XML meer dan een handig informatie-uitwisselingsformaat

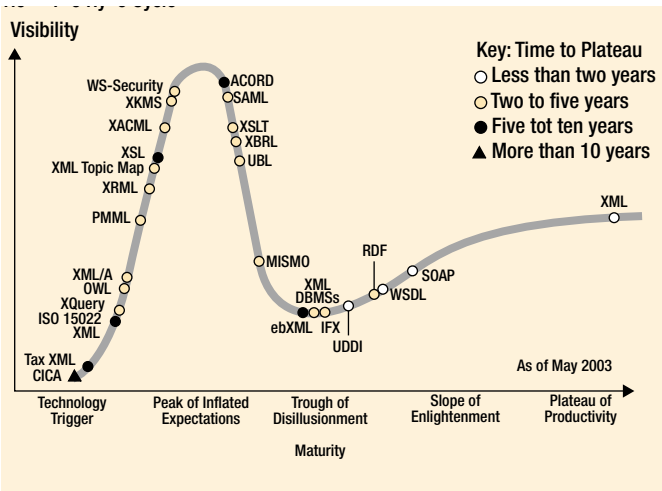
Native XML-databases steeds volwassener

Alex Stroom

Nu XML gemeengoed geworden is, vooral vanwege de uitwisselbaarheid van het dataformaat, is het logisch dat ICT-bedrijven er zoveel mogelijk aan doen om producten en applicaties te voorzien van XML-mogelijkheden. Ook de database-producten worden/zijn XML gereed gemaakt.

In Gartner's 'strategic analysis report' van mei 2003 is aardig te zien (Afbeelding 1) hoe een en ander zich ontwikkelt. We zien dat de standaarden ten behoeve van 'interoperability': XML, SOAP (Simple Object Access Protocol) en WSDL (Web Service Description Language) behoorlijk volwassen worden en dat binnen afzienbare tijd deze standaarden volop in toepassingen en producten te vinden zullen zijn.

De XML dbms-en zitten nog in het 'dal van desillusie' (Afbeelding 2), maar de ergste negatieve publiciteit is inmiddels toch wel wat verstomd. Een behoorlijk aantal database-producten is al ronduit goed en betrouwbaar te noemen. Dus tijd om er eens serieus aandacht aan te besteden. Mis nu de boot niet omdat het onderwerp 'uit' is, is de boodschap die bij deze plaats in de Hype Cycle hoort.
De opkomst van XML heeft niet alleen te maken met de eenvoud van het dataformaat of de populariteit van het internet, alhoewel dat wel de basis vormt voor moderne service-georiënteerde architecturen, maar vooral ook omdat het uit te leggen is aan IT-klanten (eindgebruikers- participatie). De instanties die verantwoordelijk zijn voor het definiëren van open informatie-standaarden, die als XML-structuren worden vastgelegd, bestaan niet alleen uit vertegenwoordigers van ICT- bedrijven, maar vooral ook uit een belangrijke afvaardiging van de business. De OASIS-site[1] geeft een zeer goed beeld van diverse branche initiatieven. Op de site staat ook een 'Registry' met de beschikbare verticale, branche gerichte XML standaarden.
Voorbeelden zijn: HR-XML – Human Resource informatie, XBRL – Business Reporting informatie, OFX – Financiële gegevens uitwisseling, UBL – business documenten, etc, etc...



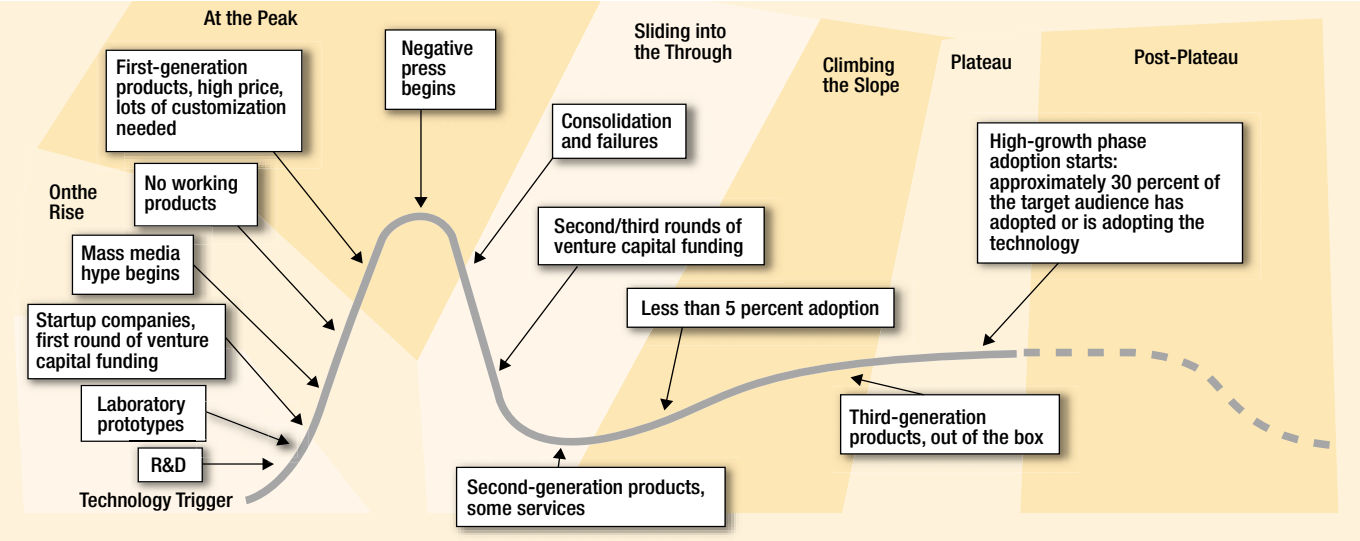
Afbeelding 1. Maturity Cycle. (bron: Gartner)

Het gevolg is dat steeds meer informatie gaat voldoen aan dit soort open XML-standaardstructuren. Hoe gaan we hier mee om?

Informatie Modellering

Om XML-definities vast te leggen is het gebruikelijk om de internationale W3C-standaarden toe te passen[2].
'XML Schema' (het W3C-model om XML-structuren te beschrijven) is een modelleringstechniek die afgeleid is van SGML's DTD (Document Type Definition) uit 1978; de hiërarchische manier om op het oog ongestructureerde informatie (zoals documenten) toch te voorzien van structuur. XML-objecten worden dan ook aangeduid als documenten. De DTD en W3C Schema modelleringstechnieken bieden zeer flexibele mogelijkheden en worden toegepast bij de eerder aangegeven branche initiatieven.

De vraag is nu of het mogelijk is om DTD structuren ook in de bekende relationele databases te gebruiken. Is een XML-document met een relationeel model te definiëren? Als voorbeeld



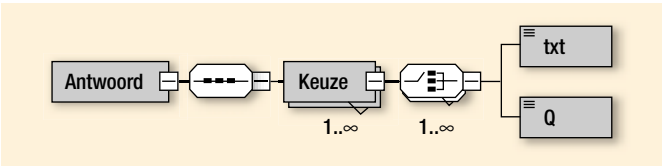
Afbeelding 2. Fasen van de Hype Cycle. (bron: Gartner)

nemen we een aantal keuze antwoorden op een multiple choice vraag. Bijvoorbeeld de vraag: 'Hoeveel kleine kubussen passen er in de grote kubus?'

De antwoorden zijn in een XML-formaat opgeslagen. XML-informatie is eigenlijk één enorme lange tekststring, maar wordt voor de leesbaarheid meestal geïndenteerd of als boomstructuur weergegeven. Elk stukje informatie tussen de vishaakjes < >, maar ook elk stukje informatie niet tussen de vishaakjes, wordt een *node* genoemd. Door middel van de W3C-standaarden XPath en XQuery kan elke node worden geadresseerd voor verwerking. Het antwoord op de kubus vraag kan er in XML als volgt uitzien:

```
<Antwoord>
<Keuze><txt>Er zitten</txt><Q>minder dan
125</Q><txt>kleine kubussen in de grote
kubus</txt></Keuze>
<Keuze><txt>Er zitten</txt><Q>precies
125</Q><txt>kleine kubussen in de grote kubus
</txt></Keuze>
<Keuze><txt>Geen van beide antwoorden is
correct</txt></Keuze>
</Antwoord>
```

Dit stukje XML is volgens het in Tabel 1 gegeven documenten-model (de DTD) gedefinieerd.
De elementen <txt> en <Q> bevatten alleen data en zijn de 'simpele' elementen. In elementen zoals <Antwoord> en <Keuze> zijn andere kenmerken dan data opgenomen en worden daarom 'complexe' elementtypen genoemd.
Zo'n DTD kan direct omgezet worden naar W3C Schema en grafisch worden gepresenteerd (Afbeelding 3).



Afbeelding 3.

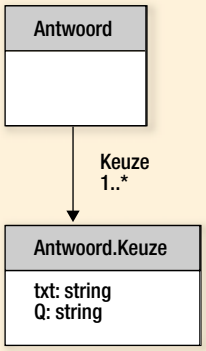
Wanneer een XML model omgezet moet worden naar een relationeel model is het handig om als tussenstap het model eerst om te zetten naar een UML objecten model. Als een soort vuistregel voor het omzetten van XML model naar een UML-objectenmodel, worden de complexe elementen omgezet naar class-objecten en simpele elementen naar attributen van een class- object (Afbeelding 4).

Zo'n UML class diagram is eenvoudig om te zetten naar een relationeel model (Afbeelding 5).

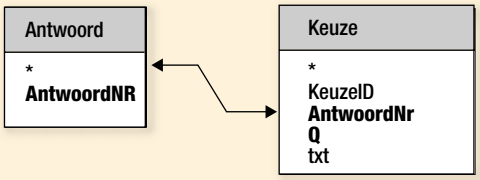
In de werkelijkheid echter zal het oorspronkelijke stukje XML waarschijnlijk anders gemodelleerd zijn dan in het voorbeeld. Logischerwijze wordt gestreefd naar de eenvoudigste manier om

<!ELEMENT Antwoord (Keuze+)>	Betekent: Het element <Antwoord> bevat 1 of meer keren het element <Keuze>
<!ELEMENT Keuze (txt | Q)+>	Betekent: Het element <Keuze> bestaat 1 of meer keren uit elementen <txt> of <Q>
<!ELEMENT Q (#PCDATA)>	Betekent: Het element <Q> bevat pcdata, ofwel tekst.
<!ELEMENT txt (#PCDATA)>	Betekent: Het element <txt> bevat pcdata, ofwel tekst

Tabel 1.

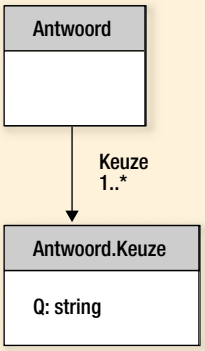


Afbeelding 4.



Afbeelding 5.

Afbeelding 6.



Afbeelding7.

de informatie structuur te beschrijven. Dezelfde informatie van het voorbeeld, kan eenvoudiger in XML zijn gedefinieerd. Hieronder bestaat het element <Keuze> uit losse tekst en uit het element <Q>.

```
<Antwoord>
<Keuze>Er zitten<Q>minder dan 125</Q>kleine kubus-
sen in de grote kubus</Keuze>
<Keuze>Er zitten<Q>precies 125</Q>kleine kubussen
in de grote kubus</Keuze>
<Keuze>Geen van beide antwoorden is correct</Keuze>
</Antwoord>
```

Het documenten model wordt nu als in Tabel 2.
Deze DTD kan weer worden omgezet naar een W3C schema, waarna dit schema grafisch gerepresenteerd kan worden als (Afbeelding 6):

Hier is eenvoudig te zien dat <Antwoord> meerdere keren bestaat uit <Keuze>, waarbinnen vervolgens <Q> 0 of meerdere keren voor mag komen. <Keuze> is nog steeds als complex type gedefinieerd en mag naast <Q> ook data bevatten.

Waar blijft in het UML-modelletje (Afbeelding 7) nu de informatie die als data in element <Keuze> is opgenomen? Hier ontstaat een probleem voor relationele databases. Elke poging om dit soort 'mixed content' op te slaan in relationele databases heeft tot gevolg dat er een en ander aan de informatie zelf moet worden aangepast. Maar dat is niet erg realistisch.
Veronderstel nu dat de data van element <Q> op commando vervangen moeten worden door andere data, bv 'meer dan 125' of '100'.
Hoe wordt dan zo'n actie uitgevoerd? Een update van de tekst binnen element <Q> is in XML direct uitvoerbaar doordat de data van element <Q> in zijn context via een XPath of XQuery-expressie adresseerbaar is. Dus bijvoorbeeld: vervang de #pcdata van <Q> in de tweede <Keuze> van <Antwoord> door.....

In de relationele wereld wordt toch met enige ingrepen het stukje XML in een aangepaste vorm opgesplitst naar een tabelstructuur (zoals eerder in Afbeelding 5); in een relationele database opgeslagen en een update functie op losgelaten. Echter het oorspronkelijke stukje tekst weer uit de database terug produceren is nog een hele klus omdat de context waarin <Q> stond weg is.

<!ELEMENT Antwoord (Keuze+)>	Betekent: element 'Antwoord' bestaat uit 1 of meer instanties van het element 'Keuze'.
<!ELEMENT Keuze (#PCDATA | Q)*>	Betekent: het element 'Keuze' bestaat 0 of meer keren uit of wel pcdata of wel het element 'Q'
<!ELEMENT Q (#PCDATA)>	Betekent: het element 'Q' bevat data.
Let wel: #PCDATA is ook een XML-node en dus apart te adresseren	

Tabel 2

Hoe wordt context en volgorde van het originele XML document bewaard in de tabellen? Ook daar zullen weer extra slimme trucs (een extra volgorde tabel bijvoorbeeld) voor moeten worden bedacht. Dit zogenaamde *round-trippen*, een XML-document opslaan en weer precies hetzelfde terugvragen, blijft het grote probleem bij object- en relationele conversies. Veel eenvoudiger is natuurlijk een oplossing waar je gewoon het hele XML-document als Blob of Clob in de database opslaat. Round-tripping is dan geen probleem. Maar helaas ontstaat er dan weer een ander probleem, want hoe is nu een directe update doen op het kwalificatie element <Q>, of alleen alle soorten kwalificaties <Q> selecteren?

> Ervaring met XML-databases heeft geleerd dat de discussie over normalisatie onverminderd uitdagend blijft

En dan zijn er ook nog de talloze andere eigenschappen van XML-modellering die attentie nodig hebben, zoals hoe om te gaan met Attributen, Entities, ProcessInstructions, Unicode, etc. De vraag blijft: 'Hoe passen rijke XML-modellen in een database-model?'
Om meer inzicht te krijgen in deze materie zijn door Ronald Bourret (ICT-freelancer) een aantal zeer goede artikelen gepubliceerd[3] en geeft Wrox Press's *Professional XML Databases* een 18-tal suggesties voor het modelleren van een DTD naar een relationeel model.
Inmiddels zijn alle bekende relationele databases uitgerust met gereedschappen (zelfs grafische 'mapping tools') om eenvoudige XML documenten op een of andere manier in een tabellen structuur op te slaan, maar werkelijk round-trippen van complexe XML-structuren is daar niet bij.

Oracle 9i gaat een stapje verder met de introductie van een nieuw data type, te weten XMLType. Dit type is als binair of XMLType definieerbaar. Als XMLType wordt het XML document inclusief zijn typische XML-modelstructuur opgeslagen in de Oracle database, zodanig dat W3C standaards om elementen te benaderen, zoals XPath en XQuery mogelijk worden. De vraag is nog of alle andere eigenschappen van XML- structuren ook werkelijk worden ondersteund.

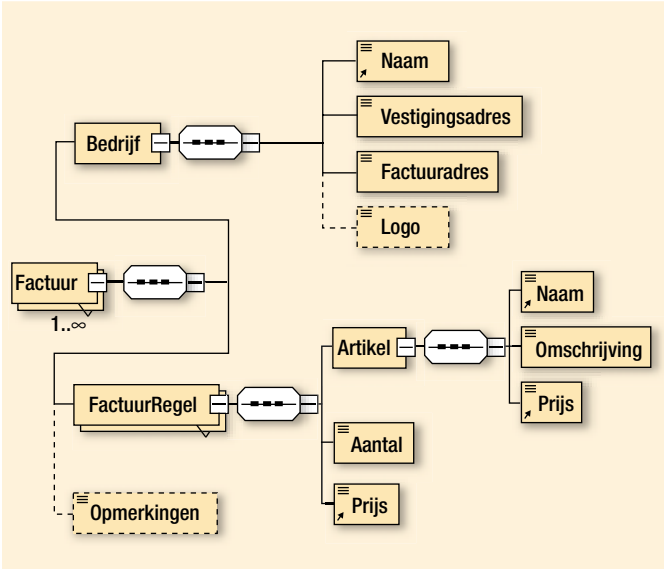
De native XML database

Om aan alle conversies en extra trucs een eind te maken, is het gebruik van databases waarin XML ook werkelijk als XML wordt opgeslagen en behandeld een betere aanpak. De term *native* XML is bedacht om de XML-database te onderscheiden van objectdatabases en relationele databases. Sommigen definiëren native XML-databases als databases waarin je XML-documenten kunt opslaan en opvragen volgens het bijbehorende XML-structuurmodel, onafhankelijk van de onderliggende fysieke manier van opslag. Over de precieze definitie wordt nog verschillend gedacht, maar native XML-databases moeten zeker in staat zijn te round-trippen en tot op elementniveau databasefuncties te ondersteunen. Populaire en goede XML-databases zoals X-Hive en Software AG's Tamino doen dat ook.
In tegenstelling tot relationele databases kennen native XML-databases zelf geen dwingend model. In principe kunnen er allerlei objecten in worden opgeslagen. XML-documenten echter worden herkend aan hun definitie als zijnde XML. Hierbij valideert de database de syntax bij inladen en opvragen. Structuurvalidaties worden meestal uitgevoerd door externe programma's, die rechtstreeks met de database communiceren.

Het beheren van native XML-databases is inmiddels zeer volwassen geworden. In de XML-projecten die bij AtosOrigin zijn uitgevoerd met Tamino wordt de DBA-functie uitgevoerd via de overzichtelijk op afstand te besturen webinterface. Hiermee worden de bekende databasefuncties als back-ups, roll-back, roll-forward, autorisaties, database- inrichting, efficiency van de opslag, automatische DB-expand en -shrink en dergelijke eenvoudig en betrouwbaar beheerd en uitgevoerd.

Performance

Misschien wel de meest gevoerde discussie tussen database-leveranciers gaat over de snelheid van hun product. Gebruikers echter zijn meer geïnteresseerd in de snelheid van hun toepassing. Meestal zit snelheid meer in de juiste opzet van de applicatie-architectuur en informatie- modellering. Toch zijn er verschillende algemeen toepasbare objectieve database-benchmarktests ontwikkeld waarin het uiteenrafelen van de XML- structuur en het weer in elkaar zetten ervan, ten behoeve van queries en updates, te meten is[4].
Bij elke aanpassing aan de data moet de integriteit van het hele document bewaakt worden. Het document moet uit de database worden gehaald en aan de update functie worden gegeven. Hiervoor worden meestal standaard API's gebruikt die het XML-document vanuit de database aanbieden aan de verwerkings-functie. Het meest toegepast is de DOM (Document Object Model) API (W3C-standaard) en plaatst het totale XML-document als boomstructuur in intern geheugen voor verwerking.
Een populaire methode bij programmeurs is SAX (Simple API for XML) die de XML als string met gebeurtenissen stukje bij beetje in geheugen inleest, waarbij XML-elementen als gebeurtenissen worden beschouwd waarmee de applicatie iets kan doen.
Het zal duidelijk zijn dat hele grote documenten tot geheugen problemen kunnen leiden bij de DOM-methode en dat SAX veel meer programmeerwerk kost om de context van XML-gebeurtenissen te onthouden en interpreteren. De DOM wordt door alle goede XML-databases volledig ondersteund. Voor performance zijn dus de omvang van het XML-document en de complexiteit van het structuurmodel de meest bepalende factoren.

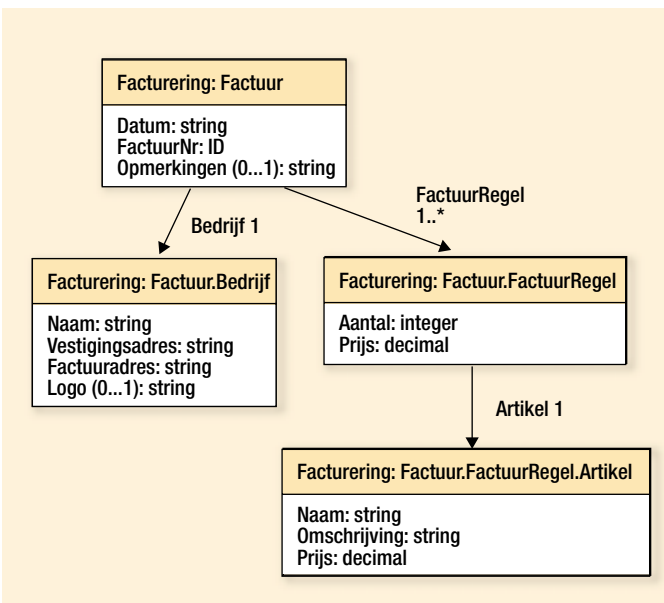


Afbeelding 8.

Normalisatie

Ervaring met XML-databases in projecten heeft geleerd dat de discussie over normalisatie van de database onverminderd uitdagend blijft. Gezien de inspirerende discussies in vorige nummers van DB/M denkt niet iedereen hetzelfde over normalisatie van relationele databases. Dezelfde discussie geldt voor XML-databases.

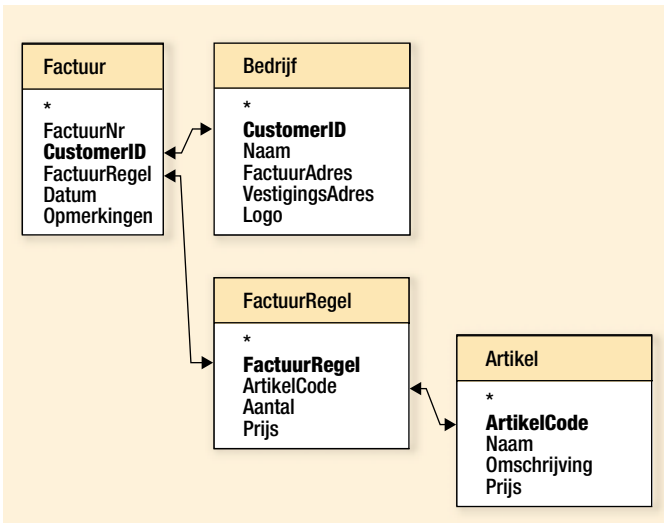
Natuurlijk kan een hiërarchisch model genormaliseerd worden. In XML wordt daarbij veelvuldig gebruikgemaakt van ID's en IDref attributen, hiermee wordt verwezen naar een ander deel van de structuur en redundantie voorkomen. Maar lang niet altijd is deze integriteitsnorm ook een absoluut vereiste. Het gaat immers over documenten.



Afbeelding 9.

Neem bijvoorbeeld facturen in een administratief proces. In een XML-database wordt de gehele factuur als gestructureerd document bewaard (een gedeelte adresinformatie, een gedeelte factuurregels en eventueel wat opmerkingen) inclusief digitale handtekening voor de rechtsgeldigheid (overigens is deze rechtsgeldigheid nog maar gedeeltelijk in wetgeving verankerd). Let wel, zo'n document heeft nog niets te maken met enige vormgeving ten behoeve van papier of webpagina. Met behulp van XML Schema kan een factuur zijn gestructureerd zoals in Afbeelding 8 is aangegeven. 'Datum' en 'FactuurNr' zijn in dit model als *required attribuut* van element <Factuur> gedefinieerd.
Facturen met deze structuur zijn altijd uit een XML database op te zoeken. Hiertoe wordt een XPath/XQuery-expressie gemaakt voor alle elementen <Factuur> waarin bijvoorbeeld het element <Naam> van het element <Bedrijf> gelijk is aan de naam van het gezochte bedrijf.

In dit XML-factuurschema is geen 'mixed content' opgenomen. De omzetting naar een relationeel model verloopt recht toe recht aan via het UML class diagram (Afbeelding 9). Er is nu goed te zien dat de attributen van het complexe type element <Factuur> terug te vinden zijn als attributen van de Factuur object class. Dit model is automatisch om te zetten naar een relationeel model (Afbeelding 10).



Afbeelding 10.

Het factuurnummer wordt hier nu de key voor de database. Dit factuurnummer zal gerelateerd moeten zijn aan een bedrijfstabel die wordt gebruikt voor de definitie van de klant zoals bedrijfsnaam en adressen en aan een factuurregel-tabel, waarin de geleverde artikelen worden benoemd. Een SQL-expressie levert gevraagde factuurinformatie op.
Er is nu één groot verschil. In de relationele database ligt de data ten behoeve van de factuur gefragmenteerd in verschillende tabellen opgeslagen en zal elke keer bij opvragen opnieuw worden samengesteld. Zo'n type factuur heeft geen rechtsgeldigheid tot

het moment van uitprinten. Om dit probleem te omzeilen wordt meestal een 'document management systeem' ingezet waar de eenmaal aangemaakte factuur binair in de database wordt opgeslagen en een tabel met kenmerken (metadata) wordt bijgehouden. Dit is meestal een platte tabel waarin bij elke factuurnummer één record wordt bewaard. De document manager fungeert dan meer als een archief en kan zelf geen factuur aanmaken.
Wat gebeurt er nu als de klant zijn bedrijfsnaam verandert of als hij verhuist? Voor de native XML-database maakt dat niets uit, de structuur van de factuur blijft hetzelfde en de bedrijfsnaam is nog steeds de originele bedrijfsnaam. In nieuwe facturen staat gewoon de nieuwe bedrijfsnaam. In de genormaliseerde relationele database zal bij dergelijke veranderingen een nieuwe CustomerID moeten worden aangemaakt met de nieuwe bedrijfsnaam. Daardoor blijft de data van de oude factuur ook nog valide.
Er moet nu waarschijnlijk wel een nieuwe tabel worden aangemaakt om aan te geven dat beide CustomerID's dezelfde klant betreffen. Kortom, de situatie bepaalt of het efficiënter is om een genormaliseerd model toe te passen of om dat juist niet te doen, dat geldt voor elk type database.

Conclusies

Het voorgaande geeft vooral aan dat er een verschil bestaat in interpretatie van informatie. Er is sprake van data-centrisch en/of document-centrisch gedrag van informatie. Grofweg wordt de verdeling gemaakt dat data-centrische informatie transactionele systemen betreft en beheerd wordt in relationele databases, terwijl document-centrische informatie veel meer terug te vinden is in content management systemen. De meeste native XML-databases zijn dan ook, vanwege het document- centrische gedrag van de XML-informatie, terug te vinden in de content management wereld. Bij data-centrisch gedrag zijn model mappings van relationeel naar XML eenvoudig te maken en is er meestal geen noodzaak voor een ander, dan het al in gebruik zijnde, type database. Tenslotte is de conclusie dat native XML databases hun plek verdienen als het over document-centrische informatie gaat en dat het echt verspilde energie is om de rijkere XML-structuren naar relationele structuren om te zetten.

> De meeste native XML-databases zijn terug te vinden in de wereld van content management wereld

Tegelijkertijd rijst wel het besef dat XML meer is dan alleen een handig informatie-uitwisselingsformaat en wordt het steeds meer gezien en toegepast als basis voor applicatie-ontwikkeling, waar de semantische kwaliteiten van XML-structuren goed kunnen worden benut.
De data/document-tweedeling wordt daar dan ook gelogenstraft door het feit dat snelle applicaties voor transactionele verwerking van informatie volledig gebouwd kunnen worden met standaard XML-technologie en -producten. Hierdoor wordt de native XML-database veel natuurlijker dan een relationele database gekoppeld met de applicatie. Een goed voorbeeld zijn applicaties gebouwd met open source XML-software van het 'Cocoon' framework[5].
We zullen ongetwijfeld nog veel meer gaan horen van native XML- databases.

Alex Stroom (alex.stroom@atosorigin.com)is IT Consultant bij AtosOrigin, met XML als expertisegebied.

Noten

De oorspronkelijke grafische representaties van XML Schema's zijn gemaakt met XMLSpy (zie www.altova.com); de omzetting naar UML is uitgevoerd met hyperModel (zie www.ontogenics.com); de originele relationele schema's zijn afkomstig uit Microsoft Access.

1. zie www.xml.org
2. zie www.w3.org
3. zie www.rpbourret.com/xml
4. zie http:/dbs.uni-leipzig.de/en/projekte/XML/paper/XMach-1.html
5. zie http://cocoon.apache.org/