ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 5 - NUMMER 1, SEPTEMBER 2007

OPEN CURSOR

In een snel ontwikkelend informatica-landschap lijken relationele database-systemen, en in het bijzonder ons aller DB2, een rots in de branding: robuust, performant, betrouwbaar, beschikbaar.
Toch ontkomt ook DB2 niet aan de belangrijke evoluties wat betreft datarepresentatie en datatoegang.
We hadden het in een vorig nummer al over Unicode.
In dit nummer bekijken we twee andere uitdagingen voor DB2: hoe omgaan met 'dimensionaal modelleren', en wat met hiërarchische datarepresentaties, i.h.b. XML?

Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- Een kijk achter de schermen van het 'waarom' en het 'hoe' van data warehousing, meer in het bijzonder de manier waarop met materialised query tables (MQTs), ook wel materialised views (MVs) genoemd, omgegaan kan worden.
- Een eerste deel in de nieuwe reeks *DB2 versie 9* waarin we kennismaken met '*pureXML'*, DB2's poging om relationele en hiërarchische data te verzoenen en via een geïntegreerde interface toegankelijk te maken.
- Cursusplanning najaar 2007.

1

CLOSE CURSOR

In het volgende nummer gaan we dieper in op XML, met daarin onder andere een korte tutorial over XPath. We nemen dan ook een andere invalshoek voor data-representatie onder de loep: het object-georiënteerde model, en de relatie tot DB2.

Tot dan!

Over MQTs & MVs - waar het echt om te doen is!

Kris Van Thillo (ABIS)

Als je een cursus mag geven getiteld 'Data warehouse concepten', is één van de eerste topics die je aankaart de definitie van data warehouses als door Bill Inmon uitgewerkt. In deze definitie somt hij een aantal eigenschappen op waaraan data warehouse 'implementaties' moeten voldoen. Eén van deze eigenschappen vat hij samen met de term *'separate'*. Immers, door de karakteristieken eigen aan een data warehouse wat betreft toegang, beschikbaarheid, beveiliging, omvang van de data, ..., kan men nooit een data warehouse en een OLTP/Batch 'operational' store als één database-omgeving implementeren. Onafhankelijke hardware en software omgevingen zijn vereist om beide systemen te implementeren; een ETT/ETL proces zorgt voor de integratie en interactie tussen beide.

Ik kan het vaak niet nalaten om over deze karakteristiek 'separate' publiek verder na te denken. Is dit nog steeds zo? Is het niet denkbaar dat op (korte) termijn beide omgevingen als één geïntegreerde omgeving zullen worden geïmplementeerd? En dat de data vervolgens slechts éénmaal zal worden opgeslagen - gebruiks'neutraal'? En dat allerlei database specifieke technieken zullen gebruikt worden om deze data dan wel in OLTP, batch, OLAP, DSS omgevingen efficiënt te kunnen benaderen?

Traditionele B-tree indexen, geïmplementeerd in de meeste RDBMS-sen (Oracle, DB2, SQL Server) zijn hier reeds een voorbeeld van. En natuurlijk ook MQTs en MVs. Materialised Query Tables (MQTs) en Materialised views (MVs) zijn in respectievelijk DB2- en Oracle-omgevingen inderdaad een poging om een optimalisatie richting OLAP en DSS door te voeren. In vorige Exploring DB2s werd reeds gesproken over MQTs. In dit artikel willen we even stilstaan bij hoe een query optimalisatie in de context van MQTs en MVs kan bijdragen tot toegenomen applicatie-efficiëntie.

We staan ook even stil bij een aantal relevante verschillen tussen DB2 for z/OS, DB2 for LUW, en Oracle.

Even vooraf - Over sterren en sneeuwvlokjes....

Dimensionaal modelleren is een kunst op zich – een grondige discussie hierover zou ons in deze context te ver leiden. Twee relevante termen moeten echter worden toegelicht: de 'ster' en 'het sneeuwvlokje'. Ze pogen immers aan te geven hoe een specifiek dimensioneel model gestructureerd is.

- een stermodel bevat één omvangrijke centrale tabel met detail data - de *fact table* - met daarrond een aantal *dimensietabellen*. Dit zijn typisch kleinere tabellen die een verklaring, een context, bieden

aan de detailgegevens. Vaak wordt per dimensie één tabel weerhouden. Typische voorbeelden: tijd, lokatie, product, ... 'Fact' en 'dimensie' worden typisch gelinkt via PK/FKs. Het resultaat is visueel best te beschrijven als een 'ster'.

• de verschillende dimensietabellen als boven aangehaald bevatten meestal 'repeating values'. Denk bijvoorbeeld aan een dimensie 'verkoopspunt'. We registreren per punt de stad, provincie, en het land waar de verkoop heeft plaatsgevonden in één dimensie - dus één tabel. Gevolg: repeating values, vermits Europa uit verschillende landen bestaat; en een land uit verschillende steden. Doet men dit echter niet - of met andere woorden, normaliseert men de dimensietabellen waardoor meerdere tabellen per dimensie ontstaan - dan ontstaat het sneeuwvlokmodel.

Beide modellen komen in de praktijk vaak voor, hebben voor en nadelen, en worden al dan niet door bepaalde RDBMSsen ondersteund. Op dit laatste wordt later nog even stilgestaan.

Wat zijn MQTs (of MVs) ook al weer?

Een MQT (of MV) is strikt genomen een tabel, die afgeleide data bevat. Afgeleid, in de zin dat het typisch om geaggregeerde data gaat, waarvan de brondata reeds in één of een aantal tabellen aanwezig is. Alhoewel de definitie van een MQT sterk lijkt op deze van een view (vandaar de Oracle-naam – Materialised View), is er toch één belangrijk verschil: MQTs zijn gematerialiseerd, d.w.z. dat ze fysiek ruimte in beslag nemen. En ze hebben ook iets van een index, omdat het gebruik van MQTs transparant is voor de gebruiker. Inderdaad, DB2 en/of Oracle bepalen zelf wanneer een MQT/MV wordt gebruikt.

Bij het aanmaken van deze objecten moet bij een aantal zaken worden stilgestaan – zie ook een vorig nummer van Exploring DB2:

- hoe (fysieke karakteristieken) een MQT/MV op te bouwen;
- hoe en op welke wijze een MQT/MV te refreshen: automatisch dan wel na DBA-interventie;
- wanneer een MQT/MV te refreshen, bijvoorbeeld rekening houdende met transacties;
- wanneer MQT/MVs initieel moeten worden opgebouwd;
- of een MQT/MV kan worden gebruikt bij query optimisatie als mogelijke alternatief toegangspad.

DB2 for z/OS ondersteunt een bepaald aantal van de boven opgenomen opties; DB2 for LUW en Oracle zijn in deze relatief evenwaardig; spreekt voor zich dat ze beide een andere, specifieke syntax hanteren!

MQTs, MVs en query optimisatie

Gebruikers die queries schrijven hoeven niet op de hoogte te zijn van het bestaan van (traditionele) indexen: de query optimizer heeft juist als doel uit te maken of het gebruik van deze indexen nuttig is of niet. Aldus wordt een belangrijke vorm van transparantie gecreëerd.

Eigenlijk moet een query optimizer op dezelfde wijze ook beslissen of MQTs/MVs beschikbaar zijn om als toegangspad te worden weerhouden.

En dus volgt een logische vraag: waar houden query optimizers typisch rekening mee op het moment dat bij optimalisatie access paden moeten worden bepaald? Een kort overzicht van een aantal relevante elementen.

Statistieken. Cost-based optimizers blijven natuurlijk in eerste instantie cost-driven; het bestaan van statistieken laat hen toe te berekenen of het betrekken van MQTs/MVs in een mogelijk toeganspad zinvol is. Berekenen van statistieken op MQTs en MVs is noodzakelijk voor een realistische inschatting van het nut van het gebruik van MQTs en MVs.

Inhoud en Kwaliteit. Komt de inhoud van de MQT/MV nog steeds overeen met de inhoud van de brondata waar deze op gestoeld is? Of met andere woorden: wanneer heeft de laatste refresh plaatsgevonden? Het is niet ondenkbaar dat de MQT/MV data ‘stale’ wordt na een bepaalde tijd; en dat er dus de voorkeur wordt aan gegeven de ‘echte’ brondata voor een bepaalde query te gebruiken. Met mogelijk, alle performance gevolgen vandien. Of omgekeerd, dat men juist een beperkte afwijking zal aanvaarden. E.e.a. wordt vaak gekoppeld aan het concept publicatiedatum, eigen aan een datawarehouse omgeving: voor publicatie (voor refresh) is de data ‘stale’; na publicatie (na refresh) is de data 'gesynchroniseerd'. De gebruiker bepaalt wat hij of zij nodig heeft: in DB2 typisch via bijvoorbeeld special registers; in Oracle bijvoorbeeld op basis van hints.

Metadata informatie. Niet de minst belangrijke. Want hoe vertaal je een query op tabel 'A' naar een query op tabel 'X' met oog voor - lees: behoud van - semantiek, integriteit, consistentie? Waarbij tabel 'X' dan eigenlijk de MQT/MV voorstelt. De database cataloog moet hiertoe worden aangewend; vraag is over welke informatie deze moet beschikken om dergelijke vertaalslag mogelijk te maken. Enter ‘Foreign Keys’; enter ‘Dimensions’.

Over Foreign Keys (FKs) (DB2 en Oracle)

Vaak wordt het definiëren van constraints (FKs) in een datawarehouse beschouwd als een soort van heiligschennis. De bedoeling van FKs is toe te laten de integriteit van de data na te kijken (te garanderen). Laat ons het makkelijkheidshalve houden op een “interne kwaliteitscontrole door het RDBMS uitgevoerd”. Dergelijke FK-definities zijn niet wenselijk in een datawarehouse, stelt men, omdat kwaliteitscontrole juist de taak is van het ETT/ETL proces; en dus, dat de data die zal worden geladen in het data warehouse, automatisch aan de hoogste kwaliteitsnormen voldoet. FKs definiëren zou enkel 'dubbel' werk zijn; resources verspillen die alternatief kunnen worden aangewend.

En toch is voor FKs een belangrijke taak weggelegd: het assisteren van de query optimizer in het evalueren van alternatieve access-paden op basis van MQTs. Want een relatie tussen ‘master’ en ‘detail’

laat de optimizer perfect toe een typische ‘roll-up’ dan wel ‘drill-down’ bewerking te evalueren op basis van bestaande MQTs/MVs. Wat de resource-impact betreft is tegenwoordig e.e.a. makkelijk opgelost:

- in DB2 definiëren we de FK als 'NOT ENFORCED' en 'ENABLE QUERY OPTIMIZATION'
- in Oracle definiëren we de FK als 'NOVALIDATE' en 'RELY'

Als we dit even relateren aan het boven aangehaald ster- of sneeuwvlok-model, is het duidelijk dat de creatie van verschillende PK-FK-relaties op basis waarvan de optimizer zal moeten kiezen een MQT/MV te gebruiken, eerder relevant zal zijn bij sneeuwvlokmodellen!

Over Dimensions (Oracle)

Een dimensie-object kan worden beschouwd als een hiërarchische beschrijving van de relatie tussen kolommen van één of meerdere tabellen (in dit laatste geval spreekt men van join-based dimensies). Op basis van deze dimensies is o.a. de optimizer in staat het nut van het gebruik van MVs in te schatten. Voorbeelden van dimensies:

- de traditionele relatie tussen fact- en dimensietabellen
- relaties die bestaan tussen kolommen binnen één of meerdere tabellen om bijvoorbeeld relaties tussen repeating values te duiden;
- functionele afhankelijkheden die bestaan tussen kolommen van één of een aantal tabellen

Een voorbeeld van een dimensie is in wat volgt opgenomen. De naam van de dimensie kan vrij worden gekozen, alsook de naam van de verschillende aggregatie ‘levels’ (lkey, lmonth, lquarter, ...). Elk level mapt typisch met één kolom uit één of meerdere tabellen; de notatie is tabel.kolom. In bijgevoegd voorbeeld is dus 'time' de tabel, en 'tkey', 'month'... steeds een kolom uit overeenkomende tabel.

```
create dimension time
level lkey is time.tkey
level lmonth is time.month
level lquarter is time.quarter
level lyear is time.year
level lweek is time.week
hierarchy julian_rollup (
tkey child of month child of year)
hierarchy fiscal_rollup (
tkey child of week child of quarter);
```

Dimensie-objecten hebben als doel de informatie van typische PK-FK-relaties uit te breiden door in eerste instantie hiervan onafhankelijk te zijn! Want was ‘separate’ niet wat we in vraag stellen? PK/FKs zijn relevant voor OLTP/Batch omgevingen, waarbij kwaliteitsgarantie door het RDBMS moet worden voorzien. Dimensie-objecten bieden de mogelijkheid dezelfde data te structureren op een meer dimensionele wijze! En dus relevant te zijn voor datawarehouse omgevingen.

Het dimensie-object wordt op dit moment door de DB2-familie nog niet ondersteund. De beschikbaarheid van dit object in Oracle duidt op het belang dat Oracle hecht aan een correcte optimisatie van sterschema's. DB2 tracht dit op te vangen door bij de creatie van MQTs te wijzen op het belang van het toevoegen van redundante info b.v. functioneel afhankelijke kolommen, count(*) waar ook sum() aanwezig is, enz.

Typische voorbeelden

Laat ons tot slot even een aantal typische voorbeelden van query rewrite aanhalen. Concrete implementatie en realisatie is van een aantal factoren afhankelijk: gebruikte RDBMS, definitie van FKs en/of dimensies, definitie van MQTs/MVs.

SQL text match. Conceptueel een eenvoudige situatie. De tekst van de end-user query komt overeen met de tekst van de MQT/MV-definitie. Traditionele trace tools - b.v. Explain plan (DB2, Oracle) of Autotrace (Oracle) - geven aan dat een MQT/MV werd gebruikt.

Roll Up. We beschikken over verkoopsdata per dag. We hebben een MQT/MV per product per maand. En een gebruiker stelt een vraag op de originele fact-tabel naar verkopen per product per jaar! Een *group by* van de MQT/MV naar jaarniveau moet de antwoorden geven waar we naar op zoek zijn.

Data-afhankelijkheid. Een MQT/MV bevat een product-ID en een som van verkopen. Een gebruiker stelt een vraag naar een productnaam en een totale verkoop. Het moet voor de database mogelijk zijn een join te maken met een afhankelijke tabel om het product-ID van de MQT/MV te vertalen naar een productnaam; de fact wordt dus niet gebruikt!

Bijkomende berekeningen. Een MQT/MV bevat totalen per product; productgroup is ook opgenomen. Gemiddelden per product worden gevraagd door een eindgebruiker. Een bijkomende berekening op de MQT/MV kan volstaan om de vraag van de gebruiker, afhankelijk van het aantal producten per productgroup, te beantwoorden.

Bedenkingen tot slot

Wat is dan de mening van het publiek? Blijft een data warehouse een 'separate' database? Waarschijnlijk niet. De dogmatische normalisatie-eis eigen aan OLTP databases wordt verder opgegeven; en het RDBMS in gebruik zal steeds meer en steeds vaker technieken bieden die centraal opgeslagen data naar verschillende omgevingen toe kan openen. Een van deze - niet eenvoudige - technieken is de MQT of MV. Belangrijk om verder onderzoek naar te verrichten: als mogelijkheid om queries verder te optimizeren; om data applicatie-onafhankelijk op te slaan; en dus, om de kosten en de uitdagingen eigen aan het ETT/ETL onder controle te krijgen!

XML in DB2 v9

Peter Vanroose (ABIS)

Dit is het eerste artikel in een reeks waarin we recent toegevoegde XML-functionaliteit van DB2 zullen exploreren.

XML is meer en meer de standaard aan het worden voor het uitwisselen van informatie tussen bedrijven, tussen applicaties, tussen webservices, ... (o.a. in het kader van 'service-oriented architectures' of SOAs). Database-applicaties moeten bijgevolg steeds vaker gegevens wegschrijven in XML-formaat, en/of krijgen gegevens aangeleverd in dit formaat. Hiervoor bestaat ondertussen software die ons het leven eenvoudiger maakt bij het lezen of schrijven van XML (zoals Java-klasses of COBOL-extensies).

Toch zouden we liever, indien mogelijk, het converteren van/naar XML vermijden door data 'als XML' op te slaan. In een RDBMS wordt data echter steeds relationeel opgeslagen, terwijl XML van nature een hiërarchische en sequentiële datarepresentatie is, twee concepten die 'taboe' zijn in een RDBMS. Data converteren (tussen RDBMS en XML-document) impliceert dus bijna altijd ook: het datamodel transformeren.

Vandaar de vraag naar intrinsieke XML-ondersteuning binnen DB2. Inclusief een transparante manier om de opgeslagen XML-data te ondervragen, hetzij met standaard SQL, hetzij met XQuery, de standaard-ondervragingstaal voor XML-documenten.

In vorige versies van DB2 was er weliswaar een 'XML extender', maar sinds versie 9 is XML-ondersteuning volledig geïntegreerd in DB2 als alternatieve datarepresentatie, onder de IBM-naam 'pureXML'. DB2 als hybride database dus, met zowel een relationele als een XML-gebaseerde interface, met daarachter zowel relationele als hiërarchische data, beide efficiënt opgeslagen (met indexering waar gewenst).

Dat is althans wat IBM ons wil doen geloven. Vandaar deze exploratie, om een beter beeld te krijgen van de draagwijdte van deze nieuwe mogelijkheden.

'pureXML' is beschikbaar in DB2 v9 op alle platformen. Voor wie nog geen toegang heeft tot de recent verschenen DB2 v9 for z/OS, is er dus ook DB2 v9 for LUW, trouwens gratis te downloaden voor MS-Windows. Volgens de handleidingen zijn er overigens zeer weinig verschillen tussen de twee implementaties. Waar van toepassing zullen we in deze artikelenreeks de verschillen vermelden.

In deze bijdrage geven we een eerste impressie van wat XML in DB2 kan betekenen, met enkele eenvoudige voorbeelden. In volgende bijdragen wordt dan dieper ingegaan op enkele belangrijke XML-concepten, zoals de boomrepresentatie, schema's, XPath, en XQuery. Telkens met de manier waarop DB2 deze concepten aanbiedt en implementeert.

Een eenvoudig voorbeeld

Stel dat we in DB2 de data bijhouden voor een telefoonboektoepassing: gegevens 'naam' en 'telnr'. Genormaliseerd worden dit twee tabellen: een tabel 'personen' met een PK-kolom 'id' (een identitykolom) en een kolom 'naam', en een tabel 'nummers' met een FK-kolom 'pid' en een kolom 'telnr'. Personen met meerdere telefoonnummers hebben uiteraard meerdere rijen (met zelfde pid) in tabel 'nummers'.

```
CREATE TABLE personen
( id INT NOT NULL GENERATED ALWAYS PRIMARY KEY,
  naam VARCHAR(256) NOT NULL );

CREATE TABLE nummers
( pid INT NOT NULL,
  telnr VARCHAR(256) NOT NULL,
  FOREIGN KEY (pid) REFERENCES personen ON DELETE CASCADE );
```

De tabellen zien er b.v. als volgt uit:

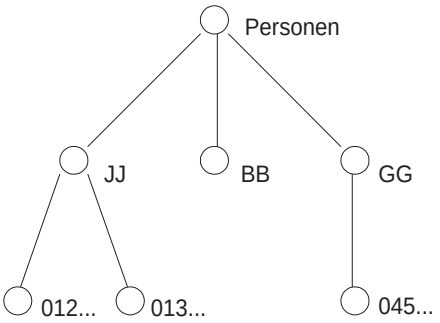
```
   personen                          nummers
 id      naam                      pid    telnr
  1      Jan Janssens                1    0123456789
  2      Bertha Bertels              1    0132456789
  3      Geert Geeraerts             3    0456789123
```

Dezelfde gegevens worden in XML hiërarchisch opgeslagen, als een lijst van namen (in een bepaalde volgorde dus), elk met een lijst (mogelijk leeg) van telefoonnummers:

```
<personen>
  <persoon id="1">
    <naam>Jan Janssens</naam>
    <nummers>
      <telnr>0123456789</telnr>
      <telnr>0132456789</telnr>
    </nummers>
  </persoon>
  <persoon id="2">
    <naam>Bertha Bertels</naam>
    <nummers/>
  </persoon>
  <persoon id="3">
    <naam>Geert Geeraerts</naam>
    <nummers><telnr>0456789123</telnr></nummers>
  </persoon>
</personen>
```

Hierin herkennen we de 'markup'-tag-notatie zoals <naam> die ook in HTML voorkomt; alleen is XML een 'eXtensible Markup Language': we kunnen zelf tags (zoals <naam>) definiëren. Indentatie is trouwens niet relevant maar helpt hier wel de hiërarchie te zien.

Stel dat dit XML-bestand binnenkomt in een applicatie, die de gegevens relationeel moet opslaan. Hiervoor moet de data allereest *geparsed* worden, d.w.z. dat de XML-tekst geïnterpreteerd wordt en intern in een boomstructuur wordt voorgesteld:



Invoer die niet ‘well-formed’ is, die dus geen syntactisch correcte XML is, wordt daarbij dadelijk overboord gegooid. In tweede instantie wordt geverifieerd of deze data geldig is, d.w.z. de data wordt ‘gevalideerd’ m.b.t. een structuurvoorschrift: DTD of Schema. Tenslotte wordt de data vertaald naar z'n relationele representatie (’shredding’). Uiteraard kan de applicatie ook gevraagd worden, een XML-output te produceren als gevolg van een database-query.

Zo zal het antwoord op de vraag

```
SELECT naam, telnr
FROM personen INNER JOIN nummers ON id = pid
WHERE telnr LIKE '01%'
ORDER BY id
```

daarna door de applicatie moeten vertaald worden naar XML, b.v. als

```
<personen>
  <persoon id="1">
    <naam>Jan Janssens</naam>
    <nummers>
      <telnr>0123456789</telnr>
      <telnr>0132456789</telnr>
    </nummers>
  </persoon>
</personen>
```

Deze dubbele conversie – van en naar XML – willen we (indien mogelijk) vermijden.

Een ‘native XML’-oplossing in DB2

Indien de enige toegang naar onze database zou bestaan uit XML, zowel voor invoer als voor uitvoer, dan is de omslachtige conversie naar/van relationele representatie overbodige overhead. DB2 v9 laat inderdaad toe, enerzijds de invoer –zonder conversie– als XML op te slaan binnen DB2, en anderzijds deze XML te ondervragen (m.b.v. SQL dan wel XQuery) met als resultaat b.v. de XML-output hierboven.

XML-objecten worden weliswaar nog steeds in tabellen opgeslagen, maar in het extreme geval kan één enkel XML-object in een tabel met één rij en één kolom gestopt worden. Een dergelijke tabel kan b.v. als volgt aangemaakt (en gevuld) worden:

```
CREATE TABLE my_xml_data ( x XML );
INSERT INTO my_xml_data (x) VALUES (
        '<personen><persoon><naam>Tessa Tessens</naam></persoon></personen>');
```

DB2 v9 gebruikt dus het nieuwe datatype XML om XML-objecten op te slaan. Dit datatype laat –waar zinvol– conversies van/naar VARCHAR toe, maar is meer dan zomaar een tekstuele opslag van XML. Zo zal bij het INSERT-statement de gegeven XML-input eerst geparsed worden en –indien well-formed– in compacte vorm opgeslagen worden.

Dit houdt geen validatie in: daarvoor moeten we iets omslachtiger te werk gaan; zie hiervoor een latere bijdrage. We kunnen dus (voorlopig) niet afdwingen dat er verplicht een tag <nummers> moet aanwezig zijn binnen elke <persoon>, laat staan dat er een tag <personen> zichtbaar is op het buitenste niveau. Wel krijgen we een negatieve SQLcode bij ongeldige ('not well-formed') XML:

```
INSERT INTO my_xml_data (x) VALUES (
         '<personen><persoon><naam>Tessa Tessens</naam></persoon>' );
```

produceert geen goedgevormd XML-document omdat de 'tag' personen niet gesloten wordt; SQLcode: -16203, "XML document input ended before all started tags were ended". (Er zijn nog enkele andere XML-fouten in de -16000-reeks.) Bijgevolg wordt er in dit geval geen rij toegevoegd aan de tabel. Alle 'XML'-velden in een kolom van datatype XML zijn gegarandeerd geldige ('well-formed') XML.

XQuery en XPath in SQL-expressies

De omweg via relationele representatie enerzijds, en een SQL-query anderzijds, kan vermeden worden (indien gewenst) door gebruik te maken van *XQuery*, de standaard manier om XML-documenten te ondervragen. XQuery ondervraagt een boomstructuur, geen tabel. We hebben dus een syntax nodig die toelaat om delen van een boomstructuur aan te duiden, zowel qua structuur als qua inhoud. De taal die dit toelaat, is *XPath*.

De vraag van daarnet, "geef naam en telefoonnummer(s) van iedereen met een telefoonnummer dat met '01' begint", betekent dat we enerzijds de boom enkel willen doorzoeken binnen alle <persoon>-entries en hun <telnr>-subentries enerzijds, en anderzijds deze deelboom (die hier toevallig de volledige boom is) willen beperken tot enkel die takken die aan een bepaalde conditie voldoen.

De XPath-expressie hiervoor is:

```
/personen/persoon[starts-with(nummers/telnr,'01')]
```

In DB2 v9 kunnen we de scalaire functie 'xmlquery' gebruiken om een XPath-expressie te laten evalueren:

```
SELECT xmlquery( '$XD/personen/persoon[starts-with(nummers/telnr,"01")]'
                 passing my_xml_data.x as XD )
FROM   my_xml_data
```

Merk op dat de '$'-notatie ('$XD') de XQuery-wijze is om naar een volledig document te verwijzen; hier moet DB2 even tussenkomen om te vertellen dat '$XD' (case-sensitive!) te vinden is in kolom x; vandaar de 'passing ... as ...'.

XQuery als alternatief voor SQL

In bovenstaande voorbeelden is de gedachtengang eigenlijk nog steeds relationeel: gebruik een SQL select-statement en pas een scalaire functie toe op de XML-velden.

DB2 v9 laat echter ook een *XML-centrische* aanpak toe: het centrale object is niet een tabel maar een XML-document, en de querytaal is niet langer SQL maar XQuery:

```
xquery db2-fn:xmlcolumn('MY_XML_DATA.X')/personen/persoon[naam="Jan Janssens"]

xquery for $y in db2-fn:xmlcolumn('MY_XML_DATA.X')/personen/persoon
       where starts-with($y/nummers/telnr,"01")
       return $y/naam
```

'XQuery' ondervraagt hier dus de documenten die door de functie "`xmlcolumn`" van namespace "`db2-fn`" geproduceerd worden. Bemerk de (noodzakelijke) hoofdletters in de tabel- en kolom-namen!

De tweede query hierboven als smaakmaker voor XQuery's eigen query-taal, met als ingrediënten 'for - in - where - return', als tegenhanger voor 'select - from - where' in SQL.

Shredding: de hybride aanpak

Een tussenoplossing (noch puur XML, noch puur relationeel) voor de telefoonboek-data zou kunnen bestaan uit een relationele tabel met één of meer XML-kolommen. In ons eenvoudig voorbeeld zou dat een tabel kunnen zijn met de kolommen 'naam' en 'nummers', waarbij deze laatste het datatype XML heeft en bestaat uit data van de vorm

```
<nummers><telnr>0123456789</telnr><telnr>0132456789</telnr></nummers>
```

dus essentieel uit lijsten telefoonnummers. Het parsen en daarna 'uiteenrafelen' van de XML uit tabel 1 naar deze tabel, wordt 'shredding' genoemd. Ook dat kan relatief eenvoudig in DB2 v9, met een combinatie van XQuery en SQL.

Voorbeeld:

```
SELECT naam, xmlquery('$XD/nummers' passing t.nummers as XD)
FROM   my_xml_data AS t
WHERE  xmlexists('$XD/nummers/telnr/text()' passing t.nummers as XD)
```

Ook hierover meer in een volgende bijdrage.

.

CURSUSPLANNING AUG - DEC 2007

DB2-concepten		op aanvraag
DB2 for z/OS, een totaaloverzicht	1825 EUR	27-31/08 (L), 08-12/10 (W), 26-30/11 (L), 10-14/12 (W)
DB2 UDB for LUW, een totaaloverzicht	1750 EUR	08-12/10 (W), 15-19/10 (L), 10-14/12 (W)
RDBMS-concepten	350 EUR	27/08 (L), 03/09 (W), 08/10 (W), 26/11 (L), 10/12 (W)
Basiskennis SQL	350 EUR	28/08 (L), 04/09 (W), 09/10 (W), 27/11 (L), 11/12 (W)
DB2 for z/OS basiscursus	1125 EUR	29-31/08 (L), 10-12/10 (W), 28-30/11 (L), 12-14/12 (W)
DB2 UDB for LUW basiscursus	1050 EUR	10-12/10 (W), 17-19/10 (L), 12-14/12 (W)
SQL workshop	750 EUR	13-14/09 (L), 27-28/09 (W), 17-18/12 (W), 19-20/12 (L)
SQL voor gevorderden *(nieuw)*	425 EUR	28/09 (L)
Gebruik van DB2 procedural extensions	425 EUR	18/10 (W)
DB2 for z/OS advanced programming	800 EUR	19-20/11 (L)
DB2 for z/OS: SQL performance	1275 EUR	14-16/11 (L)
XML in DB2	425 EUR	30/11 (L), 11/12 (W)
DB2 for z/OS database administratie	1700 EUR	23-26/10 (W), 03-06/12 (L)
DB2 for z/OS operations and recovery	1425 EUR	05-07/11 (W)
DB2 for z/OS system performance and tuning	1000 EUR	08-09/11 (W)
DB2 UDB DBA 1: Kernvaardigheden	1700 EUR	24-27/09 (L), 13-16/11 (W)
DB2 UDB DBA 2: configure & tune	1275 EUR	19-21/12 (W)
DB2 v8 upgrade		op aanvraag

Plaats: L = Leuven, W = Woerden; zie www.abis.be voor details en extra cursussen.

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