ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 7 - NUMMER 4, DECEMBER 2011

OPEN CURSOR

DB2 10 for z/OS is op kruissnelheid; waar dat niet al gebeurd is, plant men de migratie naar versie 10 in de loop van 2012.
Aanleiding genoeg voor Exploring DB2 om de rubriek "Dossier 10" te starten, naar analogie met onze vroegere dossiers 7, 8 en 9.
Ook het hoofdartikel gaat over een opvallende nieuwe mogelijkheid van versie 10 (als vervolg op een eerdere bijdrage): temporele data.
Inderdaad een zeer welkome nieuwe mogelijkheid van DB2, want het standaardiseert een database-design die u vermoedelijk zelf al enkele malen hebt moeten "her-uitvinden", denk maar aan de catalog-tabel SYSIBM.SYSCOPY op z/OS.
Lees snel het hoofdartikel voor meer details, en neem er eventueel eerst het mei-nummer van Exploring DB2 terug bij!

Veel leesplezier,
Het ABIS DB2-team.

IN DIT NUMMER:

- "*Temporele data en DB2 10 for z/OS - deel 2*" beschrijft aan de hand van een uitgewerkt voorbeeld hoe data-versioning en historiek autonoom door DB2 kan beheerd worden.
- In een tweede bijdrage, "*Scalaire functies: zo doet SQL Server het*", vindt u een overzicht van de gelijkenissen en de (soms subtiele) verschillen tussen wat DB2 en SQL Server op dit gebied te bieden hebben.
- En tenslotte "*DB2 10 for z/OS: enkele opvallende nieuwigheden*", een zeer beknopte samenvatting van de meest in het oog springende nieuwe mogelijkheden van versie 10, als aanzet voor een dossier-10-reeks in onze volgende nummers.

4

CLOSE CURSOR

Ook in de volgende nummers van Exploring DB2 zullen we, zoals u ondertussen van ons gewoon bent, enkele opvallende, nuttige, of minder bekende aspecten van DB2 onder de loep nemen.

Hebt u suggesties voor een interessant onderwerp? We horen het graag!

Temporele data en DB2 10 for z/OS (II)

Kris Van Thillo (ABIS)

In een vorig nummer stonden we stil bij een aantal conceptuele bedenkingen bij temporele data in DB2 for z/OS, waarvoor sinds versie 10 ondersteuning wordt geboden.

In dit en een volgend artikel gaan we dieper in op een aantal details. We gebruiken een eenvoudig voorbeeld uit onze eigen administratieve omgeving: een cursusprijstabel. En we richten ons in eerste instantie op de implementatie van temporele data op basis van '*system time*' als referentiekader.

Opzet

Voor opleidingen die we bieden wensen we aan te geven wanneer een bepaalde prijs van toepassing is. Concreet: elke prijs heeft een bepaalde geldigheidsduur. Dit zou bijvoorbeeld op een eenvoudige manier kunnen worden geïmplementeerd op basis van onderstaande tabel.

Tabelcreatie.

```
create table coursefees
(cfno        int           not null primary key,
 cf_courseid int           references courses on delete cascade,
 cfprice     decimal(6,2)  not null,
 cfstart     timestamp(12) generated always as row begin not null,
 cfstop      timestamp(12) generated always as row end   not null,
 cftrans_id  timestamp(12) generated always as transaction start id
                             implicitely hidden,
 period system_time (cfstart, cfstop)
);

create table cf_hist like coursefees;

alter table coursefees add versioning use history table cf_hist;
```

De basistabel bevat drie verplichte `TIMESTAMP(12)` kolommen (d.w.z. met de maximale precisie van een picoseconde; tussen haakjes: eigenlijk zijn dit nanoseconden gevolgd door drie decimalen die uniek zijn voor het subsysteem): ze bevatten naast de start/stop systeemtijd die de 'levensduur' van een rij aangeven ook een transactie-starttijd. Deze laatste wordt door DB2 gebruikt om bij te houden wanneer een transactie voor het eerst een rij heeft gewijzigd (*transaction start timestamp*). Let op de definitie van deze kolommen in bovenstaand voorbeeld - allemaal '*generated always*', dus beheerd door DB2. Voor de laatste kolom kozen we er tevens voor deze verborgen te houden want eigenlijk zonder functionele 'business'-betekenis, dit in tegenstelling tot beide andere timestamp-kolommen. De enige garantie is dat die waarde uniek is (en gemeenschappelijk) voor de lopende transactie.

De schaduwtabel bevat de 'oude' rijen na wijziging door een applicatie/transactie - en moet dus structureel identiek zijn aan de basistabel. Bemerk in het voorbeeld (laatste `alter` statement) hoe deze met zijn overeenkomende basistabel wordt gelinked.

Temporele ondersteuning (versioning) kan tevens achteraf aan een bestaande tabel worden toegevoegd, met standaard `alter table` statements. Merk op dat het naderhand wijzigen van een basistabel (aan de hand dus van bijkomende `alter table` statements) enkel mogelijk is na het opheffen van de temporele ondersteuning.

Het moment is nu aangebroken om bovenstaand opzet ook daadwerkelijk te gaan gebruiken!

DML – Update, delete en insert

Traditionele, standaard DML-instructies kunnen gewoon worden gebruikt. Belangrijk hierbij is de transactietijd: het tijdstip waarop de transactie is gestart die het DML-statement bevat.

- voor *insert* operaties zal DB2 gewoon de rij inserten in de basistabel. De insert transactietijd wordt opgenomen in de kolom die de levensduur start-systeemtijd aangeeft; de levensduur stop-systeemtijd wordt conceptueel op 'oneindig' geplaatst – concreet, op de maximale tijdswaarde die door DB2 ondersteund wordt. Hiermee geeft DB2 gewoon aan dat deze rij de huidige, current rij is en dus voorlopig onbeperkt lang geldig is;
- voor *update* operaties zal DB2 de originele rij gewoon kopiëren naar de schaduwtabel, de originele levensduur start-systeemtijd behouden, en de levensduur stop-systeemtijd gelijkstellen aan de timestamp van de update transactie. In de originele tabel wordt de update uitgevoerd: de oorspronkelijke levensduur start-systeemtijd wordt vervangen door de timestamp van de start van de update transactie;
- voor *delete* statements wordt de rij uit de basistabel weggehaald en naar de history tabel verhuisd, waar de levensduur stop systeemtijd gelijkgesteld wordt aan de delete statement transactietijd.

Het is belangrijk te noteren dat alle bovenstaande acties door DB2 automatisch worden uitgevoerd, dus volledig transparant voor de applicatie. Inderdaad, DB2 genereert alle benodigde informatie, en voegt die toe aan de hiertoe vooraf aangemaakte kolommen in zowel de basistabel als de schaduwtabel. Hierdoor ontstaat een authentieke 'rij-historiek' die voor allerlei doelstellingen kan worden gebruikt. In het verleden zouden relatief complexe triggers nodig zijn geweest om een gelijkaardigs functionaliteit te implementeren!

Een aantal voorbeelden volgen.

DML

```
insert into coursefees                    -- op tijdstip 01.02.2011 10:53
values (1111, 'DB2', 450);

insert into coursefees                    -- op tijdstip 01.03.2011 15:32
values (1112, 'ORA', 460);

update coursefees                         -- op tijdstip 01.04.2011 11:43
set cfprice = 440
where cfno = 1112;

delete from coursefees                    -- op tijdstip 01.05.2011 14:17
where cf_courseid = 'ORA'
```

Als bovenstaande statements zijn uitgewerkt tegen de achtergrond van de opgenomen referentietijden, merken we het volgende op:

- de *basistabel* bevat nu 1 rij;
- de *schaduwtabel* bevat nog steeds geen informatie betreffende de DB2-rij (`cfno=1111`), maar bevat wel 2 rijen die de wijzigingsgeschiedenis van de ORA-rij (`cfno=1112`) bevatten.

De inhoud van beide tabellen is dan als volgt (de hidden kolom is ook in dit voorbeeld 'hidden'):

Inhoud Basistabel/Schaduwtabel

coursefees

cfno	cf_courseid	cfprice	cfstart	cfstop
1111	DB2	450	01.02.2011 10:53	31.12.9999 23:59

cfhist

cfno	cf_courseid	cfprice	cfstart	cfstop
1112	ORA	460	01.03.2011 15:32	01.04.2011 11:43
1112	ORA	440	01.04.2011 11:43	01.05.2011 14:17

En wat betreft het select statement...

... is e.e.a. makkelijk uit te leggen. De `select` zonder indicatie van levensduur of referentieperiode heeft steeds betrekking op de huidige data – zoals standaard het geval. Echter, zoals ook in deel I van dit artikel (zie Exploring DB2 jaargang 7 nr. 2) uitgelegd, is dit niet meer het geval wanneer het `select` statement uitgebreid wordt met een uitdrukkelijke verwijzing naar een referentieperiode.

select

```
select count(*)
from coursefees
where cf_courseid = 'ORA'
 -- resultaat: 0


select cfprice
from coursefees
     for system_time from time as of timestamp ('2011-03-14')
where cf_courseid = 'ORA'
 -- resultaat: 460.00
```

Andere alternatieven om de `system_time` in rekening te brengen maken gebruik van de functies '`for system_time from ... to ...`' (de 'to' tijd *niet* inbegrepen) of '`for system_time between ... and ...`' (waarbij de tijd in het rechterlid *wel* is inbegrepen in het zoekcriterium).

Ook hier gebeurt heel wat achter de schermen: inderdaad, de verwijzing naar een referentieperiode is voldoende opdat DB2 transparant niet enkel de basistabel maar ook de schaduwtabel zou doorzoeken (ook al wordt in de from clause naar deze schaduwtabel niet verwezen!) Een dergelijke functionaliteit zou nooit met triggers kunnen gerealiseerd worden, en slechts met zeer veel moeite (en weinig performant) via een view op de twee tabellen!

Tot slot nog even het volgende benadrukken:

- het gebruik van 'system time' biedt enkel de mogelijkheid naar het verleden/periodes in het verleden te verwijzen. Verwijzen naar de toekomst is uitgesloten;
- sommige basistabellen kunnen niet het voorwerp uitmaken van versioning, bijvoorbeeld MQT tabellen, of clone tabellen (of tabellen die een clone hebben);
- utilities die data uit de basistabel zouden kunnen verwijderen kunnen niet op de basistabel (tablespace) worden toegepast (reorg delete, load replace); dit zijn dus situaties waar triggers ook niet in actie zouden treden;
- het toepassen van het truncate statement op de basistabel is niet toegelaten.

In een volgend artikel herwerken we bovenstaand voorbeeld om gebruik te kunnen maken van het referentiekader 'business time'. En maken we kort een overzichtelijke vergelijking tussen beide alternatieven.

Scalaire Functies: zo doet SQL Server het

Steven Scheldeman (ABIS)

Dit is het vierde artikel in de vergelijkende reeks over de SQL scalaire functies van DB2 en andere relationele database-systemen. Deze maal bekijken we de tegenhangers op SQL Server. Net zoals in de vorige bijdragen bekijken we hier SQL Server vanuit de DB2-invalshoek, en vermelden vooral die functies waarvoor een ISO/ANSI-standaardfunctie gedefinieerd is.

Vooraf toch even twee opmerkingen. Ten eerste zal deze lijst niet exhaustief zijn. Enkel de meest gebruikte scalaire functies komen aan bod. Voor een volledige lijst verwijzen we graag naar de manuals.

Ten tweede willen we even opmerken dat het voornamelijk bij de scalaire functies is dat SQL Server blijk geeft een SQL "dialect" te spreken. Waar DB2, Oracle en MySQL redelijk dicht bij de ISO standaard aansluiten, bestaan er toch wel heel wat afwijkingen op SQL Server.

Net als bij de vorige artikels in deze reeks, proberen we in wat volgt de functies zodanig te groeperen, dat onmiddellijk duidelijk wordt waar verschillen bestaan; en hoe hiermee dient te worden omgegaan. Waar er verschillen zijn, wordt met "(D)" de DB2-variant aangegeven en met "(S)" de SQL Server-variant.

Numerieke functies

Volgende lijst geeft de DB2-functies die in SQL Server op een identieke wijze kunnen worden gebruikt:

```
ABS(n), ACOS(n), ASIN(n), ATAN(n), CEILING(n), COS(n), COT(n), DEGREES(n),
EXP(n), FLOOR(n), LOG(n), LOG10(n), POWER(m, n), RADIANS(n), RAND([n]),
ROUND(n[,m]), SIGN(n), SIN(n), SQRT(n), TAN(n), TRUNCATE(n,m)
```

Volgende lijst geeft de twee DB2-functies die in SQL Server op een identieke wijze kunnen worden toegepast, maar wel een "dialectische" schrijfwijze hebben:

```
(D) ATAN2(n, m)      => (S) ATN2(n, m)
(D) LN(n)            => (S) LOG(n)
```

Tekst-manipulatie-functies

In wat volgt herkennen we verschillende argument-'types', als volgt:

- (*c1*,*c2*) vertegenwoordigen "*character strings*"
- (*n*,*m*) vertegenwoordigen numerieke waarden
- (*a*,*b*) vertegenwoordigen individuele letters
- (*set*, to-*set*, *from-set*) vertegenwoordigen reeksen van letters.

Volgende lijst geeft de DB2-tekstfuncties die in SQL Server op een identieke wijze worden toegepast:

```
ASCII(c1),  DIFFERENCE(c1, c2), SUBSTRING(c1, m[,n]), LEFT(c1, n), LTRIM(c1),
RTRIM(c1), LOWER(c1), UPPER(c1), REPLACE(c, a, b), RIGHT(c1, n), NULLIF(c1,
c2), COALESCE (c1, c2, c3, ...)
```

Volgende lijst geeft de DB2-tekstfuncties die in SQL Server eveneens op een identieke wijze worden toegepast, maar wel een “dialectische” schrijfwijze hebben:

```
(D) CHR(a)                  => (S) CHAR(a)
(D) LENGTH(c1)              => (S) LEN(c1)
(D) SUBSTR(c1, m[,n])       => (S) SUBSTRING(c1, m, n)
(D) REPEAT(c1, n)           => (S) REPLICATE(c1, n)
(D) c1 || c2                => (S) c1 + c2
(D) INSERT(c1, n, m, c2)    => (S) STUFF(c1, n, m, c2)
(D) POSITION(c1, c2, n)     => (S) CHARINDEX(c1, c2, n)
(D) LCASE(c1)               => (S) LOWER(c1)
(D) UCASE(c1)               => (S) UPPER(c1)
(D) LOCATE(c2, c1[,n])      => (S) PATINDEX(c2, c1)
```

Volgende DB2-functie vergt soms wat extra codeerwerk in SQL Server:

```
(D) STRIP(c1[,{B,L,T}[,x])=> (S) combinatie van LTRIM(c1) en RTRIM(c1)
```

Datum- en tijdmanipulatie-functies

Er zijn belangrijke verschillen tussen DB2 en SQL Server wat scalaire datum- en tijdsmanupilatie-functies betreft, maar deze zijn minder groot dan de verschillen tussen DB2 en Oracle.

Merk op dat, afhankelijk van de functie het resultaat een numerieke waarde, een datum of een *character string* kan zijn. Input kan ook variëren: een datum *(d)*, een tijd *(t)*, een timestamp *(ts)*, een *character string* die een geldige representatie van een datum is *(c)* of een numerieke waarde tussen 1 en 3652059 *(n)* die zal geïnterpreteerd worden als het aantal dagen sinds 1 januari van het jaar 1.

In wat volgt geven we voor elke individuele DB2-functie een specifiek SQL Server-alternatief. We doen een poging e.e.a. logisch te groeperen.

```
(D) CURRENT TIMESTAMP       => (S) GETDATE()
(D) CURRENT TIMESTAMP       => (S) CURRENT_TIMESTAMP
```

Werken met onderdelen van datums/timestamps zowel in DB2 als in SQL Server:

```
DAY(d/ts), MONTH(d/ts), YEAR(d/ts)
```

Werken met de overige onderdelen van datums/timestamps in SQL Server:

```
(D) QUARTER(d/ts)          => (S) DATEPART(QUARTER, d/ts)
(D) WEEK(d/ts)             => (S) DATEPART(WEEK, d/ts)
(D) DAYOFWEEK(d/ts)        => (S) DATEPART(DAYOFWEEK, d/ts)
(D) DAYOFYEAR(d/ts)        => (S) DATEPART(DAYOFYEAR, d/ts)
(D) HOUR(t/ts)             => (S) DATEPART(HOUR, t/ts)
(D) MINUTE(t/ts)           => (S) DATEPART(MINUTE, t/ts)
(D) SECOND(t/ts)           => (S) DATEPART(SECOND, t/ts)
(D) MICROSECOND(ts)        => (S) DATEPART(MICROSECOND, ts)
```

Werken met extra onderdelen van datums/timestamps in SQL Server:

```
                           => (S) DATEPART(MILLISECOND, ts)
                           => (S) DATEPART(NANOSECOND, ts)
```

Rekenen met datums en tijd:

```
(D) d/ts + n YEARS         => (S) DATEADD(YEAR, n, d/ts)
(D) d/ts + n MONTHS        => (S) DATEADD(MONTH, n, d/ts)
(D) d/ts + n DAYS          => (S) DATEADD(DAY, n, d/ts)
...
```

Conversiefuncties

Deze functies nemen het argument en vormen dit om tot een ander datatype. De argumenten kunnen numeriek (*n*) of alfanumeriek (*c*) of willekeurig (*x*) zijn. Het kan een datum (*d*), een tijd (*t*) of een timestamp (*ts*) zijn. Soms wordt het gewenste datatype (*type*) expliciet meegegeven, soms het gewenste formaat (*fmt*) en soms ook een lengte (*len*). Het resultaat van de functie hangt af van de functie zelf.

Volgende standaard-SQL conversiefunctie wordt zowel door DB2 als door SQL Server (en trouwens ook door Oracle en MySQL) ter beschikking gesteld:

```
CAST (x AS type[(argumenten*)])
```

DB2 10 for z/OS: enkele opvallende nieuwigheden

Peter Vanroose (ABIS)

Een volledig overzicht geven van alle nieuwe mogelijkheden van versie 10 is onbegonnen werk binnen het bestek van Exploring DB2.

Met de belangrijkste nieuwe mogelijkheid hebt u trouwens al kennis gemaakt in de artikelreeks “Temporele data”. Inderdaad: waar DB2 9 de XML-versie was en DB2 v8 de 64-bit versie (en de unicode-versie), is DB2 10 in de eerste plaats de versie die tijd-gerelateerde operaties een heel stuk flexibeler maakt. Daarnaast zijn er nog enkele (minder opvallende) nieuwigheden die we de revue laten passeren.

Tijd-gerelateerde zaken

Behalve de nieuwe SQL-syntax (DDL en DML) voor tijdstip-gebaseerde data-versioning (system-time en business-time), zoals besproken in de drie bijdragen hierover in Exploring DB2, zijn er nog twee belangrijke nieuwe SQL-mogelijkheden:

- een uitbreiding van datatype TIMESTAMP, nl. `TIMESTAMP WITH TIME ZONE` dat (zoals kan verwacht worden) bovenop de date/time /microseconds-componenten (4+3+3=10 bytes) ook bijhoudt voor welke tijdszone deze indicatie geldt. De tijdszone wordt gespecifieerd als het aantal uren en minuten vóór of na UTC; voor de Benelux is dat altijd +1 (winter) en +2 (zomer): we lopen namelijk 1 of 2 uur voor op het winteruur van Greenwich. De tijdszone wordt intern opgeslagen in twee bytes (= 2 decimalen voor het uur en een teken-bit, en twee decimalen voor de minuten), met (momenteel) mogelijke waarden tussen -12:30 en +14:00.

- daarnaast laat het datatype TIMESTAMP nu toe, de granulariteit van de seconden-fractie (voorheen dus enkel milliseconden) zelf te beslissen: tussen 0 en 12 decimalen. De default is nog steeds 6 decimalen. De totale (binaire) grootte van een timestamp-veld wordt dus tussen de 7 en 13 bytes (zonder tijdszone) of 9 tot 15 bytes (met tijdszone). In tekstuele vorm (b.v. als output van een SELECT-statement) zal de lengte tussen 19 en 32 characters lang zijn. Vroeger was dat altijd 26 characters. Indien ook een tijdszone-indicatie aanwezig is, kan deze tekstuele lengte oplopen tot 38 characters; b.v.: `2011-12-25-14.15.00+1:00` is kerstdag kwart over drie (zonder seconden-fracties) in de Benelux-tijdszone, dus kwart over twee UTC.

De “default”-tijdszone (die DB2 gebruikt wanneer de twee tijdszone-bytes niet aanwezig zijn), wordt gevonden in de nieuwe zParm “`IMPLICIT_TIMEZONE`”. Let dus op wanneer uw systeem fysiek niet in onze tijdszone staat (en anders ook trouwens): deze waarde zou wel

eens verschillend van +1:00 (of +2:00) kunnen zijn! Het is overigens eenvoudig om de waarde van deze zParm te achterhalen: kijk gewoon in het `CURRENT TIMEZONE` special register:

```
SELECT CURRENT TIMEZONE FROM sysibm.sysdummy1.
```

Een applicatie heeft de mogelijkheid om z'n eigen (sessie) tijdszone in te stellen m.b.v. (b.v.) `SET SESSION TIMEZONE = '-5:00'` (tijdszone New York) en op te vragen via special register `SESSION TIMEZONE`.

Een andere tijd-gerelateerde nieuwigheid bestaat al sinds versie 9, nl. de mogelijkheid om via SQL SELECT van elke rij uit om het even welke tabel op te vragen wat z'n laatste wijzigings-tijdstip is:

```
SELECT t.*, ROW CHANGE TIMESTAMP FOR t  FROM my_table t   WHERE ...
```

Eigenlijk geeft ROW CHANGE TIMESTAMP niet gegarandeerd de rij-specifieke laatste wijzigingsdatum, maar meestal slechts het laatste wijzigings-tijdstip van de fysieke page van de rij. Enkel wanneer de tabel een "row change timestamp"-kolom heeft, zal deze waarde rij-specifiek zijn. Voor rijen die nog nooit werden ge-updated wordt de insert-tijd getoond.

OLAP-nieuwigheden

Eindelijk - na herhaalde aankondigingen – ondersteunt DB2 for z/OS nu ook de zgn. "windowing"-specificaties bij groeperingsfuncties, die dus toelaten om b.v. een "moving average" te laten berekenen. Een voorbeeldje: stel dat we, op basis van omzetcijfers per dag, zonder ontbrekende gegevens, een "maandomzet-tendens" willen tonen. Voor elke dag willen we dus b.v. de gemiddelde waarde tonen van de 30 dagen "omheen" (d.w.z. van 15 voor tot 14 na) die datum. Dit kan met volgende standaard SQL, nu dus ook op z/OS:

```
SELECT  datum, AVG(omzet) OVER (ORDER BY datum
                                ROWS BETWEEN 15 PRECEDING AND 14 FOLLOWING)
FROM    my_table
ORDER BY datum
```

Een gelijkaardige syntax is eveneens mogelijk voor SUM, COUNT, MIN, MAX, STDDEV, VARIANCE, COVARIANCE, en CORRELATION.

ORDER BY

"Order by 2" heeft altijd al de data gesorteerd op de tweede kolom, en dat is niet anders in versie 10. Maar "Order by +2" (of een andere integer-expressie die de waarde 2 oplevert) sorteert de data sinds versie 10 niet langer op de tweede kolom, maar alsof de "Order by" niet aanwezig was! (Eigenlijk dus op de numerieke expressie "2", die uiteraard voor alle rijen dezelfde is, dus niet lijkt te sorteren.)

XML

In de nummers 1, 2 en 4 van jaargang 5 van Exploring DB2 (zie *http://www.abis.be/html/nlExploreDB2.html*) hadden we het over geïntegreerde ondersteuning van XML sinds versie 9 van DB2 (voor z/OS en LUW). In versie 10 zijn hier nog enkele zaken bij gekomen:

XML schema-validatie: het is nu mogelijk om aan een XML-kolom, als een soort check constraint, een afdwingbaar XML-schema de koppelen; bijvoorbeeld:

```
CREATE TABLE tutpersons(
 pno     INT NOT NULL PRIMARY KEY,
 comment XML (XMLSCHEMA URI 'http://www.abis.be/XMLschemas/perscomment'
              LOCATION 'http://www.abis.be/XMLschemas/perscomment.xsd')  )
```

INSERTs en UPDATEs zullen geweigerd worden als de XML-kolom een XML-document is dat niet beantwoordt aan het XML-schema.

Bovendien verifieert het CHECK DATA-utility nu ook XML-kolommen, zowel wat betreft well-formedness als wat betreft het valideren met één van de eventueel opgegeven XML-schema's.

XML als parameter: XML kan nu als argument doorgegeven worden naar stored procedures (ten minste: indien geschreven in SQL PL) en scalaire functies. Bovendien ondersteunt SQL PL nu ook het datatype XML voor b.v. variabelen.

De allerbelangrijkste nieuwigheid wat XML betreft is echter de mogelijkheid om slechts *een deel* van een XML-document te wijzigen. Bijvoorbeeld: stel dat we voor de persoon met `pno=10` in de XML-kolom "`comment`" achteraan een "`<ptel>`"-node willen toevoegen met het telefoonnummer van die persoon:

```
UPDATE tutpersons
   SET comment = XMLMODIFY('do insert node $tel as last into /',
                           XMLPARSE(DOCUMENT '<ptel>016/123456</ptel>') AS "tel")
WHERE pno = 10
```

De nieuwe scalaire functie XMLMODIFY zal enkel die ene node toevoegen aan het XML-document, en de rest van het document ongemoeid laten; vooral in termen van performance is dit een heel grote stap vooruit t.o.v. wat in versie 9 mogelijk was.

Zoals uit vorig voorbeeld blijkt, ondersteunt DB2 10 for z/OS nu ook *XQuery*. Dit geldt niet alleen voor de nieuwe XMLMODIFY-functie maar overal waar voorheen een XPath-expressie mogelijk was, b.v. in de functie XMLQUERY. Als gevolg hiervan worden nu ook de meeste tijd- en datum-gerelateerde aspecten van XML ondersteund, zoals de XML-datatypes `xs:dateTime`, `xs:time`, `xs:date` en `xs:duration`, i.h.b. met de mogelijkheid om hierop indexen te definiëren van datatypes TIME, DATE of TIMESTAMP. Waarmee we terug bij aandachtspunt nummer één van versie 10 zijn: datum- en tijd-aspecten.

Tenslotte

Daarnaast zijn er uiteraard talloze kleine verbeteringen, i.h.b. wat betreft performance en data-beschikbaarheid. Meer details hierover vindt u uiteraard terug in de IBM-documentatie, i.h.b. in het document GC19-2985: "DB2 10 for z/OS: What's New?"

Geïnteresseerd in meer details omtrent één van de genoemde nieuwigheden, of andere aspecten van DB2 10? Laat het ons weten, dan schrijven we erover in één van onze volgende nummers!

CURSUSPLANNING JAN. - MEI 2012

DB2 for z/OS, een totaaloverzicht	2125 EUR	13.02(L), 07.05(W)
DB2 UDB for LUW, totaaloverzicht	2125 EUR	09.01(L), 07.05(W)
RDBMS-concepten	395 EUR	09.01(L), 13.02(L), 20.02(W), 02.04(L)
Basiskennis SQL	395 EUR	10.01(L), 14.02(L), 21.02(W), 03.04(L)
DB2 for z/OS basiscursus	1335 EUR	15.02(L), 09.05(W)
DB2 UDB for LUW basiscursus	1335 EUR	11.01(L), 14.05(W)
SQL-QMF voor eindgebruikers	1335 EUR	op aanvraag
SQL workshop	840 EUR	01.03(L), 05.03(W), 16.04(L)
SQL voor gevorderden	470 EUR	23.01(W), 15.03(L), 18.04(L)
Software-ontwikkeling met SQL PL voor DB2	940 EUR	29.03(L)
DB2 triggers, stored procedures, UDFs	470 EUR	16.03(L)
DB2 for z/OS programmeren voor gevorderden	940 EUR	05.03(L)
DB2 for z/OS: SQL performance	1410 EUR	19.03(L)
XML in DB2	470 EUR	27.03(W), 01.06(L)
DB2 for z/OS database administratie	1980 EUR	23.04(L)
DB2 for z/OS installation and migration	825 GBP	23.02(UK), 24.05(UK)
DB2 for z/OS data recovery	825 GBP	21.02(UK), 22.05(UK)
DB2 for z/OS systems performance and tuning	825 GBP	27.02(UK), 28.05(UK)
DB2 LUW DBA - Kernvaardigheden	1880 EUR	06.02(L)
DB2 10 upgrade	470 EUR	op aanvraag
Data warehouse concepten	470 EUR	12.03(L), 21.05(W)
SQL voor BI	470 EUR	19.04(L)
Exploring DB2 - live!		op aanvraag

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
http://www.abis.be/
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Zaagmolenlaan 4
NL-3440 AC Woerden
Tel. 0348-413663
Fax +32-16-245639
http://www.abis.be/
training@abis.be