ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 5 - NUMMER 2, MAART 2008

OPEN CURSOR

Met de geïntegreerde ondersteuning voor XML in DB2 9 ziet het ernaar uit dat de hiërarchische datarepresentatie definitief z'n intrede doet in de relationele wereld. Dit was dan ook één van de thema's op recente DB2-bijeenkomsten zoals de IDUG-conferenties of de regionale GSE-meetings. Waar men trouwens ook al begint te praten over "version X". En dat terwijl velen onder ons nog volop met de migratie naar versie 8 bezig zijn.
Onze suggestie: leg dat soort dagelijkse beslommeringen even terzijde en droom met deze "Exploring DB2" even weg in de wondere wereld van DB2 9.

Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- We hebben voor u een tweede onderwerp in de reeks DB2 9, met daarin die andere grote nieuwigheid van de laatste DB2-release: "*Index by Expression" in DB2 9 for z/OS*.
- Zoals beloofd in het vorige nummer, waarin we XML in DB2 9 onder de loep namen, gaan we deze keer in op de "query-taal" van XML: *XPath – een tutorial*.
- En voor wie er niet bij kon zijn, vermelden we kort enkele "highlights" van de voorbije IDUG-conferentie in Athene van 5–9 nov. 2007. Misschien toch eens die presentatie-slides bij een collega lenen die er wel was, of inschrijven voor IDUG 2008 ...

2

CLOSE CURSOR

Ook in het volgende nummer blijven we u op de hoogte houden van alles wat reilt en zeilt binnen DB2. Bijdragen over compressie en over auto-rebind zijn in de maak. Of misschien wilt u zelf iets kwijt? Uw bijdrage is altijd welkom bij de redactie:
training@abis.be

“Index by Expression” in DB2 9 for z/OS

Kris Van Thillo (ABIS)

Als u ooit voor DB2 for z/OS applicaties hebt ontwikkeld, kent u ze ongetwijfeld: de 'recht-toe-recht-aan'-regeltjes die u moet naleven om efficiënte applicaties te ontwikkelen. Eén van deze regels luidt: 'Gij zult nooit expressies of functies toepassen op kolommen in *where*-condities als index-access op deze kolom wenselijk is'.

De uitdaging

De achterliggende logica is natuurlijk duidelijk: DB2 is pas in staat data op te halen op basis van index-access als de WHERE-conditie van de query geen gebruik maakt van functies en/of expressies op de indexkolommen. Met een aantal vervelende gevolgen: bijvoorbeeld de noodzaak data in DB2 zodanig op te slaan dat zoeken op basis van een index, mogelijk is. En dus maken we bijvoorbeeld een verschil tussen een 'display'-kolom die gegevens bevat zoals door een gebruiker ingegeven enerzijds, en een 'zoek'-kolom op basis waarvan DB2 via bijvoorbeeld index-access data kan opzoeken. Synchronisatie tussen beide kolommen is dan ofwel de verantwoordelijkheid van de applicatie (die de waarde op een intelligente wijze aan beide kolommen gaat toevoegen, al dan niet versleuteld); of men gaat triggers gebruiken om dit hele verhaal 'applicatie-transparant' te maken. Immers, na het toevoegen van een waarde aan een eerste kolom (bijvoorbeeld de 'display'-kolom) zorgt de trigger voor het kopiëren van de waarde naar een tweede kolom (bijvoorbeeld de 'zoek'-kolom voor index-access).

De oplossing

In DB2 9 for z/OS wordt ons een andere oplossing aangeboden - misschien 'de' oplossing: een *”Index by Expression”*-optie.

Het idee is eigenlijk heel eenvoudig: een index wordt niet langer aangemaakt op één of meerdere kolommen van een tabel, maar op een *expressie* of functie op één of meerdere kolommen! Of met andere woorden: een index kan worden aangemaakt op afgeleide data (dus data die als dusdanig strikt genomen niet in de tabel aanwezig zijn). Niet langer de kolomdata worden opgeslagen in de index, maar het resultaat van de toepassing van de expressie of functie op een bepaalde kolom (of reeks kolommen).

Het spreekt voor zich dat hiertoe het *CREATE INDEX* statement slechts minimaal diende te worden aangepast.

Niet alle expressies en functies zijn op dit moment in DB2 9 for z/OS zonder meer bruikbaar. Een aantal voorwaarden (en beperkingen) moeten we als steeds voor ogen houden. Zonder volledig te willen zijn, toch een korte opsomming:

- de lengte van de text string is beperkt tot 4000 bytes;
- de index *moet* minstens een kolom bevatten van de tabel waarop de index ligt, en kan enkel kolommen van die tabel bevatten;
- kolommen met speciale datatypes worden niet ondersteund (zoals LOB, XML). Merk op dat user defined data types wel ondersteund worden (uiteraard enkel indien ze niet op deze speciale datatypes gestoeld zijn);
- de index-expressie mag geen subqueries, aggregates, CASE- en OLAP-functies bevatten;
- het resultaat van de expressie moet deterministisch zijn - zeg maar, op basis van de expressie-INPUT een voorspelbare en vaste OUTPUT opleveren. Dit sluit dus het gebruik van volgende elementen uit: host-variabelen, sequences, special register variabelen, externe functies, ...

Praktisch gezien kunnen we ervan uitgaan dat voornamelijk scalaire functies en wiskundige berekeningen als expressies in deze context vaak aan bod zullen komen.

In wat volgt is een voorbeeld opgenomen om e.e.a. nader te verduidelijken. Om niet onmiddellijk het standaard “uppercase/lowercase”-voorbeeld erbij te halen, staan we stil bij het volgende scenario. Een cursus heeft naast een titel ook een cursuscode (“cid”), gedefinieerd als een CHAR(4). De eerste 2 tekens van deze code geven aan tot welke primaire cursusgroep deze cursus behoort. Queries die van deze informatie in de WHERE-condities gebruik maken kunnen door dit nieuwe INDEX-feature aanzienlijk efficiënter worden uitgevoerd.

Voorbeeld

```
CREATE INDEX  ix_cgrp  ON courses SUBSTR(cid, 1, 2) ;

SELECT ctitle
FROM   courses
WHERE  substr(cid, 1, 2) = 19; -- DB2 Courses only please!
```

De implicaties naar efficiëntie toe zijn duidelijk. Zonder expressie in de index zal een WHERE-conditie die de SUBSTR-functie aanroept nooit van een index gebruik kunnen maken. Een tablespace scan (of afhankelijk van de situatie mogelijk een index scan) is dan vaak het enige alternatief. Expressies in de index maken het voor de optimizer mogelijk sneller te 'matchen'; immers, de relevante expressie/functie-'data' zijn aanwezig in de index! Screening is eveneens mogelijk; en de logica eigen aan het werken met MQTs is ook beschikbaar: een index gebouwd op col1 + col2 kan worden gebruikt ook als de *where*-conditie refereert naar col2 + col1! Wat is trouwens, zeker conceptueel, het verschil tussen een 'index by expression' INDEX en een MQT die permanent 'in sync' wordt gehouden?

Vanuit query runtime standpunt is overhead eigen aan het gebruik van deze index zo goed als onbestaand. De index wordt, als steeds, automatisch bijgewerkt bij het uitvoeren van DML-operaties op de betreffende tabel; alsook bij het uitvoeren van index-onderhoud aan

de hand van UTILITIES. De voordelen voor leestoegang zijn aanzienlijk: daling van de CPU-consumptie enerzijds, en het aantal te behandelen data/index-pagina's anderzijds, ten gevolge van betere accesspath-selectie, ligt hieraan ten grondslag.

We hadden het reeds even over de overeenkomst die bestaat tussen deze index-optie en MQTs. Laten we hopen dat IBM de hier ingeslagen weg verderzet: wat te denken van *Join indexes* bijvoorbeeld, waarbij de index op één tabel waarden uit een andere tabel kan opslaan? Datawarehouse DBAs kunnen zich hier ongetwijfeld iets bij voorstellen.

Naarmate DB2 verder evolueert, sneuvelen evidente regeltjes. Het is onze verantwoordelijkheid als DB2-gebruikers deze evolutie te onderkennen, en de mogelijkheden die deze evolutie biedt volop te gebruiken. Zodat we binnenkort nog maar één enkele regel moeten naleven. Een regel die voor zich spreekt: *Bovenal, bemin DB2!*

Enkele highlights van IDUG Athene - november 2007

IDUG, de “International DB2 User Group”, is een onafhankelijke organisatie van DB2-gebruikers. De IDUG-conferenties zijn al sinds jaren één van de belangrijkste gelegenheden voor DB2-gebruikers om ervaringen uit te wisselen. IDUG is méér dan een conferentie: er is o.a. een zeer actieve mailing-list, een elektronische "Solutions Journal", lokale usergroups, en een online-aanbod op *http://www.idug.org/*.

De jaarlijkse Europese versie van de IDUG-conference ging deze keer door in Athene van 5 tot 9 november 2007. IDUG-leden kunnen de slides van de presentaties raadplegen op de IDUG-website. Er waren in totaal een 120-tal lezingen van telkens 1 uur. Een poging om de belangrijkste hoogtepunten te vermelden:

- “user experiences” met de migratie op z/OS naar versie 8; er was hierover o.a. een zeer boeiende lezing van John Campbell
- verschillende lezingen over DB2 9 for z/OS; i.h.b. de “best speaker award” voor Bryan Paulsen met “DB2 9 for z/OS - John Deere’s ESP Experience”, en lezingen over performance (John Campbell), over LOBs, over “New Cool SQL” (Willie Favero), “SQL challenges” (Susan Lawson), log analyzer (Phil Grainger), ...
- enkele opmerkelijke thema’s waren verder: Real-time statistics (Bryan Smith), recursive SQL (Suresh Sane), zIIPs (Terry Purcell), performance (Jan Henderyckx, e.a.), Rebind (Bonnie Baker), data warehouses (Roger Miller), tuning (William Miller), security (Roger Miller; Bryce Krohn), XML (Ralf Neumann), ...
- Belangrijkste thema’s onder de noemer “LUW”: query optimization (John Hornibrook), compression (Martin Hubel, George Baklarz), triggers (Daniel Luksetich), en natuurlijk “pureXML” (Matthias Nicola).
- En niet te vergeten de vijf one-day seminars, die naar verluidt allemaal heel goed waren, met o.a. “DB2 9 for z/OS in-depth” door Phil Granger, en “Things I wish they told me 8 years ago, part 10” door Bonnie Baker.

Op onze website vindt u de presentatie die ikzelf verzorgde op de conferentie (zie *http://www.abis.be/resources/presentations/idug20071106ctedb2.pdf*) .

Peter Vanroose

XPath - een tutorial

Ludo Van den dries (ABIS)

Dit jaar viert XML zijn tiende verjaardag. Met terechte fierheid, want ondertussen is XML uitgegroeid tot een populaire manier om tekstgegevens voor te stellen, geschikt voor uitwisseling én opslag in zeer diverse toepassingen. De regels voor 'well-formed' XML zijn immers eenvoudig: schrijf de data in een universele karakterset (Unicode), voorzie ze van zelf te kiezen tags (markup) en zorg dat het alles samen een nette boomstructuur vormt (tree), met één root-element op de kop en daaronder child-elementen (die uiteindelijk tekst-inhoud bevatten); vul het geheel aan met andere 'nodes', zoals attributen en commentaar.

Het eerste artikel in deze reeks beschreef hoe XML als volwaardig datatype geïntegreerd is in DB2 9, met eigen manieren van opslag én ook een eigen manier van opvraging: XQuery (op zichzelf, of ingebed in SQL). Ons tweede artikel handelt over de techniek die aan de grondslag ligt van XQuery, namelijk XPath.

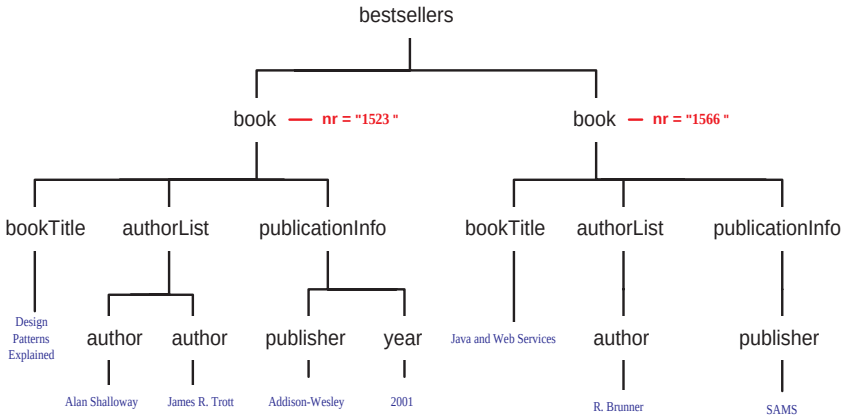
XPath als hulptaal

XPath is een 'taal', een syntax die gebruikt wordt om onderdelen van een XML node-tree te selecteren (of gegevens over die onderdelen weer te geven). Het is een basistechniek, die niet op zichzelf gebruikt wordt, maar 'ingebed' zit in verschillende andere standaarden, bijvoorbeeld in XQuery bij het zoeken in XML-documenten, in XSLT bij het transformeren van documenten, in XML Schema voor het opleggen van documentstructuur en in XLink voor het linken van documenten.

Ook in DB2 9 worden XPath-expressies gebruikt om delen van een XML-document te extraheren, zowel in de SQL interface (wanneer functies gebruikt worden waarvan de naam met “xml” begint) als (uiteraard) in de nieuwe XQuery interface.

Ons werkvoorbeeld

```
<bestsellers>
   <book nr="1523">
      <bookTitle>Design Patterns Explained</bookTitle>
      <authorList>
         <author>Alan Shaloway</author>
         <author>James R. Trott</author>
      </authorList>
      <publicationInfo>
         <publisher>Addison-Wesley</publisher>
         <year>2001</year>
      </publicationInfo>
   </book>
   <book nr="1566">
      <bookTitle>Java and WebServices</bookTitle>
      <authorList>
         <author>R. Brunner</author>
      </authorList>
      <publicationInfo>
         <publisher>SAMS</publisher>
      </publicationInfo>
   </book>
</bestsellers>
```



De XPath-expressie

Het basisprincipe van XPath is simpel: hoe navigeren we door de node-tree (d.w.z. welk 'path' volgen we) om tot bij de gezochte nodes te komen?

Een XPath-expressie (= *location path*) bestaat uit één of meer stappen. Elke stap vertrekt vanuit een startpositie (de 'context node', het referentiepunt) en bestaat uit *drie* onderdelen:

- *waar* zoeken we? (de "*axis*")
- *wat* zoeken we? (de "*node-test*")
- welke van de gevonden nodes willen we eventueel nog *wegfilteren*? (d.m.v. "*predikaten*")

De algemene syntax van één stap is:

```
axis::node-test[predikaat1][predikaat2][etc...]
```

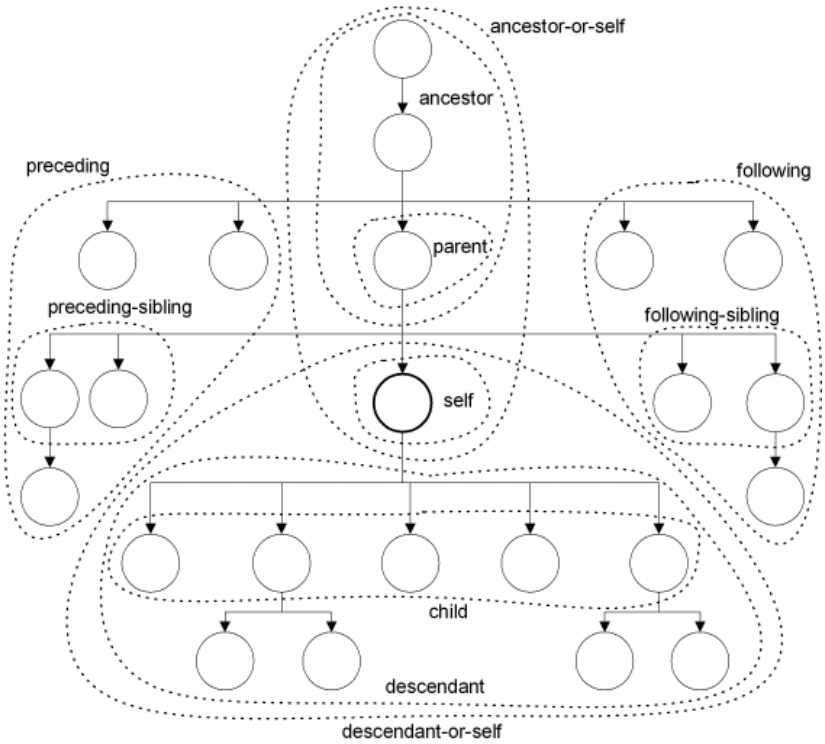
De axis

XPath steunt vooral op de structuur van de node-tree, d.i. de boomstructuur waarin alle soorten nodes (element, text, attribute, comment, etc....) verenigd zijn onder de 'root node'. Deze heeft zelf geen naam, maar wordt voorgesteld met een slash. Zoals gebruikelijk bij boomstructuren, worden de onderlinge verhoudingen uitgedrukt in stamboom-termen: ouder, kind, broers/zussen, voorouders, enz.

Het "*axis*"-gedeelte van de XPath-expressie geeft de *zoekrichting* aan binnen de boom t.o.v. de context-node. Veelgebruikte axes zijn:

child (= de default axis), descendant, self, parent, ancestor, preceding-sibling en following-sibling (een "zus-of-broer"), attribute, ... Zie de figuur hieronder voor een volledig overzicht.

Merk op dat attributen bij een element, zoals b.v. nr="1523" bij `book`, niet als kinderen van dat element beschouwd worden: we moeten ze niet zoeken in de kind-richting, maar in de attribuut-richting! (Attributen zijn ook a.h.w. steriel: ze hebben zelf nooit kinderen.)



De node-test

Met de node-test geven we aan wát we zoeken. Dit kan op twee manieren (hetzij/hetzij):

- het *node-type*: node(), text(), comment()
- de *node-naam*: book, author, *, ...

Enkele voorbeelden van axis + node-test (waarbij de context-node het eerste '`book`' is):

`child::bookTitle`	*geeft het element 'bookTitle' (dus* `<bookTitle>DesignPatternsExplained</bookTitle>`*)*
`bookTitle`	*geeft hetzelfde, want* `child::` *is de default axis.*
`child::author`	*geeft niks (alias: een lege node-set)* *(want geen enkel 'author'-element is een rechtstreeks kind van 'book')*
`descendant::author`	*geeft 2 auteurs (Alan & James)*
`descendant::node()`	*geeft de verzameling (set) van alle nodes binnen het eerste 'book', netjes op een rij* *(een bookTitle, een authorList, twee authors, een publicationInfo,* *publisher en year, en daarbij nog eens alle text-nodes* *als aparte nodes: 'Design Patterns...', 'James...', etc.)*
`descendant::text()`	*geeft uiteraard alleen de betreffende text-nodes*
`descendant::*`	*geeft alle element-nodes (niet de text-nodes of attribuut-nodes!)* *(de elementnaam valt onder de wildcard *; text-nodes hebben geen naam)*
`parent::node()`	*geeft het element 'bestsellers'*
`self::node()`	*geeft het element 'book' zelf (het eerste book)*
`following-sibling::book`	*geeft het tweede 'book'*

Predikaten

Predikaten zijn optioneel: zij leggen bijkomende voorwaarden op aan de nodes die we gevonden hebben uit de axis::node-test, en dienen dus als een bijkomend filter.

Bijvoorbeeld, dit keer vertrekkend van 'bestsellers' als context node:

`child::book`	*geeft een set met de twee 'book's*
`child::book[position()=2]`	*geeft een set met slechts het tweede 'book'*

Samengesteld location path

Een *location path* kan samengesteld worden uit verschillende stappen, bij voorbeeld:

```
child::book/descendant::author
```

- de verschillende stappen worden gescheiden door een slash;
- we vertrekken vanuit één context node (bvb. het eerste 'book');
- de eerste stap levert 2 nodes, die nu de nieuwe context vormen voor de volgende stap (die dus 2 keer wordt geëvalueerd: met het 1ste 'book' als context node, en daarna met het 2de 'book' als context node); de verzamelde resultaten leveren op hun beurt weer een node-set die als start-context voor een eventuele volgende stap zal dienen...
- Elke stap mag uiteraard zijn eigen predikaten hebben; een predikaat kan eveneens uit meerdere stappen bestaan.

Resultaat:

```
child::book/descendant::author                 geeft de drie "author"-elementen
child::book[position()=2]/descendant::author   geeft alleen "R. Brunner"
child::book/attribute::nr                      geeft de beide 'nr' attributen
    (vaak afgekort als book/@nr)
```

Absoluut en relatief pad

Tot nu toe zijn we in onze voorbeelden steeds gestart vanaf een opgegeven context: dit heet een *relatief* pad. Men kan ook starten vanaf de root node (aangeduid met "/") d.m.v. een *absoluut* pad:

```
/child::bestsellers/child::book/child::bookTitle geeft beide 'bookTitle'-elementen
/bestsellers/book/bookTitle                      geeft hetzelfde (een beetje leesbaarder...)
```

Afkortingen

We hebben al verschillende verkorte notaties ontmoet; handig als je maar onthoudt wat er achter schuil gaat... Hier nog enkele:

```
@nr                          <==>  attribute::nr
book[2]                      <==>  book[position()=2]
book[*/year=2001]/bookTitle  <==>  book[child::*/year/text()="2001"]/child::bookTitle
/bestsellers//bookTitle      <==>  /bestsellers/descendant-or-self::node()/bookTitle
    (vaak fout begrepen!...)
../bookTitle                 <==>  parent::node()/bookTitle
```

Dit laatste voorbeeld doet eens te meer denken aan de notatie voor bestanden en folders in enkele populaire operating-systemen. (Niet verwonderlijk: dat zijn ook bomen.)

Datatypes en functies

Onze voorbeelden resulteerden tot nu toe steeds in een node-set (met nul of meerdere nodes erin). In het algemeen kan een XPath-expressie leiden tot 4 soorten data:

- node-set
- string
- number
- boolean

Dit komt vooral tot uiting bij gebruik van de XPath *functies*:

```
concat(//bookTitle[1],"xx",//bookTitle[2])  geeft een string met de twee aaneengeplakte 'bookTitles':
                                            "Design Patterns ExplainedxxJava and WebServices"
count(//book)                               geeft het totale aantal 'book'-elementen (het getal 2 dus)
contains(//bookTitle[1], "Patterns")        geeft booleantrue, want de eerste 'bookTitle' bevat "Patterns"
```

Hierbij merken we op dat XPath heel bereidwillig types converteert als dat nodig is voor de operatie of functie. Zie het concat-voorbeeld: //bookTitle[1] geeft een volledige node, maar de *concat()*-functie verwacht een string als eerste argument, en dus wordt automatisch de string-waarde van de node //bookTitle[1] genomen.

XPath en DB2 9

Stel dat we het bovenstaande XML-document in DB2-tabel “MYTABLE” hebben opgeslagen, in de kolom "MYXMLCOL". De volgende SQL-query geeft dan de auteur (R. Brunner) van het tweede boek:

```
SELECT xmlquery('/bestsellers/book[2]/authorList/author/text()'
                passing myXMLcol) AS auteur
FROM   mytable
WHERE  <rijselectie>
```

Let op de aanhalingstekens: XML (en dus ook XPath) zijn hoofdlettergevoelig, terwijl SQL dat niet is!

Korter (maar wellicht minder performant):

```
SELECT xmlquery('//book[2]//author/text()' passing myXMLcol) AS auteur
FROM   mytable   WHERE <rijselectie>
```

Wanneer we de XQuery-interface van DB2 9 gebruiken (voorlopig alleen in de LUW-versie beschikbaar), wordt dit:

```
XQUERY
db2-fn:xmlcolumn('MYTABLE.MYXMLCOL')/bestsellers/book[2]/authorList/author/text()
```

Let opnieuw op de hoofdlettergevoeligheid van XQuery: MYTABLE.MYXMLCOL moet in hoofdletters opgegeven worden, en verder alles in de juiste "mixed case".

Ten slotte gebruikt ook de functie XMLEXISTS de XPath-syntax. Stel dat de tabel bevolkt is met meerdere XML-documenten die allemaal exact één `<book>`-element bevatten. De volgende query geeft dan alle boektitels gepubliceerd in 2001:

```
SELECT xmlquery('//book/bookTitle/text()' passing myXMLcol) AS titel
FROM   mytable
WHERE  xmlexists('//book[publicationInfo/year/text()="2001"]' passing XMLcol)
```

Dit geeft echter onverwachte resultaten (met name aaneengeplakte titels, titels van het verkeerde jaar) indien één XML-document meerdere boeken bevat. XQuery is dan flexibeler:

```
XQUERY for $book in
        db2-fn:xmlcolumn('MYTABLE.MYXMLCOL')//book[publicationInfo/year=2001]
      return 'bookTitle'
```

Bemerk ook het gebruik van een relatief pad, zowel in de return (“`bookTitle`”) als in de conditie (“`publicationInfo/year`”). Merk ook de verkorte vorm "`year=2001`" i.p.v. "`year/text()="2001"` ": beide zijn equivalent, zoals hogerop uitgelegd werd.

Verwar ook `XQUERY` en `xmlquery()` niet: de eerste is een nieuwe (niet-relationele) interface voor DB2, onafhankelijk van en naast SQL; de tweede is een SQL-functie die b.v. in een SELECT-statement gebruikt kan worden! De enige gelijkenis tussen de twee is dat beide XPath gebruiken. De uitdrukking “`db2-fn:xmlcolumn`” daarentegen is geen scalaire functie maar een standaard XQUERY-functie.

XPath 1.0 en 2.0

XPath 1.0 dateert van 1999, en werd ontwikkeld als ondersteunende syntax voor standaarden als XSLT 1.0 (transformeren van XML) en XLink/XPointer (linken van XML). De bovenstaande introductie behandelde vooral deze XPath 1.0 versie.

In 2007 verscheen, na lange barensweeën, eindelijk XQuery 1.0, de standaard voor het ondervragen van XML-data (in één document of een verzameling van documenten, bvb in een DB2 9 database). Dit bracht ook nieuwe versies (2.0) van XPath én XSLT met zich mee. Voor XPath betekende dit vele nieuwe mogelijkheden, maar ook een veranderde syntax, en geen volledige 'backward-compatibility' (maar wees gerust: de bovenstaande introductie is zowel bruikbaar voor XPath 2.0 als voor XPath 1.0 ...).

Enkele nieuwigheden:

- rijkere datatypes
  (inclusief user-defined, in combinatie met XML Schema)
- veel meer functies en operatoren (ook user-defined)
- uitgebreide programeerbaarheid (bvb 'for'-lussen)
- node-sets worden 'sequences', met meer vrijheden inzake volgorde etc.

De populariteit van XPath 2.0 is op dit ogenblik nog onduidelijk: niet alle tools en libraries hebben hem reeds geïntegreerd. Het is duidelijk dat de 'embedding' standaarden (zoals XSLT 2.0, maar vooral XQuery 1.0) de drijvende kracht achter de algemene aanvaarding van XPath 2.0 zullen zijn. Een reden te meer om met belangstelling de ontwikkelingen van DB2 9 te volgen, want daarin worden deze nieuwe standaarden (minstens gedeeltelijk) ondersteund.

CURSUSPLANNING APRIL - JULI 2008

DB2-concepten		op aanvraag
DB2 for z/OS, een totaaloverzicht	1900 EUR	14-18 apr (L), 15-21 mei (W), 2-6 jun (L)
DB2 UDB for LUW, totaaloverzicht	1825 EUR	15-21 mei (W)
RDBMS-concepten	350 EUR	14 apr (L), 15 mei (W), 2 jun (L)
Basiskennis SQL	350 EUR	15 apr (L), 16 mei (W), 3 jun (L)
DB2 for z/OS basiscursus	1200 EUR	16-18 apr (L), 19-21 mei (W), 4-6 jun (L)
DB2 UDB for LUW basiscursus	1125 EUR	19-21 mei (W)
SQL-QMF voor eindgebruikers	1200 EUR	14-16 mei (W)
SQL workshop	750 EUR	14-15 apr (W), 28-29 apr (L), 12-13 jun (L), 23-24 jun (W)
SQL voor gevorderden	425 EUR	30 apr (L), 25 jun (W)
Gebruik van DB2 procedural extensions	425 EUR	26 jun (W)
DB2 for z/OS programmeren voor gevorderden	800 EUR	22-23 mei (W)
DB2 for z/OS: SQL performance	1275 EUR	16-18 jun (W)
XML in DB2	425 EUR	14 apr (W)
DB2 for z/OS database administratie	1700 EUR	7-10 apr (W), 9-12 jun (L)
DB2 for z/OS operations & recovery	1500 EUR	21-23 apr (W)
DB2 for z/OS installation&migration	1050 EUR	16-17 jun (High Wycombe)
DB2 for z/OS system performance and tuning	1050 EUR	24-25 apr (W)
DB2 LUW DBA 1: Kernvaardigheden	1700 EUR	6-9 mei (W)
DB2 LUW DBA 2: configure & tune	1275 EUR	11-13 jun (W)
DB2 v8 upgrade		op aanvraag
DB2 v9 upgrade		op aanvraag

Plaats: L = Leuven, W = Woerden; voor details en andere cursussen, zie `http://www.abis.be/`

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