VARCHAR(255)" door b.v.:

```
<xsl:choose>
 <xsl:when test="@type='xs:string'"> VARCHAR(255)</xsl:when>
 <xsl:when test="@type='xs:integer'"> INT</xsl:when>
 <xsl:when test="@type='xs:date'"> DATE</xsl:when>
 <xsl:when test="xs:simpleType/xs:restriction/@base='xs:string'">
  CHAR(<xsl:value-of
     select="xs:simpleType/xs:restriction/xs:maxLength/@value"/>)</xsl:when>
 <xsl:otherwise> Unknown_DataType</xsl:otherwise>
</xsl:choose>
```

Voorlopig vangen we eventuele "andere datatypes" dus op met het onbestaande datatype "Unknown_DataType". Hier kunnen later, indien nodig, meer gevallen toegevoegd worden.

Zie *Bijlage 2* op p. 18 voor meer gevallen.

Een ander soort XML Schema "restriction" kan bestaan uit een opsomming. Zo zou b.v. een "country"-veld door het Schema beperkt kunnen worden tot de waarden 'BE', 'NL', 'LU', als volgt:

```
<xs:element name="country">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="BE"/>
   <xs:enumeration value="NL"/>
   <xs:enumeration value="LU"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Hieruit zou het XSL-stylesheet de bijkomende specificatie "CHECK (country IN ('BE','NL','LU'))" moeten genereren. Dat kan door het volgende fragment toe te voegen net vóór het einde van de xsl:for-each:

```
<xsl:if test="xs:simpleType/xs:restriction[xs:enumeration]">
 CHECK (<xsl:value-of select="@name"/> IN (
  <xsl:for-each select="xs:simpleType/xs:restriction/xs:enumeration">
   <xsl:if test="position()&gt;1">,</xsl:if>
```

```
   '<xsl:value-of select="@value"/>'
  </xsl:for-each>) )
</xsl:if>
```

Bemerk de geneste `xsl:for-each`, met opnieuw het gebruik van `position()` om te vermijden dat er een extra komma wordt gegenereerd.

Ten slotte is nog een laatste verfijning mogelijk, die ervoor zorgt dat ook XML Schema's die geen zuivere "Russian Doll"-design hebben, zoals b.v. het onderstaande, toch correct geconverteerd worden:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="name" type="xs:string"/>
 <xs:element name="street">
  <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="40"/></xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Company" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Company">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="street" minOccurs="0"/>
    <xs:element name="town" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

In een "*Salami Slice*"-design wordt trouwens elke complexe node op die manier gespecificeerd. Dat verklaart mede waarom het zoeken op "unbounded" nodig was om het juiste "startpunt" (nl. element name "`Companies`") eenduidig terug te vinden.

De genoemde verfijning bestaat erin, om binnen de `xsl:for-each` een variabele te introduceren, die als inhoud de nodelist zal krijgen bestaande uit hetzij enkel de "current node" (indien er een "`name=`" attribuut in voorkomt), hetzij de current node en de gerefereerde node (indien "`ref=`" voorkomt).

Dit kan met behulp van het volgende XSLT-fragment, dat als eerste statement binnen de `xsl:for-each` moet worden opgenomen:

```
 <xsl:variable name="refnode"
               select=".|/xs:schema/xs:element[@name=current()/@ref]"/>
```

De variabele heet dus "`refnode`" en z'n inhoud kan met de uitdrukking "`$refnode`" gebruikt worden. Door de vertikale streep "|" in de select wordt dus een node-set van lengte 2 gegenereerd, waarbij de eerste of de tweede component mogelijk onbestaand (en dus afwezig) is. (In dit geval kan uiteraard enkel de tweede afwezig zijn.)

Deze variabele moet systematisch gebruikt worden in elke XPath-uitdrukking binnen de `xsl:for-each`, in de plaats van de current node ".". Zo moet b.v. de uitdrukking

```
<xsl:when test="@type='xs:string'">
```

vervangen worden door

```
<xsl:when test="$refnode/@type='xs:string'">
```

Ook de XPath-expressie in de `xsl:for-each` zelf moet op een gelijkaardige manier aangepast worden, omdat de node "`Company`" eveneens een "`ref`" kan zijn:

```
 <xsl:variable name="rownode"
               select="xs:.|/xs:schema/xs:element[@name=current()/@ref]"/>
 <xsl:for-each select="$rownode/xs:complexType/xs:sequence/xs:element">
```

Als ten slotte alle ingrediënten in elkaar gepast worden, krijgen we een generiek XSL-stylesheet voor het converteren van een willekeurig XML Schema naar een bijhorend "`CREATE TABLE`" SQL-statement..

Zie *Bijlage 2* op p. 18 voor de finale stylesheet.

Prijsvraag

Op een zeer gelijkaardige manier moet het mogelijk zijn om datzelfde XML-Schema te converteren naar een XSL-stylesheet dat SQL "`INSERT`"-statements genereert uit een bijhorend XML-document (d.w.z., een document dat geldig is voor het XML-Schema).

Schrijf een dergelijk XSL-stylesheet, dat dus een XML-Schema converteert naar een "dedicated" XSL-stylesheet voor conversie van XML naar SQL INSERT, en maak kans op een prijs! De beste inzending ontvangt namelijk een waardebon voor gratis deelname aan één van de avondsessies "Exploring DB2 for z/OS - live!". De inzendingen worden beoordeeld op correctheid, elegantie en volledigheid door XML-specialisten van ABIS.

Om in aanmerking te komen, moet uw inzending toekomen op het volgende emailadres vóór 1 februari 2009: `training@abis.be`. De winnaar (en het winnende XSL-stylesheet) worden gepubliceerd in het volgende nummer van *Exploring DB2*. Veel succes!

Uiteraard ontvangen we ook graag andere reacties of aanvullingen op dit artikel, b.v. verfijningen van het XSL-stylesheet in bijlage, of varianten voor een "tabelstructuur"-XML-Schema, die uitgaan van een ander mogelijk datamodel voor het opslaan in XML van tabel-data.

CURSUSPLANNING JANUARI - MEI 2009

DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	12.01(L), 02.03(L), 23.03(W), 11.05(W)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	12.02(L), 11.06(W)
RDBMS-concepten	375 EUR	12.01(L), 02.03(L), 23.03(W), 11.05(W)
Basiskennis SQL	375 EUR	13.01(L), 03.03(L), 24.03(W), 12.05(W)
XML concepten	450 EUR	19.01(L), 20.02(W), 06.04(L), 11.05(W)
XML basiscursus	1200 EUR	20.01(L), 25.02(W), 07.04(L), 12.05(W)
XSLT	850 EUR	16.02(L), 22.04(W), 18.06(L)
DB2 for z/OS basiscursus	1275 EUR	14.01(L), 04.03(L), 25.03(W), 13.05(W)
DB2 UDB for LUW basiscursus	1275 EUR	19.02(L), 18.06(W)
SQL-QMF voor eindgebruikers	1275 EUR	06.05(W)
SQL workshop	800 EUR	22.01(L), 16.03(L), 02.04(W), 25.05(W)
SQL voor gevorderden	450 EUR	23.04(W), 29.06(L)
DB2 procedural extensions	450 EUR	24.04(W), 30.06(L)
DB2 for z/OS programmeren voor gevorderden	900 EUR	16.04(L), 11.06(W)
DB2 for z/OS: SQL performance	1350 EUR	23.03(L), 27.05(W)
XML in DB2	450 EUR	29.04(L), 26.05(W)
DB2 for z/OS database administratie	1900 EUR	09.02(W), 04.05(L)
DB2 for z/OS installation & migration	1050 EUR	23.02(UK), 01.06(UK)
DB2 for z/OS operations & recovery	1425 EUR	25.02(UK), 04.05(W), 03.06(UK)
DB2 for z/OS systems performance and tuning	1000 EUR	09.02(UK), 27.04(UK), 07.05(W)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	16.02(L), 09.06(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Exploring DB2 - live!	175 EUR	27.01(L), 26.02(L), 26.03(L), 27.04(L) (avonden)

Plaats: L = Leuven, W = Woerden, UK = High Wycombe (bij Londen);
voor details en andere cursussen, zie `http://www.abis.be/html/nlTraining.html`

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245639
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Pelmolenlaan 1-K
NL-3440 AC Woerden
Tel. 0348-435570
Fax 0348-432493
training@abis.be

Bijlage 1

Downloaden van msxsl.exe kan vanaf de volgende URL:

```
http://download.microsoft.com/download/f/2/6/f263ac46-1fe9-4ae9-8fd3-
21102100ebf5/msxsl.exe
```

Bijlage 2

Hieronder vindt u een XSL-stylesheet dat uit een XML Schema een bijhorend “CREATE TABLE”-statement zal genereren. Voorbeelden van een dergelijk Schema, en van validerende XML, vindt u verderop.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xsl:output method="text" omit-xml-declaration="yes"/>
 <xsl:template match=

"/xs:schema/xs:element/xs:complexType/xs:sequence/xs:element[@maxOccurs='unbo
unded']">
 CREATE TABLE <xsl:value-of select="../../../@name"/> (
  <xsl:variable name="rownode" select=
       ".|/xs:schema/xs:element[@name=current()/@ref]"/>
  <xsl:for-each select="$rownode/xs:complexType/xs:sequence/xs:element">
   <xsl:variable name="refnode" select=
        ".|/xs:schema/xs:element[@name=current()/@ref]"/>
   <xsl:variable name="rstr" select="$refnode/xs:simpleType/xs:restriction"/>
   <xsl:if test="position()&gt;1">
  , </xsl:if><xsl:value-of select="@name|@ref"/>
   <xsl:choose>
    <xsl:when test="$refnode/@type='xs:string'"> VARCHAR(255)</xsl:when>
    <xsl:when test="$refnode/@type='xs:int'"> INT</xsl:when>
    <xsl:when test="$refnode/@type='xs:integer'"> INT</xsl:when>
    <xsl:when test="$refnode/@type='xs:date'"> DATE</xsl:when>
    <xsl:when test="$refnode/@type='xs:time'"> TIME</xsl:when>
    <xsl:when test="$rstr/@base='xs:date'"> DATE</xsl:when>
    <xsl:when test="$rstr/@base='xs:time'"> TIME</xsl:when>
    <xsl:when test="$rstr/@base='xs:integer'">
     <xsl:choose>
  <xsl:when test="$rstr/xs:maxInclusive/@value&lt;'32768'"> SMALLINT</xsl:when>
      <xsl:otherwise> INT</xsl:otherwise>
     </xsl:choose>
    </xsl:when>
    <xsl:when test="$rstr/@base='xs:decimal'"> DECIMAL(<xsl:value-of select=
         "$rstr/xs:totalDigits/@value"/>,<xsl:value-of select=
         "$rstr/xs:fractionDigits/@value"/>)</xsl:when>
    <xsl:when test="$rstr/@base='xs:string'">
     <xsl:choose>
  <xsl:when test="$rstr/xs:maxLength/@value&lt;'25'">CHAR(<xsl:value-of select=
          "$rstr/xs:maxLength/@value"/>)</xsl:when>
<xsl:when test="$rstr/xs:maxLength/@value&gt;'0'">VARCHAR(<xsl:value-of select=
          "$rstr/xs:maxLength/@value"/>)</xsl:when>
      <xsl:otherwise> VARCHAR(255)</xsl:otherwise>
     </xsl:choose>
    </xsl:when>
    <xsl:otherwise> Unknown_DataType</xsl:otherwise>
   </xsl:choose>
   <xsl:if test="not($refnode/@minOccurs='0')"> NOT NULL</xsl:if>
   <xsl:if test="$rstr[xs:enumeration]">
   CHECK (<xsl:value-of select="$refnode/@name"/> IN (<xsl:for-each select=
          "$rstr/xs:enumeration"><xsl:if test=
          "position()&gt;1">,</xsl:if>'<xsl:value-of select=
          "@value"/>'</xsl:for-each>) )</xsl:if>
  </xsl:for-each>
```

```
  <xsl:for-each select=
       "$rownode/xs:complexType/xs:attribute[@type='xs:integer']">
  , <xsl:value-of select="@name|@ref"/>
  INT<xsl:if test="@use='required'"> NOT NULL PRIMARY KEY</xsl:if>
  </xsl:for-each>
 );
 </xsl:template>
</xsl:stylesheet>
```

En enkele voorbeelden van XML Schema's die als input kunnen dienen voor het bovenstaande XSL-stylesheet, telkens met de bijhorende output:

1. puur “Russian Doll”:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Company" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Company">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="tel" minOccurs="0">
     <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="16"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="street" type="xs:string" minOccurs="0"/>
    <xs:element name="strno" type="xs:string" minOccurs="0"/>
    <xs:element name="townno" minOccurs="0">
     <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="10"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="town" type="xs:string"/>
    <xs:element name="country">
     <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="4"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="c_pno" type="xs:integer" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="nr" type="xs:integer" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

---

2. puur “Salami Slice”, dus allemaal met “ref=”, d.w.z.: de kolommen worden apart gedeclareerd, en de tabel is een sequence van verwijzingen naar deze declaraties.

De “minOccurs” (d.w.z.: “NOT NULL” of niet) zit uiteraard nooit mee in die declaratie maar altijd in de tabel zelf:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="street" type="xs:string"/>
 <xs:element name="strno" type="xs:string"/>
 <xs:element name="town" type="xs:string"/>
 <xs:element name="townno">
  <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="10"/></xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="country">
  <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="4"/></xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="tel">
  <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="16"/></xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="c_pno" type="xs:integer"/>
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Company" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Company">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="tel" minOccurs="0"/>
    <xs:element ref="street" minOccurs="0"/>
    <xs:element ref="strno" minOccurs="0"/>
    <xs:element ref="townno" minOccurs="0"/>
    <xs:element ref="town"/>
    <xs:element ref="country"/>
    <xs:element ref="c_pno" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="nr" type="xs:integer" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

3: met een mengeling van “Russian Doll” en “Salami Slice”, en met bovendien een “xs:restriction” die naar een CHECK vertaald wordt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="country">
  <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="4"/>
    <xs:enumeration value="B"/>
    <xs:enumeration value="NL"/>
    <xs:enumeration value="D"/>
    <xs:enumeration value="GB"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="tel">
  <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="16"/></xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Company" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Company">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="tel" minOccurs="0"/>
    <xs:element name="street" type="xs:string" minOccurs="0"/>
    <xs:element name="strno" type="xs:string" minOccurs="0"/>
    <xs:element name="townno" minOccurs="0">
     <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="10"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="town" type="xs:string"/>
    <xs:element ref="country"/>
    <xs:element name="c_pno" type="xs:integer" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="nr" type="xs:integer" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Een XML-bronbestand (dat valideert met elk van de drie bovenstaande XML Schema's):

```
<?xml version="1.0" encoding="UTF-8"?>
<Companies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="Companies.xsd">
 <Company no="3">
  <name>ABIS</name>
  <tel>016/245610</tel>
  <street>Diestsevest</street>
  <strno>32</strno>
  <townno>3000</townno>
  <town>Leuven</town>
  <country>B</country>
  <c_pno>1</c_pno>
 </Company>
 <Company no="9">
  <name>Technisoft</name>
  <tel>02977-22456</tel>
  <town>Rotterdam</town>
  <country>NL</country>
 </Company>
</Companies>
```

Output van het XSL-stylesheet voor elk van deze Schema's, indien toegepast op dit XML-bronbestand:

```
CREATE TABLE Companies (
 name VARCHAR(255) NOT NULL
, tel VARCHAR(16)
, street VARCHAR(255)
, strno VARCHAR(255)
, townno CHAR(10)
, town VARCHAR(255) NOT NULL
, country CHAR(4) NOT NULL
, c_pno INT
, nr INT NOT NULL PRIMARY KEY
);
```

Voor de prijsvraag wordt een XSL-stylesheet gevraagd dat uit elk van de drie Schema's een (nieuw) XSL-stylesheet moet genereren dat het bovenstaande XML-bronbestand converteert naar het volgende:

```
INSERT INTO Companies (nr,name,tel,street,strno,townno,town,country,c_pno)
 VALUES ('3','ABIS','016/245610','Diestsevest','32','3000','Leuven','B','1');
INSERT INTO Companies (nr,name,tel,town,country)
 VALUES ('9','Technisoft','02977-22456','Rotterdam','NL');
```