ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 6 - NUMMER 1, MAART 2009

OPEN CURSOR

Het centrale thema van het vorige nummer was XML; in dit nummer besteden we aandacht aan Data Warehouses (DW) en Business Intelligence (BI).
We hebben dit thema uitgekozen, omdat in de wereld waarin we vandaag leven, informatie, en controle van die informatie, nog steeds aan belang wint. We willen met name stilstaan bij de technische functionaliteiten die DB2 voor z/OS vandaag biedt. Maar toch ook zo aangebracht dat niet-DB2-specialisten een aantal aangehaalde concepten ongetwijfeld zullen kunnen vertalen naar hun eigen RDBMS-platform toe.
Uiteraard beperken we ons tot een aantal database functionaliteiten, eigen aan dit onderwerp. Samen met relevante artikels reeds in vorige nummers verschenen, vormen ze een interessant geheel; een bloemlezing waar u bij uw activiteiten in het DW-domein ongetwijfeld nuttig gebruik van zult kunnen maken!

Het ABIS DB2-team.

IN DIT NUMMER:

- Een overzicht van DB2-functionaliteiten rond DW, besproken in vorige nummers, passeert de revue in *Data Warehousing en DB2 for z/OS - A Recap.*
- Het combineren van dimensie- en facttabellen in *Over star schema joins...*
- Een overzicht van de mogelijkheden geboden door OLAP-functies, relatief nieuw in DB2, in *Haal meer uit je database met OLAP SQL.*
- *DB2 for z/OS en de Dynamic Statement Cache* geeft aan wat de impact kan zijn van deze vaak onderkende functionaliteit voor DW-applicaties.

1

CLOSE CURSOR

In een volgend nummer laten we het 'thema'-idee even los. We besteden o.a. aandacht aan XQuery support in DB2, en staan stil bij de nieuwe EXPLAIN mogelijkheden in v8 en v9.

Data Warehousing en DB2 for z/OS - A Recap

Kris Van Thillo (ABIS)

Data Warehouses (DW) en, in een bredere context, Business Intelligence (BI) winnen nog steeds aan belang. Ook DB2 for z/OS bevat steeds meer faciliteiten die speciaal binnen de context van DWs extra relevant zijn. We herhalen in het kort een aantal van deze functies die reeds in vorige nummers van Exploring DB2 werden aangehaald.

- DWs bestaan typisch uit omvangrijke tabellen, met een relatief groot aantal indexen. Het spreekt immers voor zich dat in deze vaak 'read-only' omgeving ad-hoc queries en rapporten enkel aan de hand van indexen efficiënt kunnen worden ondersteund. In deze context is compressie dan ook vaak interessant. Zie *Index compression*, *Table space compression*, in 'Datacompressie in DB2: wat en hoe?', Exploring DB2, jaargang 5, nummer 3. Real-time statistics zijn dan bijvoorbeeld nuttig om na te gaan of deze indexen dan ook inderdaad allemaal efficiënt worden gebruikt. Zie *Real-time statistics*, en het belang ervan op index gebruik, in 'Realtime statistics in DB2 for z/OS', Exploring DB2, jaargang 3.
- Het indexeren van expressies biedt vaak aanzienlijk voordelen in de context van end-user, ad-hoc querying. Zie *Indexes op expressies*, in 'Index by Expression in DB2 9 for z/OS', Exploring DB2, jaargang 5, nummer 2.
- Het beschikbaar houden van DWs enerzijds, en de hele problematiek van 'datapublicatie' anderzijds, is een complexe materie. Hoe bijvoorbeeld nieuwe data laden met minimale impact op gebruikers van het DW? En hoe er voor zorgen dat nieuw geladen data in het DW op een tijdsconsistente wijze beschikbaar wordt binnen het DW? 'Clone tables' kunnen hier zekere bij helpen. Zie *Clones*, in 'Clone tables', Exploring DB2, jaargang 5, nummer 4.
- Omvangrijke tabellen stellen vaak uitzonderlijke eisen naar beheer en onderhoud toe. En ook met specifieke eisen eigen aan DWs moet worden rekening gehouden: archivering, data publicatie,... Data partitioning speelt hierbij een belangrijke rol. Zie hiertoe bijvoorbeeld *Table space partitioning*, in 'Tabelpartitionering in DB2 versie 8', Exploring DB2, jaargang 4, nummer 4.
- Materialized query tables (MQTs) vormen één van de belangrijkste topics uit deze lijst, zeker bekeken vanuit het standpunt performance. We hebben nu immers de mogelijkheid om echte tabellen aan het DW toe te voegen, die, net als indexes, door de optimizer gebruikt kunnen worden om de query snel en efficiënt uit te voeren. Ideaal voor aggregates ('summeries'), afgeleide berekeningen,... Ook het 'Query Rewrite' feature kan hier gebruik van maken. Zie *Materialized query tables (MQT)*, in 'Over MQTs & MVs - waar het echt om te doen is!', in Exploring DB2, jaargang 5, nummer 1.

Een eerste link naar BI werd tevens gelegd in het artikel 'Cognos, an IBM Company (deel 1)', Exploring DB2, jaargang 5, nummer 3.

Over star schema joins...

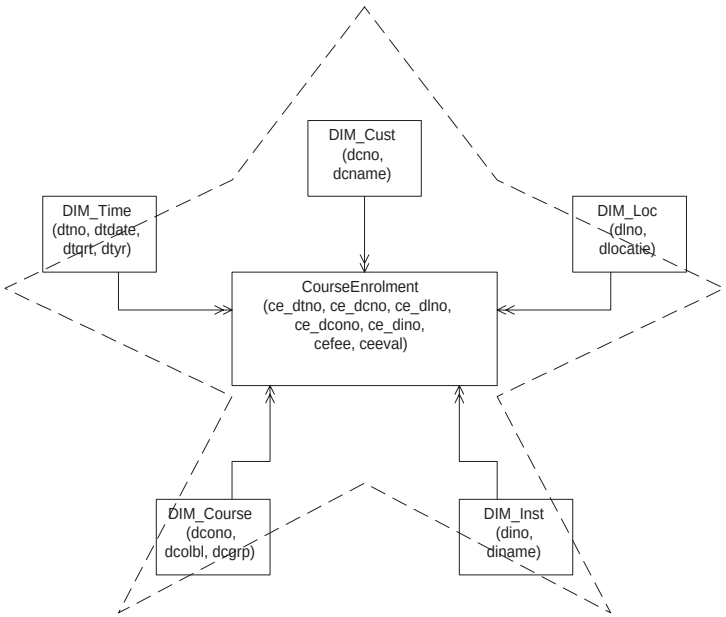
Kris Van Thillo (ABIS)

Eénieder die met DWs te maken krijgt, zal ongetwijfeld op één of ander moment in aanraking komen met star schema's en, vanuit fysiek database standpunt, met star joins. En daar willen we het in dit artikel in hoofdlijnen over hebben: star schema's en star joins, specifiek in een DB2 for z/OS omgeving.

Definities.

Een 'star schema' - typisch voor logisch DW design - bestaat uit één centrale 'fact'- of 'event'-tabel, en een aantal 'dimensie'-tabellen, die de context van de 'fact'- of de 'event'-tabel beschrijven. Onderstaande figuur biedt een voorbeeld: CourseEnrolment is de facttabel; de 5 'DIM_'-tabellen zijn de tabellen die de achtergrond of context van een fact beschrijven. Concreet, bijvoorbeeld, geven ze aan voor één bepaalde cursusinschrijving, voor welke periode deze werd gemaakt, wie de klant is, waar de cursus plaatsvindt, over welke cursus het gaat, en wie de instructeur is. Wat we vastleggen voor de fact is bijvoorbeeld de prijs betaald voor de opleiding, en evaluatie achteraf door de deelnemer toegekend. Een 'snowflake schema' gaat nog één stap verder - de dimensietabellen zelf zijn extra genormaliseerd!

Figuur - Een eenvoudig Star Schema



Merk in bovenstaande figuur bijvoorbeeld op dat DIM_Time en Dim_Course entiteiten elk genormaliseerd kunnen worden. Zo bevat DIM_Time zich herhalende waarden voor kwartaal en jaar - makkelijk af te splitsen in een tweetal tabellen; en ook DIM_Course kan worden genormaliseerd in een 'cursus' (dcono, dcolbl) en een 'cursusgroep' (dcgrp) gerelateerde tabelstructuur. Het aldus normailseren van een 'dimensie' in meerdere tabelstructuren geeft aanleiding tot het ontstaan van 'snowflakes' of sneeuwvlokjes.

Hoe een typische star schema query er dan uit kan zien, wordt in onderstaand voorbeeld aangegeven, aangemaakt op basis van het star schema op vorige pagina aangekaart.

Voorbeeld - star schema query

```
select  sum (cefee), avg (cefee)
from    DIM_Time, DIM_Cust, DIM_Loc, DIM_Course, DIM_Inst, CourseEnrolment
where   DIM_Time.dtno       = CourseEnrolment.ce_dtno
   and  DIM_Cust.dcno       = CourseEnrolment.ce_dcno
   and  DIM_Loc.dlno        = CourseEnrolment.ce_dlno
   and  DIM_Course.dcono    = CourseEnrolment.ce_dcono
   and  DIM_Inst.dino       = CourseEnrolment.ce_dino
   and  DIM_Time.dtyr       = 2008
   and  DIM_Cust.dcname     = 'KLM'
   and  DIM_Loc.dlocatie    = 'ABIS Woerden'
   and  DIM_Course.dcolbl   = 'QMF'
```

De eerste vijf condities in dit voorbeeld beschrijven de join tussen de facttabel en de verschillende dimensietabellen; de volgende condities beschrijven de voorwaarden opgelegd aan de waarden in de dimensietabellen.

Uitdagingen.

Elk RDBMS - DB2 for z/OS, DB2 for LUW, Oracle - heeft in essentie dezelfde uitdagingen. Deze zijn in wat volgt samen te vatten.

- identificatie van de query als zijnde een star schema query: waarom is dit geen gewone, standaard-join?
- de keuze van de facttabel: welke tabel is de significant grootste tabel in de join?
- de keuze van de meest optimale join-volgorde: welke tabellen lees ik eerst/laatst? En waarom?

Dit is geen gewone join omdat typisch de dimensietabellen onderling geen verbinding hebben; de relaties lopen enkel via de facttabel; en filtering is enkel mogelijk via de dimensietabellen, waarop concreet de (filter)-voorwaarden zijn gedefinieerd. Uiteraard willen we de grootste tabel - de facttabel - zo laat mogelijk lezen; hoe minder rijen te verwerken, hoe beter. Identificatie van de facttabel is voor de optimizer niet steeds eenvoudig - statistieken kunnen helpen. Uitdaging is natuurlijk dat bepaalde dimensietabellen groot kunnen zijn... En tot slot: de join-volgorde - hoe omgaan met het feit dat inderdaad tussen de dimensietabellen geen relaties bestaan?

In wat volgt komen we op deze punten terug.

Een star join?

Of we al dan niet te maken hebben met een star join is niet steeds makkelijk uit te maken. Het is in de praktijk mogelijk dat DB2 for z/OS beslist het star join algoritme toe te passen voor standaard-joins; of voor star schema joins dit juist niet te doen - met niet optimale uitvoerplannen tot gevolg. Zoals steeds moet het door DB2 weerhouden accesspad ook in deze situatie kritisch worden geëvalueerd.

De volgende dynamische systeemparameters (DSNZPARMs) moeten geactiveerd zijn om star join processing mogelijk te maken (andere parameters hebben eveneens een invloed, e.g. SJMISSKY):

- STARJOIN: disable, 1, enable, [$n > 1$]. Parameter die aangeeft dat star join processing niet beschikbaar is; beschikbaar is met de grootste tabel als de facttabel bepaald op basis van cardinaliteiten; beschikbaar is als de cardinaliteit van de facttabel minimaal 25 keer de cardinaliteit van de grootste dimensietabel is; beschikbaar is als de cardinaliteit van de facttabel minimaal *n* keer de cardinaliteit van de grootste dimensietabel is.
- SJTABLES: geeft aan hoeveel tables in de star join minimaal moeten betrokken zijn om van een star join te kunnen spreken (totaal aantal tabellen - fact, dimensies, ...).
- MXQBCE: beperkt het aantal join alternatieven door de optimizer in rekening gebracht gedurende de optimizatiefase.

Ook de star join query zelf moet aan een aantal voorwaarden voldoen. Een niet-exhausieve lijst voorwaarden:

- elke query bevat ten minste 2 dimensietabellen - denk aan SJTABLES;
- alle join-condities liggen tussen de facttabel en de respectieve dimensietabellen, en niet tussen dimensietabellen onderling;
- enkel equi-joins zijn toegelaten binnen de scope van star joins, en het datatype en de lengte van de kolommen aan beide zijden van de join-conditie moeten identiek zijn;
- outer joins worden niet ondersteund;
- gecorreleerde subqueries overheen dimensies worden niet ondersteund;
- condities op de dimensietabellen zijn dusdanig dat van zodra één dimensie als 'false' wordt geëvalueerd, de volledige factrij niet hoeft te worden weerhouden.

Indexen.

Om als star join te worden weerhouden dienen een aantal indexen te worden aangemaakt. Meer specifiek:

- maak een multi-kolom index aan op alle dimensie-key kolommen in de facttabel (ce_dtno, ce_dcno, ce_dcono, ce_dino in ons voorbeeld). Ga er vanuit de meest selectieve kolommen in deze index vooraan te plaatsen; verwijder (of plaats achteraan) dimensiekolommen die zelden/nooit in queries worden gebruikt;

- de multi-kolom index wordt typisch aangemaakt als een clustering index;
- maak indien relevant en nuttig een index aan op de dimensietabellen om toegang (filtering) op de dimensietabellen te vergemakkelijken (naast de traditionele key indexes op deze dimensietabellen).

Indexen kunnen al dan niet compressed zijn.

Rol van de DB2 optimizer.

De rol van de DB2 optimizer blijft ook in deze context ongewijzigd: keuzes worden gemaakt op basis van beschikbaarheid van indexen, clustering factors, selectiviteit van rijen, ...

Keuze van de facttabel door de optimizer is cruciaal: het gaat om de grootste tabel, waarvan we het lezen zo lang mogelijk willen uitstellen! De keuze van de facttabel wordt bepaald door:

- de systeemparameters boven reeds aangehaald;
- de relatieve omvang van (en tussen) de tabellen: cardinaliteiten;
- de aanwezigheid van kolommen in de star join waarop unieke indexen werden aangemaakt (dan wel een multi-kolom index) geeft aan welke tabel het label fact dan wel dimensie dient te ontvangen.

Indien het gebruik van de star join werd geactiveerd, zal DB2 for z/OS op basis van selectiviteit van de multi-kolom index op de facttabel een cartesiaans product maken tussen dimensietabellen, na toepassing van de equi-join filters. Of nog, op basis van de selectiviteit van de multi-kolom index op de fact, wordt bepaald welke dimensietabellen na filtering van relevante rijen het onderdeel zullen uitmaken van het cartesiaans product. Het resultaat van dit product wordt als het ware een virtuele tabel, die in een volgende stap met de facttabel zal worden gejoined. Deze virtuele tabel wordt mogelijk gematerialiseerd in een workfile (sparse index access).

Concreet dus, in stappen:

- aanmaken via een cartesiaans product van een virtuele dimensiestabel - keuze van de dimensietabellen die zullen worden gecombineerd o.a. op basis van de cardinaliteiten en selectiviteiten van de multi-kolom index op de facttabel;
- join van de virtuele dimensietabel met de facttabel, aan de hand van de multi-kolom index op de facttabel.

Hoe de dimensietabellen worden gelezen wordt opnieuw autonoom door de optimizer bepaald: via index access, table space scan, ... Indien een table space scan wordt weerhouden kan het nuttig zijn na te gaan of de betreffende dimensietabel voldoende (en correct) van indexen werd voorzien. De facttabel wordt typisch verwekt op basis van de multi-kolom index.

De plantabel.

De inhoud van de plantabel werd aangepast om informatie betreffende de star join te kunnen opnemen. De belangrijkste wijziging betreft de kolom JOIN_TYPE: een waarde 'S' geeft aan dat de optimizer voor een star join optimisatie heeft gekozen. De mogelijke materialisatie in een workfile (`PRIMARYACCESS_TYPE = 'T'`), en het gebruik van de multi-kolom index op de facttabel zou eveneens duidelijk moeten aangegeven zijn.

In het kort - 1: de pair-wise join.

Indien men de multi-kolom index op de facttabel vervangt door één individuele index per dimensie-key (die dus overeenkomst met een dimensietabel), kan DB2 opteren voor een pair-wise join. Concreet zal DB2 in parallel per dimensie de dimensietabel combineren met de index op de facttabel; dit resulteert in een RID-lijst per dimensie, die in een volgende stap worden gecombineerd. De RID pool, of indien deze onvoldoende groot is, een workfile kan hiertoe worden aangewend.

Als op de dimensiekolommen van de facttabel zowel een multi-kolom index als individuele indexen per key worden aangemaakt; zal DB2 o.a. de star join en de pair-wise join als valabele alternatieven evalueren, en mogelijk, één van beide als optimaal weerhouden.

In het kort - 2: data caching.

Indien data caching werd geactiveerd, kunnen star joins (net als andere join-types in DB2 v9) hiervan gebruik maken. Gevolg is dat data, eens opgehaald uit een workfile, in deze cache wordt bijgehouden; en aldus onmiddellijk beschikbaar is voor hergebruik op een later moment.

Binnenkort op onze cursusplanning:

Data warehouse concepten, Leuven, 27 maart 2009.

Data warehouse concepten, Woerden, 4 mei 2009.

Exploring DB2 Live! besteedt ook aandacht aan een aantal van deze topics:

Partitioning in DB2 for z/OS: best practices, Leuven, 26 maart 2009.

Dynamic SQL and the Dynamic Statement Cache, Leuven, 8 juni 2009.

Haal meer uit je database met OLAP SQL

Patrick Van Damme (ABIS)

Voor OLAP heb je niet altijd geavanceerde tools nodig; een aantal basis data mining strategieën kan je gewoon zelf toepassen: in SQL laten GROUP BY extensies en analytische functies je immers toe andere (in)zichten te krijgen met betrekking tot je business data. We overlopen de voornaamste en laten bewust gespecialiseerde statistische, distributieve of lineair regressieve functies e.d. buiten beschouwing.

Alle voorbeelden gaan uit van een tabel ORDERS, met voor elke bestelling een bestelnummer, een product-id, het aantal bestelde producten en de streek waar de bestelling geplaatst is:

```
CREATE TABLE orders(ordnr     SMALLINT  NOT NULL,
                    prodid    VARCHAR(4)        ,
                    nrprod    SMALLINT          ,
                    region    VARCHAR(10)       )
```

ORDNR	PRODID	NRPROD	REGION
1	3214	8	WEST
2	5927	3	EAST
3	3214	4	EAST
4	3214	1	WEST
5	7489	1	NORTH
6	5927	10	WEST
6	3214	1	WEST

GROUP BY extensies: ROLLUP - CUBE - GROUPING SETS

De functionaliteit van alle GROUP BY uitbreidingen is van per dimensie te groeperen en (sub)totalen weer te geven.

De ROLLUP extensie van GROUP BY voegt rijen met subtotalen en een algemeen totaal toe aan de gewone rijen in de resultset, op basis van de opgegeven groeperingen. De verzameling van alle waarden wordt voorgesteld door null: CASE laat toe om een betekenisvolle tekst aan die cellen toe te kennen.

```
SELECT   (CASE WHEN GROUPING(region) = 1 THEN 'ALL' ELSE region END) AS region,
        (CASE WHEN GROUPING(prodid) = 1 THEN 'TOT' ELSE prodid END) AS product,
         SUM(nrprod) as "NUMBER ORDERED"
FROM     orders
GROUP BY ROLLUP (region,prodid)
```

REGION	PRODUCT	NUMBER ORDERED
EAST	3214	4
EAST	5927	3
EAST	TOT	7
WEST	3214	10
WEST	5927	10
WEST	TOT	20
NORTH	7489	1
NORTH	TOT	1
ALL	TOT	28

De GROUPING functie die in de query gebruikt is, dient voornamelijk om de 'super-aggregate rows' te identificeren: voor die rijen heeft GROUPING de waarde 1 en zoniet altijd 0. Alternatief kan men uiteraard ook `COALESCE(region, 'ALL')` gebruiken, zoals verderop gebeurd is.

ROLLUP (region,prodid) is in feite een shortcut voor GROUPING SETS ((region, prodid), (region),()), dat precies hetzelfde resultaat oplevert: namelijk één rij per groepering op (region, prodid), zoals hier de eerste 2 rijen, een subtotaal per groepering op (region), wat een rollup (een samennemen) is op prodid, zoals de 3de rij, en een algemeen totaal voor de gehele tabel, gegroepeerd op (), dus zonder parameter, zoals voorgesteld in de laatste rij en wat een rollup is op region.

De CUBE extensie van GROUP BY is een combinatie van ROLLUP extensies.

```
SELECT    COALESCE(region,'ALL')  AS region,
          COALESCE(prodid,'TOT')  AS product,
          SUM(nrprod) as "NUMBER ORDERED"
FROM      orders
GROUP BY  CUBE (region,prodid)
```

REGION	PRODUCT	NUMBER ORDERED
ALL	TOT	28
ALL	3214	14
ALL	5927	13
ALL	7489	1
EAST	TOT	7
EAST	3214	4
EAST	5927	3
WEST	TOT	20
WEST	3214	10
WEST	5927	10
NORTH	TOT	1
NORTH	7489	1

CUBE (region,prodid) is een shortcut voor ROLLUP(prodid), ROLLUP(region). Je kan inderdaad de GROUP BY extensies met elkaar combineren. Het resultaat in dit geval is hetzelfde als dat van GROUPING SETS ((region,prodid),(region),(prodid),()), hetgeen duidelijk aantoont dat CUBE de unie berekent van alle mogelijke (namelijk 2 tot de n-de) groeperingen.

Dergelijke combinaties van GROUP BY extensies bieden vele mogelijkheden. We illustreren dit aan de hand van een voorbeeld dat betrekking heeft op hiërarchie: elk product-id behoort tot een bepaalde categorie (food of non-food). Daarvoor hebben we een tweede tabel nodig.

```
CREATE TABLE products (prodnr    VARCHAR(4)   ,
                       cat       VARCHAR(8)   )
```

PRODID	CAT
3214	food
5927	non-food
7489	non-food

```
SELECT   COALESCE(o.region, 'ALL')  AS region,    p.cat,
         COALESCE(o.prodid, 'TOT')  AS product,
         SUM(o.nrprod)  AS "NUMBER ORDERED"
FROM     orders o INNER JOIN products p ON o.prodid = p.prodnr
GROUP BY ROLLUP (o.region), ROLLUP(p.cat,o.prodid)
```

REGION	CAT	PRODUCT	NUMBER ORDERED
ALL	food	3214	14
ALL	food	TOT	14
ALL	non-food	5927	13
ALL	non-food	7489	1
ALL	non-food	TOT	14
ALL		TOT	28
EAST	food	3214	4
EAST	food	TOT	4
EAST	non-food	5927	3
EAST	non-food	TOT	3
EAST		TOT	7
WEST	food	3214	10
WEST	food	TOT	10
WEST	non-food	5927	10
WEST	non-food	TOT	10
WEST		TOT	20
NORTH	non-food	7489	1
NORTH	non-food	TOT	1
NORTH		TOT	1

Het eindresultaat is het cartesiaans product van ROLLUP (region) (of GROUPING SETS ((region),())) met ROLLUP(cat,prodid) (of GROUPING SETS ((cat,prodid),(cat),())), en komt neer op GROUPING SETS ((region,cat,prodid),(region,cat),(region),(cat,prodid),(cat),()).

Analytische functies

De grote meerwaarde die analytische functies bieden is dat ze in één tabel, per rij, zowel groepsgegevens (aggregate) als detailgegevens (scalar) kunnen weergeven. De syntax is:

```
FUNCTIE_NAAM(<argument>,<argument>,...) OVER
    (<PARTITION BY clausule> <ORDER BY clausule> <windowing clausule>)
```

De functie staat in de SELECT clausule en wordt dus uitgevoerd op de resultaatset na de WHERE en GROUP BY (en HAVING) clausules.

Het sleutelwoord OVER, waarmee je overigens onmiddellijk een functie kan herkennen als analytisch, scheidt de functienaam van het bereik (slice) van rijen/data waar de functie ‘over’ gespreid wordt.

De drie clausules volgend op de OVER in de functie syntax bepalen de "window", de weergave van de functie. De haakjes na OVER kunnen leeg blijven ofwel een partitioneringsclausule en/of een ordeningsclausule (met als eventuele uitbreiding van die ordening een windowing clausule).

De operatie die dient uitgevoerd te worden op de opgegeven 'slice' van data is gespecificeerd in de FUNCTIE_NAAM. We zullen vooral dieper ingaan op 2 veelgebruikte soorten functies: de windowing aggregate functies, zoals SUM, AVG, MAX, MIN, COUNT, FIRST_VALUE of LAST_VALUE (die één resultaatwaarde geven als berekening op de groep) en ranking functies, zoals RANK, DENSE_RANK of ROW_NUMBER (die de positie aangeven binnen de groep).

1. Geef mij detail- en geheelgegevens van de verschillende bestellingen, gegroepeerd per product: PARTITION BY

De PARTITION BY clausule laat toe per kolom te bepalen op welke groep van rijen een functie moet toegepast worden. Partitionering zorgt voor een onderverdeling van de hele kolom in volledig zelfstandige windows, absolute partities (b.v. is prodid de dimensie, dan vormen alle rijen met hetzelfde prodid een window), waarbij, in tegenstelling tot een GROUP BY, alle rijen worden getoond maar voor berekeningen worden gegroepeerd. Is er geen partitioneringsclausule in de functie, dan is er uiteraard maar 1 partitie, namelijk de hele kolom.

```
SELECT prodid, nrprod, region,
       SUM(nrprod) OVER (PARTITION BY prodid)        AS byprodid,
       SUM(nrprod) OVER (PARTITION BY region)        AS byregion,
       AVG(nrprod) OVER (PARTITION BY prodid)        AS avgbyprod,
       AVG(nrprod) OVER ()                           AS avgtotal,
       MAX(nrprod) OVER (PARTITION BY prodid,region) AS maxnrprod,
       MIN(nrprod) OVER (PARTITION BY prodid,region) AS minnrprod
FROM   orders;
```

PRODID	NRPROD	REGION	BYPRODID	BYREGION	AVGBYPROD	AVGTOTAL	MAXNRPROD	MINNRPROD
3214	4	East	14	7	3,5	4	4	4
3214	8	West	14	20	3,5	4	8	1
3214	1	West	14	20	3,5	4	8	1
3214	1	West	14	20	3,5	4	8	1
5927	3	East	13	7	6,5	4	3	3
5927	10	West	13	20	6,5	4	10	10
7489	1	North	1	1	1	4	1	1

De twee SUM kolommen tonen de mogelijkheid om voor elke functie, dus elke kolom, een andere partitionering te hanteren.

De twee AVG kolommen illustreren het verschil tussen een functie met of zonder partitionering: de kolom AVGNRPROD geeft ons het gemiddelde verkoopsaantal per productid (dus per partitie, per groep), terwijl de kolom AVGTOTAL het algemeen gemiddelde (over de hele kolom) weergeeft.

2. Geef mij de geleidelijke evolutie: ORDER BY met cumulatieve en voortschrijdende functies

De ORDER BY clausule laat inderdaad toe de tussenwaarden van een functie te zien. Een ORDER BY verdeelt immers de partitie nog eens in kleinere gehelen (relatieve partities), wier relatieve waarde afgewogen wordt t.o.v. de absolute waarde van de partitie en geeft dus steeds antwoord op de vraag: aan welke totaalwaarde zit de functie voor de partitie nadat de functie apart is berekend voor deze relatieve partitie?

```
SELECT prodid,region,nrprod,ordnr,
       SUM(nrprod) OVER (PARTITION BY prodid)                AS part1,
       SUM(nrprod) OVER (PARTITION BY prodid ORDER BY ordnr)AS part10rd4,
       SUM(nrprod) OVER (ORDER BY prodid)                    AS ord1,
       SUM(nrprod) OVER (ORDER BY prodid, nrprod)            AS "ORD1,3",
       SUM(nrprod) OVER (ORDER BY nrprod, prodid)            AS "ORD3,1"
```

```
FROM   orders
ORDER BY prodid, ordnr;
```

PRODID	REGION	NRPROD	ORDNR	PART1	PART1ORD4	ORD1	ORD1,3	ORD3,1
3214	West	8	1	14	8	14	14	18
3214	East	4	3	14	12	14	6	10
3214	West	1	4	14	13	14	2	2
3214	West	1	6	14	14	14	2	2
5927	East	3	2	13	3	27	17	6
5927	West	10	6	13	13	27	27	28
7489	North	1	5	1	1	28	28	3

Deze query geeft verschillende views op de evolutie van het totaal aantal bestelde producten.

In kolom PART1 bevat de SUM functie alleen een PARTITION BY clausule: elke partitie wordt volledig apart beschouwd, hetgeen voor elk item van de partitie dezelfde som oplevert. We spreken dan van een rapporterende functie.

In kolom PART1ORD4 is er daarnaast ook een ORDER BY clausule gedefinieerd: dan geeft de functie het totaal voor de partitie (prodid), opgesplitst per relatieve partitie (we kunnen de evolutie volgen van 8 naar 14 voor prodid 3214). Voor de 3 ORD kolommen is er geen partitionering en zien we dus de evolutie tot het totaal van de hele kolom. Kolom ORD1 geeft voor prodid 5927 het voorlopige totaal na de berekening op relatieve partitie 5927. Elke combinatie van 2 kolommen in ORDER BY zorgt voor een sneller en geleidelijker voortschrijden. De ORD1,3 kolom doet dit in de volgorde van de gehele query (de prodid volgorde), daar waar de ORD3,1 kolom ons een ander zicht geeft door de volgorde te volgen van nrprod (maar uiteindelijk op hetzelfde resultaat uitkomt).

Wanneer de functie een ORDER BY clausule bevat, spreken we van een cumulatieve functie voor SUM of COUNT en van een voortschrijdende functie voor AVG, MAX of MIN (waarbij een tussenwaarde lager kan zijn dan de vorige tussenwaarde).

3. Elke rij vergelijken met een selectie van andere rijen: de windowing clausule met RANGE of ROWS

De windowing clausule is steeds een uitbreiding van de ORDER BY clausule en zorgt ervoor dat elke huidige rij beschouwd wordt als centrale waarde in relatie tot het (logische of fysische) bereik van rijen dat opgegeven is. Het resultaat is een sliding window en de functie wordt dan een gecentreerde functie genoemd.

```
SELECT ordnr, prodid, nrprod,
       COUNT(*) OVER (ORDER BY nrprod RANGE 2 PRECEDING)       AS "COUNT-2",
       FIRST_VALUE(nrprod)OVER (ORDER BY nrprod RANGE 2 PRECEDING)AS "LOWEST",
       COUNT(*)     OVER (ORDER BY nrprod RANGE BETWEEN 2 PRECEDING AND 2 FOLLOWING),
       AVG(nrprod)  OVER (ORDER BY ordnr ROWS 3 PRECEDING)        AS "AVG-3",
       AVG(nrprod)  OVER (ORDER BY ordnr ROWS UNBOUNDED PRECEDING)AS "AVG-n"
FROM   orders   ORDER BY nrprod DESC;
```

ORDNR	PRODID	NRPROD	COUNT-2	LOWEST	COUNT-/+2	AVG-3	AVG-n
6	5927	10	2	8	2	3,25	4
1	3214	8	1	8	2	8	8
3	3214	4	2	3	2	5	5
2	5927	3	4	1	5	5,5	5,5

```
6       3214    1       3       1       4       1,75    3
5       7489    1       3       1       4       2,25    3,4
4       3214    1       3       1       4       4       4
```

De 3 eerste kolommen (idealiter zonder detaillering per ordnr) hanteren de RANGE functie (logische offset), die toelaat te moduleren op basis van de waarden in een kolom (enkel mogelijk voor nummers en datums).

De RANGE 2 PRECEDING (kolom COUNT-2) genereert een sliding window, die toelaat om te zien hoe vaak b.v. 6 tot 8 items besteld geweest zijn: de RANGE 2 PRECEDING toegepast op 8 telt in dit geval -- de ORDER BY is immers default ascending, en in die volgorde zijn 6 en 7 de 2 waarden die voorafgaan (PRECEDING) aan 8 -- alle bestellingen van minimum 6 en maximum 8 items samen. Dus tot (en met) de huidige rij. Bij ORDER BY nrprod DESC RANGE 2 PRECEDING zouden 10 en 9 de 2 waarden zijn die voorafgaan aan 8: dus alle bestellingen van maximum 10 tot minimum 8 items.

De functie FIRST_VALUE (kolom LOWEST) geeft de waarden weer die eerst gevonden worden (minimumwaarde bij ASC en maximumwaarde bij DESC), aan het uiteinde van het bereik en LAST_VALUE zou de waarde van de huidige rij weergeven (die wordt immers laatst gevonden).

RANGE BETWEEN 2 PRECEDING AND 2 FOLLOWING (kolom COUNT-/+2) laat toe om, b.v. voor 8, onmiddellijk het aantal bestellingen te krijgen met min. 6 en max. 10 items.

De 2 AVG kolommen werken met de ROWS functie (fysieke offset), die dus toelaat te vergelijken met een bepaald aantal fysiek rijen. Bij ROWS zijn er geen beperkingen qua datatype. De ORDER BY nrprod ROWS 3 PRECEDING (in de kolom AVG-3) laat toe het aantal items in de huidige bestelling te vergelijken met het gemiddeld aantal items berekend op deze bestelling en de 3 vorige. In kolom AVG-n vergelijkt de UNBOUNDED PRECEDING met alle vorige in de huidige partitie (in dit geval de hele kolom).

De sleutelwoorden PRECEDING, FOLLOWING, UNBOUNDED, BETWEEN ... AND ..., en ook CURRENT ROW (voor complexere formuleringen met huidige rij als start- of eindpunt) kunnen zowel met RANGE als met ROWS gebruikt worden.

4. Geef de producten die het slechtst verkopen en waar: ranking functies

Ranking functies, enkel mogelijk in combinatie met een ORDER BY specificatie, laten toe een top-N te bepalen voor een bep. groep rijen.

```
SELECT prodid, region, nrprod,
       RANK()       OVER (PARTITION BY prodid ORDER BY nrprod) AS ranking,
       DENSE_RANK() OVER (PARTITION BY prodid ORDER BY nrprod) AS dense_ranking,
       ROW_NUMBER() OVER (PARTITION BY prodid ORDER BY nrprod) AS row_number
FROM   orders    ORDER BY prodid;
```

PRODID	REGION	NRPROD	RANKING	DENSE_RANKING	ROW_NUMBER
3214	West	1	1	1	1
3214	West	1	1	1	2
3214	East	4	3	2	3
3214	West	8	4	3	4

```
5927      East      3          1          1          1
5927      West     10          2          2          2
7489      North     1          1          1          1
```

Deze query geeft de rank van iedere bestelling in de groep van hetzelfde product. Bij gelijkwaardigheid vallen er hiaten, RANK volgt dus de Olympische stijl van ranking. De functie DENSE_RANK, zoals de naam laat vermoeden, geeft hetzelfde weer maar zonder hiaten. En ROW_NUMBER berekent het sequentiële rijnummer binnen de partitie.

5. Geef mij een vergelijking met een vorig/volgend lid van dezelfde dimensie: LAG/LEAD

```
SELECT prodid, SUM(nrprod)  AS sum,
       LAG (SUM(nrprod),1) OVER (ORDER BY prodid) AS lag_prodid,
       LEAD(SUM(nrprod),1) OVER (ORDER BY prodid) AS lead_prodid
FROM   orders
GROUP BY prodid
```

```
PRODID   SUM         LAG_PRODID   LEAD_PRODID
3214      14                       13
5927      13          14            1
7489       1          13
```

De LAG/LEAD functies laten dus toe om elk prodid te vergelijken met het vorige/volgende prodid. Als tweede argument van de functie wordt het aantal rijen opgegeven dat de vergelijkingsrij scheidt van de huidige rij. In deze query zou 2 als tweede argument prodid 7489 vergelijken met 3214 voor LAG en 3214 met 7489 voor LEAD. Deze functies bieden dus toegang tot meer dan één rij tegelijk, hetgeen het gebruik van een self-join vermijdt en dus de performance aanzienlijk verbetert.

Oplossing van de prijsvraag van het vorige nummer

De respons op de XSLT-prijsvraag in nummer 4, jaargang 5 van Exploring DB2 was eerder matig te noemen: er waren enkel inzendingen van ABIS-medewerkers. Een gemiste kans dus voor een gratis deelname aan een avondsessie “Exploring DB2 - live!”

In de online versie vindt u de uitgewerkte oplossing *(zie Bijlage op p. 17)*.

DB2 for z/OS en de Dynamic Statement Cache

Peter Vanroose (ABIS)

Het gebruik van de juiste indexen in een DW-omgeving onderstelt uiteraard dat de optimizer een goede kostinschatting kan maken van de mogelijke accesspaden. Static SQL heeft in dat opzicht, t.g.v. zijn onvermijdelijke parameter markers (met hostvariabelen) als belangrijk nadeel dat die kostberekening niet accuraat genoeg gebeurt. Dit geldt des te meer voor de complexe en rekenintensieve queries (o.a. OLAP) van een DW.

Dynamische SQL lost dit probleem op, maar heeft het grote nadeel, zeker voor complexe queries, dat de optimizatie (de BIND) telkens opnieuw moet gebeuren bij elke uitvoering. Zelfs een tweede EXECUTE van hetzelfde prepared statement, na een COMMIT, vereist een nieuwe PREPARE! Dit is een onaanvaardbare overhead. Waarom het accesspad niet ergens bijhouden ("cachen") en hergebruiken?

BIND heeft (sinds versie 6) een optie KEEPDYNAMIC(YES) die voor de lopende applicatie het accesspad in de *local cache* opslaat en hergebruikt. Dit is echter onvoldoende om bij een nieuwe uitvoering van dezelfde applicatie (of een andere met exact dezelfde query) een nieuwe PREPARE te vermijden. De *dynamic statement cache* (DSC) laat dit wel toe, tenminste als ZPARM CACHEDYN op YES werd gezet. (Sinds DB2 v8 is dit de default.) Zowel de SQL van elke dynamische query als het gekozen accesspad worden daar bijgehouden (zolang er ruimte is). En nog belangrijker: de optimizer zal de cache doorzoeken vooraleer een nieuwe dynamische PREPARE aan te vatten.

De details i.v.m. het hoe en wanneer van deze cache reuse zijn nogal ingewikkeld en vol uitzonderingsgevallen. Ook is een goede configuratie (i.h.b. grootte) van de DSC, in de EDM-pool, belangrijk. De default-grootte van de DSC is 5MB. Wanneer de cache vol zit, wordt de "oudste", dus minst recent gebruikte entry verwijderd.

Sinds versie 8 is het ook mogelijk om de DSC te bekijken, dus enerzijds de SQL te zien van recent uitgevoerde dynamische queries, en anderzijds hun accesspad te bekijken (via de EXPLAIN-tabellen).

Met het SQL-statement `EXPLAIN STMTCACHE ALL` wordt de hele cache (of toch alle queries van uzelf) in uw `DSN_STATEMENT_CACHE_TABLE` geplaatst. Elke query krijgt hier een STMT_ID, op basis waarvan ook het accesspad kan opgevraagd worden. Deze tabel bevat ook reeds statistische gegevens zoals de elapse-time en CPU-time, op basis waarvan een meer selectieve EXPLAIN kan gebeuren.

Met het SQL-statement `EXPLAIN STMTCACHE STMTID nnnn` worden de explain-details voor dit ene statement naar uw PLAN_TABLE en de andere explain-tabellen weggeschreven. Vanaf daar begint dan de klassieke performance-analyse: accesspad doorlichten, nieuwe indexen creëren, query herschrijven, ...

CURSUSPLANNING MAART - JULI 2009

Data warehouse concepten	450 EUR	27.03(L), 04.05(W)
DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	02.03(L), 23.03(W), 11.05(W), 08.06(L)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	11.06(W)
RDBMS-concepten	375 EUR	02.03(L), 23.03(W), 11.05(W), 08.06(L)
Basiskennis SQL	375 EUR	03.03(L), 24.03(W), 12.05(W), 09.06(L)
DB2 for z/OS basiscursus	1275 EUR	04.03(L), 25.03(W), 13.05(W), 10.06(L)
DB2 UDB for LUW basiscursus	1275 EUR	18.06(W)
SQL-QMF voor eindgebruikers	1275 EUR	06.05(W)
SQL workshop	800 EUR	16.03(L), 02.04(W), 25.05(W)
SQL voor gevorderden	450 EUR	23.04(W), 29.06(L)
DB2 procedural extensions	450 EUR	24.04(W), 30.06(L)
DB2 for z/OS programmeren voor gevorderden	900 EUR	16.04(L), 11.06(W)
DB2 for z/OS: SQL performance	1350 EUR	23.03(L), 27.05(W)
XML in DB2	450 EUR	29.04(L), 26.05(W)
DB2 for z/OS database administratie	1900 EUR	04.05(L)
DB2 for z/OS installation & migration	1050 EUR	01.06(UK)
DB2 for z/OS operations & recovery	1425 EUR	04.05(W), 03.06(UK)
DB2 for z/OS systems performance and tuning	1000 EUR	27.04(UK), 07.05(W)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	09.06(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Exploring DB2 - live! - partitioning	175 EUR	26.03(L) (avond)
Exploring DB2 - live! - SQLPL	175 EUR	27.04(L) (avond)
Exploring DB2 - live! - dynamic statement cache	175 EUR	08.06 (L) (avond)
Exploring DB2 - live! - UNICODE	175 EUR	26.06(L) (avond)

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
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Pelmolenlaan 1-K
NL-3440 AC Woerden
Tel. 0348-435570
Fax 0348-432493
training@abis.be

Bijlage

Hieronder vindt u een XSL-stylesheet dat uit een XML Schema een nieuw XSL-stylesheet zal genereren dat op z'n beurt SQL "INSERT"-statements genereert wanneer het wordt toegepast op een bijhorend XML-document, d.w.z., een document dat geldig is voor het XSML-Schema.

Een voorbeeld van een dergelijk Schema, van validerende XML, en van de gegenereerde INSERT-statements vindt u verderop.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xsl:template
match="/xs:schema/xs:element/xs:complexType/xs:sequence/xs:element[@maxOccurs
='unbounded']">
   <xsl:variable name="rownode"
select=".|/xs:schema/xs:element[@name=current()/@ref]"/>
<xsl:text disable-output-escaping="yes">
&lt;xsl:stylesheet version=&quot;1.0&quot;
xmlns:xsl=&quot;http://www.w3.org/1999/XSL/Transform&quot;&gt;
   &lt;xsl:output method=&quot;text&quot; omit-xml-
declaration=&quot;yes&quot;/&gt;
   &lt;xsl:template match=&quot;/</xsl:text><xsl:value-of
select="../../../@name"/><xsl:text disable-output-escaping="yes">&quot;&gt;
      &lt;xsl:for-each select=&quot;</xsl:text><xsl:value-of
select="@name|@ref"/><xsl:text disable-output-escaping="yes">&quot;&gt;
   &lt;xsl:text&gt;INSERT INTO </xsl:text><xsl:value-of
select="../../../@name"/> (<xsl:value-of
select="$rownode/xs:complexType/xs:attribute/@name|$rownode/xs:complexType/xs
:attribute/@ref"/>
      <xsl:text disable-output-escaping="yes">&lt;/xsl:text&gt;
      </xsl:text>
      <xsl:for-each select="$rownode/xs:complexType/xs:sequence/xs:element">
         <xsl:text disable-output-escaping="yes">&lt;xsl:if
test=&quot;</xsl:text><xsl:value-of select="@name|@ref"/>
         <xsl:text disable-output-
escaping="yes">&quot;&gt;,</xsl:text><xsl:value-of select="@name|@ref"/>
         <xsl:text disable-output-escaping="yes">&lt;/xsl:if&gt;
      </xsl:text>
      </xsl:for-each>)
      <xsl:text disable-output-escaping="yes">VALUES (&apos;&lt;xsl:value-of
select=&quot;@</xsl:text>
      <xsl:value-of
select="$rownode/xs:complexType/xs:attribute/@name|$rownode/xs:complexType/xs
:attribute/@ref"/>
      <xsl:text disable-output-escaping="yes">&quot;/>&apos;&lt;xsl:for-each
select=&quot;*&quot;&gt;,&apos;&lt;xsl:value-of
select=&quot;.&quot;/&gt;&apos;&lt;/xsl:for-each&gt;);
      &lt;/xsl:for-each&gt;
   &lt;/xsl:template&gt;
&lt;/xsl:stylesheet&gt;</xsl:text>
   </xsl:template>
</xsl:stylesheet>
```

Het volgende XML Schema kan dienen als input voor het bovenstaande XSL-stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Companies">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Company" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Company">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="tel" minOccurs="0">
     <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="16"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="street" type="xs:string" minOccurs="0"/>
    <xs:element name="strno" type="xs:string" minOccurs="0"/>
    <xs:element name="townno" minOccurs="0">
     <xs:simpleType>
  <xs:restriction base="xs:string"><xs:maxLength value="10"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="town" type="xs:string"/>
    <xs:element name="country">
     <xs:simpleType>
   <xs:restriction base="xs:string"><xs:maxLength value="4"/></xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="c_pno" type="xs:integer" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="nr" type="xs:integer" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Het volgende XML-bronbestand valideert met dit XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Companies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="Companies.xsd">
 <Company no="3">
  <name>ABIS</name>
  <tel>016/245610</tel>
  <street>Diestsevest</street>
  <strno>32</strno>
  <townno>3000</townno>
  <town>Leuven</town>
  <country>B</country>
  <c_pno>1</c_pno>
 </Company>
 <Company no="9">
  <name>Technisoft</name>
  <tel>02977-22456</tel>
  <town>Rotterdam</town>
  <country>NL</country>
 </Company></Companies>
```

Output van het XSL-stylesheet, wanneer toegepast op het Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="text" omit-xml-declaration="yes"/>
        <xsl:template match="/Companies">
                <xsl:for-each select="Company">
        <xsl:text>INSERT INTO Companies (COno</xsl:text>
                <xsl:if test="COname">,COname</xsl:if>
                <xsl:if test="COtel">,COtel</xsl:if>
                <xsl:if test="COstreet">,COstreet</xsl:if>
                <xsl:if test="COstrno">,COstrno</xsl:if>
                <xsl:if test="COtownno">,COtownno</xsl:if>
                <xsl:if test="COtown">,COtown</xsl:if>
                <xsl:if test="COcountry">,COcountry</xsl:if>
                <xsl:if test="COc_pno">,COc_pno</xsl:if>)
            VALUES ('<xsl:value-of select="@COno"/>'<xsl:for-each select="*">,
                        '<xsl:value-of select="."/>'</xsl:for-each>);
                </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```

Output van dit laatste XSL-stylesheet toegepast op het bovenstaande XML-bronbestand:

```
 INSERT INTO Companies (nr,name,tel,street,strno,townno,town,country,c_pno)
  VALUES ('3','ABIS','016/245610','Diestsevest','32','3000','Leuven','B','1');
 INSERT INTO Companies (nr,name,tel,town,country)
  VALUES ('9','Technisoft','02977-22456','Rotterdam','NL');
```