XML in databases

Symbiose of parasiet?

Klaas Brant

In het vroege voorjaar van 2003 hebben Rick van der Lans en de schrijver dezes een onderzoek gedaan naar XML en databases. Het doel was om eens te kijken wat de huidige (XML) databases te bieden hebben en eens te brainstormen over het nut van dit alles. In dit artikel wil ik graag een samenvatting geven van onze research, mijn conclusies en overpeinzingen over dit geheel.

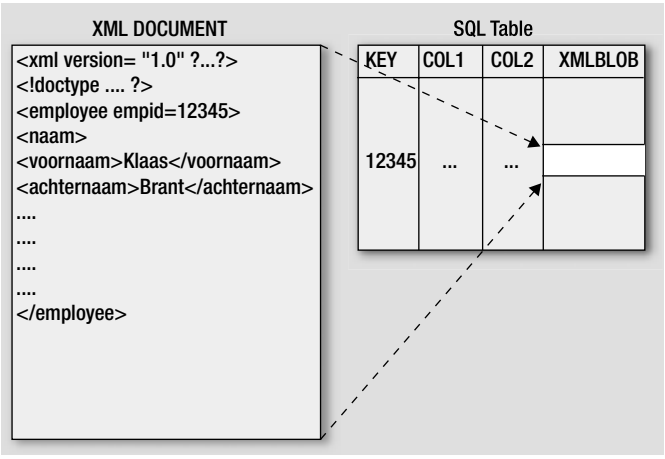
Voordat we de integratie tussen XML en databases verder kunnen onderzoeken moeten we eerst ons terrein een beetje afbakenen. Ik zal dit doen door een overzicht te geven van XML en technologieen die iets met XML te maken hebben. Daarna kijken we naar hoe we een en ander in onze database wereld kunnen gebruiken en tot slot kijken we hoe dit alles in onze huidige infrastructuur past.

Wat is XML?

Ik wil geen verhaal houden over waar XML nu eigenlijk precies vandaan komt, omdat dat niet belangrijk is. Het belangrijkste aspect van XML is dat het om data gaat; gestructureerde data om precies te zijn. Dat klinkt database-mensen als muziek in de oren, maar er zitten ook negatieve kanten aan. XML heeft van zichzelf een vrij open structuur en dat kan ons parten gaan spelen als we echt gestructureerd willen werken. Laten we XML eens onder de loep nemen. Een file met XML-data heet in principe een XML-document. Als je de file opent dan vallen je gelijk twee dingen op: de tags, die sterk aan HTML doen denken en het feit dat alles leesbaar is. De data is omsloten door tags die als het ware melden wat de informatie die zij insluiten moet voorstellen. De tags kunnen genest zijn en dat wil zeggen dat de informatie dus een soort hierarchie kent, zoals we die ook tegenkomen in complexe datatypes en record layouts. Maar elementen kunnen meerdere malen voorkomen en dat wil zeggen dat de structuur dus sterk doet denken aan een hierarchische database. Hiermee hebben we onmiddellijk een groot probleem te pakken: hierarchische data is niet 1:1 compatibel met een relationele manier van denken en opslag. In een relationele database worden de elementen uit elkaar getrokken tot individuele tabellen (normalisatie). Komt een element meerdere malen voor dan verschijnen er meerdere rijen in de tabel. Voor mensen die zich bezig houden met relationele databases is XML dus een hoeveelheid data die niet genormaliseerd is. Met andere woorden, voor hun is XML een "probleem". Neem het XML uit het volgende voorbeeld:

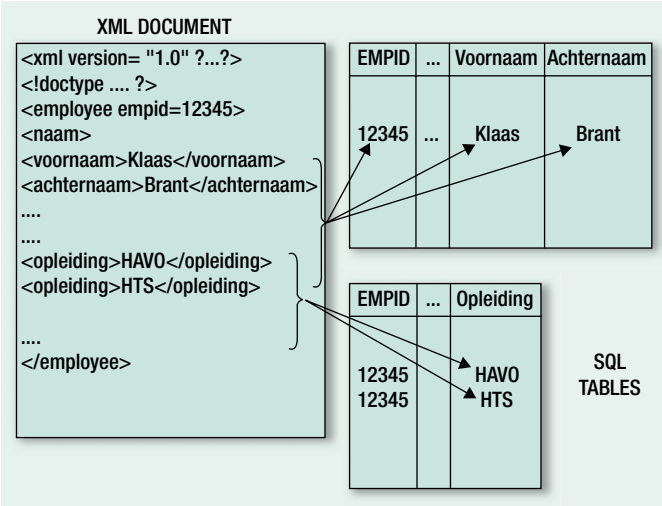
```
<?xml version=”1.0” standalone=no? encoding=”UTF-
8”?>
<?xml-stylesheet href=”employee.xsl”
type=”text/xsl’°?>
<!DOCTYPE employee SYSTEM “employee.dtd”>
<medewerker salarisnummer=”12345”>
    <naam>
          <voornaam>Klaas</voornaam>
          <achternaam>Brant</achternaam>
    </naam>
    <firma>KBCE b.v.</firma>
    <functie>Algemeen Directeur</functie>
```

Afbeelding 1 geeft een overzicht van opslaan. Hierbij slaan we het XML-document onveranderd op zodat we het ook weer 1:1



Afbeelding 1. Opslaan van XML-document.

kunnen reproduceren zoals we deze ontvangen hebben.
Dit kan belangrijk zijn. Omdat een XML-document erg groot kan worden zit er niks anders op dan de data op te slaan in een *blob*. Zoeken in de data wordt moeilijker maar niet onmogelijk (zie Databasetalen voor XML). Ook de omvang van de blob kan aardig uit de hand lopen. Tussen de tags kan veel zogenaamde *white space* zitten. White space is geen data maar zijn spaces tussen de tags. Veel tools geven de mogelijkheid om de white space te verwijderen maar dit kan soms niet gewenst zijn.
Het is bijvoorbeeld mogelijk om een XML van een CRC-achtige handtekening te voorzien. Een dergelijke handtekening wordt dan berekend over het gehele XML document inclusief de white space. Het weghalen van 'overbodige' white space heeft in dit geval tot gevolg dat de handtekening ongeldig wordt.



Afbeelding 2. XML-document afbreken.

Afbeelding 2 geeft een overzicht van het afbreken van XML. Dit is overigens een voorbeeld want er zijn vele varianten mogelijk. Afbreken heeft tot gevolg dat het XML-document uit elkaar gehaald wordt en via een soort mappingmechanisme in de database wordt opgeslagen. Problemen doen zich voor als er elementen in de XML zitten die niet voorkomen in de mapping. Hoe hiermee om te gaan? Afbreken of negeren? Als we alle data opslaan is het mogelijk om later de XML weer te reconstrueren maar de layout zal dan ongetwijfeld anders zijn (maar logisch identiek). De opslag van de data in de database verschilt tussen de diverse leveranciers. Zo gebruikt IBM simpele tabelstructuren en gebruikt Oracle veel meer geavanceerde complexe datatypes voor de opslag.
IBM heeft ook nog een hybride vorm waarbij data wel 1:1 wordt opgeslagen, maar tijdens de verwerking van het XML-document slaat een parse-proces ook de waarde van bepaalde elementen op. Deze worden dan gebruikt om een soort user index te bouwen om later snel het XML-document weer snel te vinden via SQL of SQLX. Zo zou het mogelijk zijn om de uit de data van ons voorbeeld de achternaam in de XML te vinden en deze te gebruiken om naar een blob te wijzen waar het gehele XML document in op geslagen wordt. Deze hybride methode kan erg gemakkelijk zijn bij de integratie van XML in legacy applicaties.

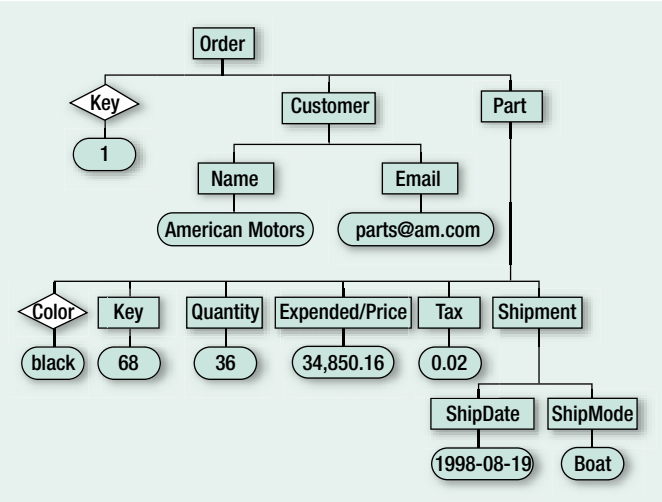
De native XML-database

Naast relationele databases bestaan er tegenwoordig ook databases die niet gebaseerd zijn op de relationele theorie maar XML zelf als vertrekpunt hebben. We noemen dit een native XML-database. Voorbeelden van dergelijke databases zijn Ipedo, Sonic, Tamino en X-Hive. In sommige gevallen kan het zeer aantrekkelijk zijn om met een dergelijke native XML-database te werken. Neem bijvoorbeeld een XML-document dat een product beschrijft en tijdens het productieproces wordt opgebouwd. Wellicht dat sommige processen wel of niet iets toevoegen, maar alle data met betrekking tot het product is compact bij elkaar. Daar waar men data al in XML-vorm heeft en dit ook zeer gewenst is, zoals bij een datastore van een content provider, is een XML-database natuurlijk een zegen. Voor bestaande legacy systemen is het toepassen van XML-databases niet zo gebruikelijk. Mensen die gewend zijn aan relationele databases vinden native XML-databases simpel qua structuur en mogelijkheden. Zo ontbreken bijvoorbeeld de *integrity rules* in dergelijke databases. Wellicht is het juist de openheid en simpele structuur die deze databases zo krachtig kan maken. Hoe ze performen bij gebruik van echt grote volumes is bij deze databases nooit aangetoond.

XML in programma's

Iedere database heeft zijn eigen interface. Als er uit een database een XML-document terugkomt (al dan niet via SQL of een XML-querytaal) dan moeten we hier in een programma op een flexibele manier mee om kunnen gaan. Op dit moment zijn er twee standaarden op gebied van XML interfacing: SAX (Simple API for XML) en DOM (Document Object Model). Ook hier hebben beide hun voor- en nadelen. Zoals de naam al aangeeft is SAX relatief simpel maar uitermate geschikt om XML-documenten van grote omvang te processen. Dit komt omdat SAX uitgaat van parsing, door het XML-document loopt en nooit het document geheel in storage heeft. SAX was een initiatief van de XML-DEV mailinglist.

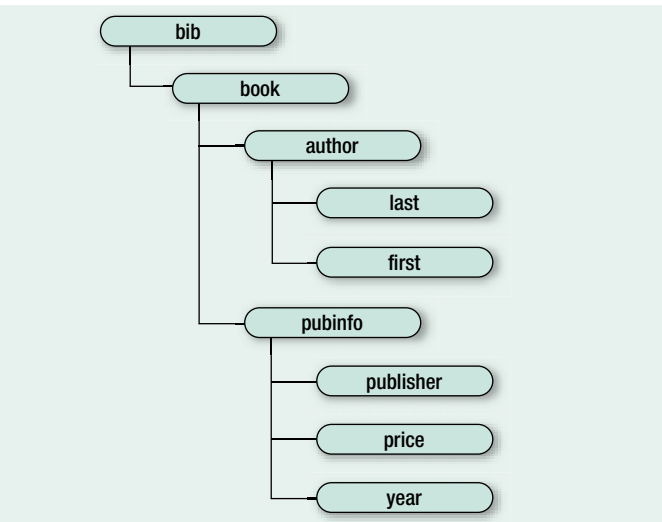
DOM is een initiatief van het W3C consortium. Het idee van DOM is dat een XML-document in storage wordt geladen en daarna bewerkt kan worden. Veel databases wijzen naar DOM als het gaat om de update van een XML-document. Het scenario is simpel: u doet de update zelf. De database geeft het XML-document als DOM-object en u gaat in uw programma naar hartelust dit XML-document aanpassen. Als u klaar bent, geeft u het document terug om weer geschreven te worden. Wat dat betreft doet het denken aan *record processing* uit de jaren zestig. Het zal duidelijk zijn dat we een databasetaal nodig hebben die direct in de database kan zoeken en updaten. Als de leveranciers niet staat zijn deze goed te implementeren dan zullen XML-databases altijd inferieur zijn aan relationele databases.



Afbeelding 3. XPath-expressie van een node.

Database-talen voor XML

Een goede vriend van mij zei altijd: "Het mooie van standaards is dat er zoveel zijn om uit te kiezen". Dit is in geval van XML helaas maar al te waar. De oudste van de XML-querytalen is XPath (nov 1999). Eigenlijk is XPath niet een volledige querytaal, maar een manier om een node (tak in de hiërarchie) aan te wijzen. In afbeelding 3 geeft de XPath-expressie */Order/Part/Shipment/Shipdate* een node aan. Door wildcards te gebruiken kunnen we in een XPath-expressie zelfs meerdere nodes aangeven. Dat XPath niet voldoende was en er een echte querytaal nodig is was snel duidelijk; al snel verscheen er een taal die XQL (XML Query Language) heette, een taal die sterk aan XPath deed denken. Maar ook een taal als XML-QL zag het levenslicht. Echter, in februari 2001 verscheen er een standaard, genaamd XQuery, die echte goede functionele aspecten bevat. Niet zo verwonderlijk, want een van de mensen die aan de wieg van deze taal stonden was Don Chamberlin, die wel eens *De Vader van SQL* wordt genoemd. XQuery is een zeer krachtige taal die reeds door vele leveranciers geadopteerd is. Tot de implementaties behoren o.a. IBM Information Integrator, Oracle 9i en natuurlijk de native XML-databases zoals Ipedo, Tamino, Sonic en X-Hive.



Afbeelding 4. XML-structuur uitgevers.

Dat XQuery iets totaal anders is dan SQL, blijkt wel uit de volgende query die uit de XML-structuur van Afbeelding 4 de uitgevers haalt die meer dan 100 boeken hebben uitgegeven:

```
FOR  $pub in distinct //publisher
LET  $bk :_ //book[pubinfo/publisher=$pub]
WHERE count($bk) >100
RETURN $pub
```

XQuery is ook in staat om een aantal relationele bewerkingen te doen zoals Join (inner en outer) en Sorting. Ook zijn er faciliteiten in XQuery voor Insert, Update en Delete (zonder DOM dus).
U begrijpt het, mocht u SQL onder de knie hebben dan kunt u met XML weer helemaal opnieuw beginnen. Naar mijn mening een reden waarom XML-databases het niet op korte termijn gaan maken. Ook de leveranciers begrijpen dit en een aantal leveranciers zijn begonnen met een XML-uitbreiding op SQL, het zogenaamde SQLX (zie www.sqlx.org). Tot de initiatiefnemers behoren de grotere databaseleveranciers zoals IBM, Microsoft, Oracle en Sybase. Inmiddels is dit SQLX tot ANSI-proposal gemaakt. SQL wordt op deze manier uitgebreid met XPathextensies. Op dit moment wordt alleen de uitbreiding aan het SELECT statement beschreven. Een voorbeeld van een dergelijk SQLX statement:

```
SELECT  ID, XMLELEMENT( NAME "Emp",
     XMLATTRIBUTE( id, name as "name")) AS "result"
FROM employee
WHERE
```

Het resultaat:

```
ID result
_____________________

1001 <Emp ID="1001" name="Smith"/>
1002 <Emp ID="1002" name="Martin"/>
```

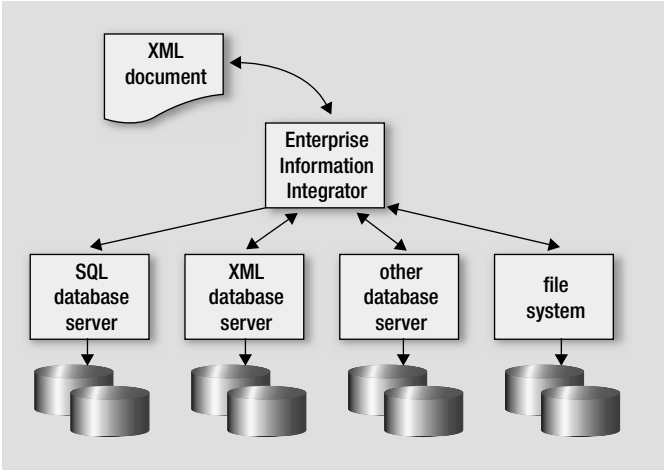
Deze uitbreiding aan SQL die als functies in SQL geïmplementeerd worden zijn veel eenvoudiger voor relationele mensen te leren. Met name voor de bestaande SQL-databases die XML-enabled worden is deze taal waarschijnlijk de toekomst. Native XML-databases hebben niets aan SQLX omdat ze de SQL niet ondersteunen.
Als laatste is er dan nog de taal Xupdate, die ontwikkeld is door het XML:DB Initiative. Deze taal beschrijft de update aan een XML door middel van een XML-achtige notatie. Als SQLX uitgebreid zal worden met INSERT, UPDATE en DELETE extenties is het nog maar de vraag of XUpdate nog toekomst heeft. Immers, XQuery en SQLX zouden dan beide aan de wensen van updates in XML voldoen, maar SQLX is SQL gericht.

XML en onze bestaande database

Na de studie van al deze technologieen kunnen we ons gaan afvragen hoe we XML kunnen inpassen in onze (bestaande) infrastructuur. Willen we überhaupt wel XML in onze database? Is het een symbiose of is XML een parasiet? Wat zijn de alternatieven? Laten we voorop stellen dat XML in eerste instantie bedoeld is als transportmiddel van informatie. Als XML alleen als statische informatie wordt gebruikt in een omgeving die verder niet XML-minded is, dan kunnen we waarschijnlijk maar beter zorgen dat we die informatie uit de XML halen die ons interesseert en verder de XML-structuur verlaten. Opslaan in de database zou onzinnig zijn. Dit kan door parsing als we de database binnenkomen, of zelfs op een nog hoger nivo zoals in de middleware. Zien we XML echter als een compleet, wellicht rechtsgeldig, document zoals een order, dan kan het prettig zijn om het XML-document op te slaan in onze database. Je kan het XML-document in zijn geheel dan beschouwen als een stuk (complexe) informatie. Dat je later in deze XML-documenten nog kan zoeken is natuurlijk geweldig. Sommige databases geven zelfs bij de opslagmethode de mogelijkheid tot het aanleggen van (user) indexes.
Is het XML-document erg dynamisch en/of wordt het gebruikt in een typische XML-omgeving (zoals een website die content-driven is) dan zijn native XMLdatabases op dit moment duidelijk superieur aan de XML-enabled relationele databases.

Het alternatief

Zijn er alternatieven? Jawel, een nieuwe trend is een Enterprise Information Integrator (EII). Deze tools zijn in staat om met een groot aantal heterogeneous database servers om te gaan.
In afbeelding 5 is te zien hoe een dergelijk tool in staat is om diverse informatiebronnen te ondervragen en het resultaat aan de buitenwereld af te geven als een XMLdocument. Maar ook omgekeerd kan een XML-document worden afgebroken en de informatie worden opgeslagen in de diverse database-servers.
De meeste EIItools gebruiken XQuery als querytaal, maar de onderliggende databaseservers hoeven helemaal geen XML te ondersteunen. De EII-tools doen de vertalingen. Sommige EII-tools kunnen ook SQL aan als querytaal. Informatie uit diverse databronnen kan door het EII-tool middels een join-operatie met elkaar gecombineerd worden. Queries die binnenkomen worden door het EII geanalyseerd en afgebroken tot kleinere queries die dan worden doorgegeven aan de onderliggende database-servers. Dit maakt deze EII-tools tot bijzonder krachtige query- en update-tools die in staat zijn met iedere vorm van informatie om te gaan. Van Active Directory van Windows/NT, SQLdatabases en file systems tot XML-databases. Door het tool XML met de buitenwereld te laten communiceren is helemaal universeel inzetbaar. Mocht u denken dat dit alles toekomstmuziek is dan vergist u zich: er zijn nu reeds een aantal EII-tools te koop zoals: Enosys Server, IBM Information Integrator, Ipedo XML Information Hub, MetMatrix Server en Nimble Integration Suite. Sommige hiervan zijn reeds bijzonder krachtig.



Afbeelding 5. De Enterprise Information Integrator.

Zijn er nog meer alternatieven? Natuurlijk, we kunnen zelf aan de slag met Java en NET class libraries die ons in staat stellen om met XML en relationele informatie om te gaan. Ook bieden deze class libraries services om XML te transformeren en parsen. Is dit de juiste aanpak? Dat valt moeilijk te zeggen. De ontwikkelingen gaan nu zo snel dat oplossingen binnen 5 jaar verschrikkelijk ouderwets kunnen worden. Langs de andere kant kan een `home made solution' goed aansluiten bij de wensen van ons bedrijf en veel geld besparen. Daar waar XML alleen in de interface gebruikt wordt, kan zo'n home made solution perfect zijn.

Conclusie

Tijdens ons onderzoek hebben we naar veel databases gekeken en ons soms verbaasd (in positieve en negatieve zin) over de mogelijkheden. Dat XML geen hype is maar een standaard-manier om informatie weer te geven, is inmiddels duidelijk. Dat ook uw organisatie iets met XML van doen krijgt is vrij zeker. Wellicht was het niet uw idee maar wel dat van een van de mensen met wie u zaken doet. Ook voor de database-leveranciers is het duidelijk dat een goede XML-ondersteuning een ware must is. Voor de relationele databases is het een uitdaging om XML goed en snel te ondersteunen en voor de native XML-databaseleveranciers is het een uitdaging om de wereld te overtuigen dat zij de juiste oplossing hebben voor al uw XML-wensen.
Wat is wijs? Zoals zo vaak is het belangrijk om ons niet direct te verlagen naar oplossingen, maar eerst het probleem eens goed onder de loep te nemen. Waarom heb ik XML en wat wil ik ermee? Probeer eens uw ICT-architectuur te schetsen zoals die nu is en hoe die moet worden, hoe u denkt dat deze er over 5 jaar uit ziet. wees daarbij realistisch. Denk bijvoorbeeld aan fusies en markttrends. Alleen op die manier kunt u een gefundeerde keuze maken waar en hoe u XML gaat inzetten. Als het architectuurplaatje klopt dan kunnen XML en databases een perfecte symbiose vormen.

Klaas Brant (kbrant@kbce.nl) is DB2-specialist en directeur van KBCE b.v.