ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 2 - NUMMER 1, SEPTEMBER 2003


## OPEN CURSOR

*We hebben de eerste jaargang achter de rug!*
*Het volgens planning afwerken van relevante artikels is geen eenvoudige zaak. Je denkt immers automatisch: je hebt de kennis in huis - de rest is toch maar gewoon 'op papier zetten' ?*
*Niet dus.*
*Daarom hebben we tijdens de zomermaanden extra veel aandacht besteed aan het klaarstomen van de artikels die in de volgende nummers aan bod zullen komen. Applicatieontwikkeling krijgt een belangrijke plaats. SQL performance komt opnieuw aan bod, maar ook de nieuwste ontwikkelomgevingen: Java en .NET. We hebben het ook over de problematiek van triggers en redundantie, temporary tables, en utilities.*
*Op naar de tweede jaargang!*

*Het ABIS DB2-team.*

## IN DIT NUMMER:

- Over UNION-based views, in *Partitioned tablespaces - er is een alternatief*!
- Het begin van een nieuwe reeks rond applicatieperformance - *DB2 performance - case 1*.
- Voor het eerst in Exploring DB2 - *Dossier 8!*
- *Cursusplanning september 2003 - december 2003*.

## CLOSE CURSOR

Volgende maand krijgt u uiteraard een tweede case voorgeschoteld in onze reeks DB2 applicatieperformance. Verder bekijken we Java en .NET, specifiek vanuit DB2 standpunt.

Tot dan!

# Partitioned tablespaces - er is een alternatief!

*Kris Van Thillo (ABIS)*

Sinds DB2 voor OS/390 versie 2.3 ondersteunt DB2 formeel 'partitioned tablespaces'. Kort samengevat is de bedoeling de volgende:

- beschikbaarheid - grote hoeveelheden data en grote tabellen in meer beheersbare, onafhankelijke fysieke eenheden (datasets) op te delen. Utilities bijvoorbeeld halen hier het meeste voordeel uit.

- performance - verbeteren van antwoordtijden door het toepassen van optimisatietechnieken eigen aan partitionering: partitie eliminatie, parallelle I/O.

Elke nieuwe release van DB2 voegt een reeks eigenschappen toe aan de reeds rijke feature-set van partitioned tablespaces.

In dit artikel wordt een alternatieve techniek voor data-partitionering besproken. In eerste instantie bespreken we de louter technische opzet; daarna bekijken we toepasbaarheid en voordelen. Tenslotte vergelijken we summier beide technieken.

## Uitgangspunt: fullselect in VIEWs

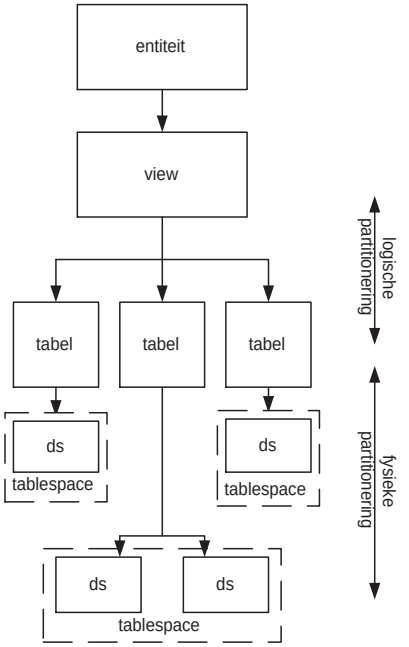
Het gebruik van de UNION- en UNION ALL-statements is in DB2 versie 7 voor OS/390 aanzienlijk uitgebreid. In vroegere versies van DB2 kon dit statement enkel worden gebruikt waar fullselects waren toegelaten - een reële beperking. Sinds DB2 versie 7 worden ze ook toegelaten overal waar in vorige versies subselects werden ondersteund! Meer specifiek worden deze statements nu o.a. toegelaten bij INSERT-operaties, in het SET-statement van een UPDATE, in tabelexpressies, en ook bij het aanmaken van views. En daar is het in dit artikel allemaal om te doen.

## Logische en fysieke partitionering

Het gebruik van partitioned tablespaces leidt steeds tot de creatie van 'onafzettelijke' partities - eigenlijk datasets - waarover tabeldata aan de hand van een cluster index wordt verspreid. Met 'onafzettelijke' bedoelen we dat de partities voor de gebruiker één geheel vormen. Traditioneel maakt men een onderscheid tussen 'range'- en 'hash'-partitionering. Range-partitionering wordt op een relatief eenvoudige manier geïmplementeerd. Hash-partitionering vergt typisch de generatie van een applicatie-gecontroleerde hash key. Recentelijk kunnen we hiervoor echter ook gebruik maken van automatisch door DB2 gegenereerde keys van het type ROWID.

Hoe deze partities worden aangemaakt heeft eigenlijk weinig belang; wat fundamenteel is in deze context, is dat deze door DB2 aangemaakte partities als dusdanig geen enkel *logische* betekenis hebben!

*Figuur 1: Partitionering*




De partities bestaan immers niet vanuit logisch standpunt. U kan de individuele partities niet benoemen, lezen, verwijderen, indexeren. U kan ze niet samenvoegen en u kan er geen toevoegen.

Inderdaad, de huidige DB2-partities hebben enkel fysieke karakteristieken: ze zijn via utilties als onafhankelijke entiteiten te benaderen. We denken hierbij bijvoorbeeld aan het 'copy' en 'reorg' utility. En uiteraard spelen ze een cruciale rol in de context van parallelle I/O. Opnieuw fysieke karakteristieken dus!

DB2 voor OS/390 versie 7 biedt ons de mogelijkheid om daarnaast ook logisch te partitioneren aan de hand van UNION-based views. Figuur 1 vat een en ander samen.

Het idee is vrij eenvoudig: een logische entiteit wordt op basis van logische criteria in een aantal logische onderdelen opgesplitst. Deze logische onderdelen worden aangemaakt als DB2-tabellen; de logische entiteit wordt geëxternaliseerd door de creatie van een UNION-based view. Hierdoor wordt de onderliggende structuur voor de applicatieontwikkelaar gedeeltelijk transparant.

Merk op dat deze logische partitionering perfect kan gecombineerd worden met de fysieke partitionering als boven uiteengezet; de boven bedoelde tabellen kunnen immers op hun beurt in gepartitioneerde tablespaces worden aangemaakt.

Extra voorbeeld: *Opzet van logische partitionering* op p. 13

## Opzet

Het opzetten van logische partitionering is zeer eenvoudig: we definiëren één tabel per logische partitie, en een view die de verschillende partities tot één geheel combineert. Voorbeeld 1 toont aan hoe e.e.a. conceptueel kan worden geïmplementeerd.

*Voorbeeld 1*

```
    create table partAA
    (aa integer,
     bb char(10))              We creëren een gewone tabel, met gewenste
    in database parttst;        database en tablespace karakteristieken!
    create table partBB         Check constraints om de logische grenzen
   (aa integer,               van elke tabel aangeven mag, maar moet niet!
    bb char(10))                Uiteraard aan te raden!
    in database parttst;

    create view part
    as                          View aangemaakt als een UNION ALL van
    select * from partAA        de boven gedefinieerde tabellen.
(b) where aa = 10               Let op de where-condities!
(a) union all
    select * from partBB
(b) where aa = 20;
```

Hoe transparant is deze structuur voor de applicatieontwikkelaar? Het antwoord is vrij eenvoudig. SELECT-statements ondervinden van deze structuur geen hinder - integendeel. UPDATE-, INSERT- en DELETE-operaties zijn minder transparant en eisen vaak een specifieke codering naar de onderliggende tabellen toe. Vaak gebruikt men in deze context ook applicatie-specifieke mappingtabellen als tussenoplossing. In de nabije toekomst zal DB2 Versie 8 hier wat extra mogelijkheden bieden.

## Rol van de optimizer

Het 'Query Rewrite' feature van de DB2-optimizer gaat als volgt om met UNION-based views (zie ook voorbeeld 2):

- voorwaarden, joins, en aggregaties gespecificeerd op de view worden gedistribueerd naar de afzonderlijke subselects in de view;
- subselects die na deze distributie niet langer relevant zijn, worden gewoon verwijderd; men spreekt van subselect-pruning. Dit is enkel mogelijk als aan 2 belangrijke voorwaarden is voldaan:
  - de view is aangemaakt op basis van een aantal UNION ALL-statements (**a**);
  - de logische grenzen van de tabellen gebruikt in de subselects moeten worden aangegeven aan de hand van een relevante WHERE-conditie in de view (**b**).

De keuze om voor dit laatste eigenlijk een dummy WHERE-conditie te voorzien komt vreemd over: strikt genomen zou de definitie van een CHECK-constraint op de onderliggende tabellen reeds voldoende moeten zijn.

*Voorbeeld 2*

```
select *
from part
where aa = 10
  and cc like '%x%'
                              na query rewrite:

                              select * from partAA
                              where aa = 10
predicaatdistributie  ---->   and aa = 10
predicaatdistributie  ---->   and cc like '%x%'
                              union all
                              select * from partBB
                              where aa = 20
predicaatdistributie  ---->   and aa = 10
predicaatdistributie  ---->   and cc like '%x%'

                                                    DB2 voert uit

                                                    select * from partAA
                                                    where aa = 10
                                                    and aa = 10
                                                    and cc like '%x%'
```

Aan de hand van de EXPLAIN-techniek kan subselect-pruning eenvoudig worden aangetoond.

```
Entiteit: sess (DB2 view)

Logische partities:
(DB2 tabellen, logische
partitionering op basis van
sesdate)

 sessj80p2, 1985 t/m 1989
 sessj90p1, 1990 t/m 1994
 sessj90p2, 1995 t/m 1999
 sessj00p1, 2000 t/m 2004

 sessj80p2 indx on sesdate
 sessj90p1 indx on sesdate

Entiteit: cours (DB2 table)
```

Een voorbeeld maakt dit verder duidelijk. Het is gebaseerd op de code die we via het web ter beschikking stellen. Om de EXPLAIN-output goed te kunnen interpreteren zijn een aantal eigenschappen van de gebruikte objecten in deze syntaxbox opgenomen.

Voorbeeld 3 geeft de inhoud van de EXPLAIN-tabel weer na het analyseren van het aldaar opgenomen voorbeeld. Een aantal observaties:

- Inderdaad - 'subselect-pruning' heeft plaatsgevonden: 2 van de 4 logische partities zijn door de optimizer genegeerd. De join werd gedistribueerd naar de weerhouden subselects. Eén join werd aldus vertaald in 2 onafhankelijke join operaties.

- De waarden in de 'QB' kolom geven het 'parent query block' weer van elk aanwezig queryblock; queryblock 1 en 5 leveren aldus 'input' voor queryblock 2, de eigenlijke UNION ALL stap.

- Elke logische partitie is individueel, en onafhankelijk van de andere partities geoptimiseerd. Partitie-onafhankelijke optimisatie is dus een feit! De kritische lezer begrijpt dat we in deze context eigenlijk te maken hebben met queryblock-onafhankelijke optimisatie -

een techniek die reeds lang bestond, maar NIET in de context van partitioned tablespaces. Inderdaad, partitie 'sessj90p1' werd via een 'R'-scan gelezen, terwijl partitie 'sessj80p2' aan de hand van een index werd benaderd.

- Een belangrijke factor die de performance van deze partitioneringstechniek bepaalt is UNION ALL-materialisatie: het al dan niet gebruiken van DB2-'workfiles' voor het opslaan van tussentijdse resultaten (een TT-waarde 'W'). Een aantal belangrijke APARS zijn beschikbaar om het gedrag van DB2 in deze context bij te sturen.

*Voorbeeld 3*

```
explain plan set queryno = 100 for
select * from sess, cours
where sesdate between '15.12.1989' and '01.12.1994'
and se_cno = cno

QBl  Meth  Tab        Acc  IxNme  SortN  SortC PQ  Pref  TT  QBT
 2    0                -         NNNN   NNNN   0  -     -   UNIONA
 1    2    cours       R         NYNN   NYNN   2  S     T   NCOSUB
 1    0    sessj90p1   R         NNNN   NNNN   2  S     T   NCOSUB
 5    2    cours       R         NYNN   NYNN   2  S     T   NCOSUB
 5    0    sessj80p2   I    IX   NNNN   NNNN   2  L     T   NCOSUB

PQ: Parent query- QBl of the parent block
TT: Table type - T(Table), W(Workfile), Q(Temp.result), F(table funct)
```

Een join tussen logisch-gepartitioneerde views wordt door DB2 correct 'gepruned'. DB2 is echter niet in staat de gevolgen van het eventueel 'equi-partitioned' zijn van deze views in te schatten, zelfs niet na toevoeging van redundante WHERE-condities!

## Host variabelen

Pruning van de subselects vindt normaal plaats op moment van de bind. Bij gebruik van host-variabelen is dit niet mogelijk: hun inhoud is op dit moment nog niet gekend. Pruning wordt automatisch uitgesteld tot op het moment dat de inhoud van deze variabelen vaststaat. Dit is steeds het geval, onafhankelijk van de bind-optie REOPT(VARS), vereist voor partitie-pruning bij het gebruik van partitioned tablepaces.

## Een summiere vergelijking

Logische partitionering, al dan niet gekoppeld aan fysieke partitionering, levert een aantal belangrijke, bijkomende voordelen.

- De maximale entiteit grootte is plots in omvang toegenomen: van 16 TB (partitioned tablespaces) naar 255 * 16 TB! Inderdaad, een VIEW kan 'slechts' 255 tabellen (en/of subselects) omvatten.

- Logische partitionering op basis van onafhankelijke tabellen geeft de DBA de mogelijkheid een volledige, partitie-onafhankelijke indexstrategie uit te werken. De individuele tabellen kunnen immers onafhankelijk van mekaar worden geïndexeerd, ze kunnen er een

specifieke clustervolgorde op nahouden, etc. Afhankelijk van de vereiste toegang maakt men bijkomende indexen aan.

Merk op dat in dit scenario, de omvang van bijvoorbeeld reorganisaties, beperkt blijft tot één enkele partitie. Of anders gezegd: doordat de tabel en de index steeds 'equi-partitioned' zijn, is de impact van onderhoud op een partitie ook steeds beperkt tot die partie (en de erop aangemaakte indexen).

Op omvangrijke gepartitioneerde tabellen creëert men doorgaans geen, dan wel een zeer beperkt aantal bijkomende indexen (de zogenaamde NPIs - non partitioned indexes). Onderhoud van deze indexstructuren na het uitvoeren van UPDATE-, INSERT- en DELETE-operaties op de tabel is doorgaans het grootste probleem.

- Eigen aan een datawarehouse-omgeving is het 'rolling window' fenomeen. Partities met oude data moeten worden verwijderd; nieuwe partities met recente data moeten worden toegevoegd. Een taak die met partitioned tablespaces zo goed als onmogelijk is (akkoord, met een combinatie van o.a. COPY en REORG komen we een heel eind). Maar wat is makkelijker dan een tabel met oude data uit een view-definitie te verwijderen, en een met nieuwe data aan de definitie toe te voegen? Merk op dat deze verwijderde tabel als 'gearchiveerd' kan worden beschouwd, en toch ook beschikbaar blijft voor directe SQL-toegang. Maar zoals steeds: hou rekening met 'view-dependenties' ... Plannen en packages zullen door DB2 zelf onderhanden worden genomen.

- Voorbeeld 3 geeft het reeds aan: DB2 beschikt over meer alternatieven wat betreft toegangspadselectie bij het gebruik van 'logische' partitionering. Dit vergroot het belang van correcte statistieken (RUNSTATS). Uiteraard neem ook de 'kost' van de optimisatie toe.

- Query parallellisme is een probleem. De verschillende queryblokken worden immers steeds sequentieel uitgevoerd. Parallellisme binnen een query block is mogelijk indien de gebruikte tablespace gepartitioneerd is. Logische partitionering leidt meestal tot een lagere 'degree of parallelism' dan fysieke partitionering.

## En dus?

Beide technieken hebben duidelijk voor- en nadelen. In dit artikel zijn er een aantal kort op een rijtje gezet. De belangrijkste boodschap is echter duidelijk: UNION-based views bieden DBAs een aantal voordelen die allemaal gerelateerd zijn aan het werken met logisch manipuleerbare entiteit-'subcomponenten'.

De view is terug van weggeweest!

# DB2 performance - case 1

*Eric Venmans (ABIS)*

Optimaliseren van SQL-requests is een mooie zaak. Vooral als het gaat over requests die als 'static embedded SQL' gebruikt worden in programma's; programma's die nadien veelvuldig worden uitgevoerd. Enerzijds moet geoptimaliseerd worden als blijkt dat de uitvoertijd van een bepaalde query te wensen overlaat, anderzijds kan ook vroeger geoptimaliseerd worden, nog voor de SQL-request in de programmacoding is aangebracht. Dit laatste heeft als voordeel dat men de coding nog moet schrijven. Dit vergemakkelijkt het eventueel uitsplitsen van SQL-requests (b.v. een geprogrammeerde join), het aanpassen van de verwerkingsprocedure (b.v. via temporary tables) en zo meer.

Optimaliseren vooraleer men requests in programma's plaatst, veronderstelt:

- dat men een omgeving heeft die 'realistisch' is; d.w.z. een omgeving met structuren (tabellen, indexen, clustering, partitioning, ... en statistieken!) die overeenkomen met de bestaande of nog te construeren productieomgeving;
- dat men de EXPLAIN-functie correct weet te gebruiken.

Dit laatste is minder evident dan het lijkt.

In dit en volgende artikels geven we voorbeelden van hoe het NIET moet. Ook daaruit valt heel wat te leren.

## CASE 1: betere EXPLAIN, slechter toegangspad

Onderstaande query wordt in deze case als uitgangspunt gebruikt. Ga er van uit dat P de tabel is, PrCode een kolom uit deze tabel, waarop tevens een index IXPR is aangemaakt.

```
Explain plan
set queryno = 100 for
Select
from P
where substr(PrCode, 3, 1) =
:variabele
```

DB2 gebruikt initieel - zie Voorbeeld 1 - geen index omdat de voorwaarde in de SQL-request een functie, de 'substr()', bevat. Het is een stage-2 voorwaarde en de DB2 Data Manager, die als enige de inhoud van indexen kan interpreteren, is niet in staat om deze stage-2 voorwaarden te controleren.

*Voorbeeld 1: Query 100 - Explain*

| QNo | Methd | TN | Access | MatchCols | IxNme | IxOnly | Sort_N | Sort_C | Prefetch |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 0 | P | R | 0 | | N | NNNN | NNNN | |

(Access 'R' staat voor 'table scan', d.w.z. het sequentiëel lezen van de ganse tabel)

Volgende variant van de SQL-request levert wel het gebruik van een index op, een matching nog wel (MatchCols > 0):

```
Explain plan
set queryno = 200 for
Select
from P
where substr(PrCode, 3, 1) =
:variabele
and PrCode between
:variabele1 and :variabele2
```

Variabele1 en variabele2 worden bij het uitvoeren ingevuld met 'extreme' waarden (typisch: low en high values) om het resultaat van de query niet te beïnvloeden. De voorwaarde waarin dit gebeurt, wordt dikwijls een 'dummy' voorwaarde genoemd.

*Voorbeeld 2: Query 200 - Explain*

| QNo | Methd | TN | Access | MatchCols | IxNme | IxOnly | Sort_N | Sort_C | Prefetch |
|---|---|---|---|---|---|---|---|---|---|
| 200 | 0 | P | I | 1 | IXPR | N | NNNN | NNNN | L |

DB2 wil nu wel de index gebruiken op de kolom die vermeld wordt in de voorwaarden, maar dit is enkel om de extra voorwaarde, de 'dummy' voorwaarde, te controleren (Voorbeeld 2). Het wordt een index access waarvoor alles wat de Data Manager kan controleren, 'matcht'. We hebben een toegangspad dat volgens EXPLAIN beter oogt, maar bij het uitvoeren echter nog meer resources vergt dan in de oorspronkelijke schrijfwijze.

De DB2-kostenindicaties (neergezet in de DSN_STATEMNT_TABLE) tonen dit aan (de hier weergegeven resultaten komen uit een test, uitgevoerd in een DB2 for OS/390 V7 systeem) - zie Voorbeeld 3.

*Voorbeeld 3: Query 100 versus 200 - Evaluatie van de kosten*

| QNo | SqlType | Category | MSec | SUnits | Reason |
|---|---|---|---|---|---|
| 100 | SELECT | A | 9088 | 35086 | |
| 200 | SELECT | B | 913 | 3522 | HOST VARIABLES |

Bij het oppervlakkig bekijken van de kostenindicatie lijkt de ingreep met de 'dummy' voorwaarde (QNo = 200) een 'goede' zaak. Een grondige analyse toont echter het tegendeel. DB2 veronderstelt immers voor QNo = 200 dat men voor variabele1 en variabele2 'echte, selectieve' waarden gaat invullen. Zolang deze waarden via variabelen

worden doorgegeven, kunnen we die veronderstelling moeilijk beïnvloeden. We kunnen echter wel bij wijze van experiment een EXPLAIN vragen met ingevulde waarden (low en high values) voor variabele1 en variabele2.

*Voorbeeld 4: Query 200 - Evaluatie van de kosten (variant a)*

| QNo | SqlType | Category | MSec | SUnits | Reason |
|---|---|---|---|---|---|
| 200 | SELECT | A | 9282 | 35836 | |

De kosten met de ingevulde 'dummy' voorwaarde zijn nu iets groter dan deze zonder. De 'echte' kosten liggen nog een stuk hoger, want ondertussen heeft DB2 weer de 'table scan' als toegangspad gekozen. De optimizer 'weet' nu immers dat de 'dummy' voorwaarde 'matcht' voor alle index waarden en dat dus het gebruik van een index hier geen zin heeft.

Om de 'echte' kosten te kennen, kunnen we met 'hints' werken. We vragen via 'hints' aan DB2 om voor de query met ingevulde waarden (low en high values) toch een matching index scan toe te passen. Dit levert volgende berekende kosten:

*Voorbeeld 5: Query 200 - Evaluatie van de kosten (variant b)*

| QNo | SqlType | Category | MSec | SUnits | Reason |
|---|---|---|---|---|---|
| 200 | SELECT | A | 12577 | 48560 | |

Ook deze kosten moeten kritisch bekeken worden. DB2 gebruikt namelijk 'list-prefetch' bij het lezen van de index. Als bij het uitvoeren ervan, DB2 'merkt' dat er te veel indexinformatie (meer dan 25%) moet verwerkt worden, dan wordt dynamisch het gebruik van die index 'gecanceld' en vervangen door een 'table scan'. Dit zal voor onze query zeker gebeuren, omdat 100% van de index informatie 'matcht'.

Het is echter zonder meer duidelijk dat het toegangspad voor de variante met de 'dummy' voorwaarde enkel een mooiere explain oplevert (van 'table scan' naar 'matching index scan'), maar geen beter toegangspad (van 35086 service units naar iets tussen 35836 en 48560 service units).

Dit betekent niet dat het gebruik van 'dummy' voorwaarden geen oplossing kan zijn voor performance-problemen. Het voorbeeld dat we hier gebruiken (en dat met opzet zeer eenvoudig is gehouden), toont enkel aan dat het gebruik van 'dummy' voorwaarden niet automatisch de gewenste performance verbetering geeft. De ingreep moet omzichtig gebeuren en vraagt een 'grondige' analyse met kennis van zaken.

# DOSSIER 8

## Alles wordt groter - grenzen worden verlegd!

We mogen inderdaad voor de nieuwe versie van DB2 wederom de adjectieven groter, meer en beter gebruiken. Uiteraard biedt DB2 V8 voor z/OS betere performance, meer mogelijkheden, grotere beschikbaarheid en toegenomen flexibiliteit. We gaan het hier niet in het algemeen over hebben, maar wel concreet vermelden waar we de adjectieven 'groter' en 'meer' mogen gebruiken. Een selecte bloemlezing.

DB2 V8 voert de multiple-row-INSERT in. Met één enkel INSERT-statement kunnen meerdere rijen toegevoegd worden aan een tabel. Gelijkaardig werkt de nieuwe multiple-row-FETCH, met één FETCH worden meerdere rijen opgehaald. Dit vereist aanpassingen aan het INSERT-, DECLARE CURSOR-, en FETCH-statement.

Een partitioned tablespace kan maximum 4096 partities bevatten (254 in versie 7). Wanneer men elke dag 1 partitie in gebruik wenst te nemen dan kan men nu i.p.v. de gegevens van de laatste 8 maand te bewaren, de gegevens van de laatste 11 jaar bewaren. Deze verandering gaat gepaard met een aangepast maximale grootte voor de partitioned table zelf: van 16 terabytes naar 128.

In DB2 V8 kunnen 225 tabellen gejoined worden, dit ten opzichte van 15 tabellen in vorige versies.

SQL-statements kunnen veel langer zijn. Maximale lengte: 2MB!

De lengte van een kolomnaam gaat van 18 naar 30 bytes. Een tabel, view, of indexnaam kan 128 karakters bevatten.

Het maximum van 255 bytes in een index-key wordt verlegd naar 2000 bytes.

De 'multiple distinct' brengt meer mogelijkheden voor het SELECT-statement. Volgende queries kunnen in DB2 V8 probleemloos uitgevoerd worden:

```
SELECT AVG(DISTINCT COL1), SUM(DISTINCT COL2)
FROM TAB1
```

```
SELECT DISTINCT COUNT(DISTINCT(COL1)), COUNT(COL2)
FROM TAB1
```

```
SELECT COUNT(DISTINCT(COL1)), COUNT(DISTINCT(COL2))
FROM TAB1
```

```
SELECT COUNT(DISTINCT(COL1))
FROM TAB1
GROUP BY COL2 HAVING AVG(DISTINCT(COL3))>1
```

DB2 V8 vereist de z/Architecture met z/OS V1R3. DB2 V8 zal dus niet alleen 64-bit adressering gebruiken voor real storage maar ook voor virtual storage. Direct gevolg hiervan is dat hiperpools en dataspaces overbodig geworden zijn, nog sterker, ze worden niet meer ondersteund.

*Katrien Platteborze (ABIS)*

# CURSUSPLANNING SEPT - DEC 2003

| | | |
|---|---|---|
| DB2 concepten | 375 EUR | 24/10 (L), 09/12 (W) |
| DB2 for OS/390, een totaaloverzicht | 1625 EUR | 13-17/10 (W), 03-07/11(L), 08-12/12 (W) |
| DB2 UDB, een totaaloverzicht | 1625 EUR | 13-14&20-22/10 (W), 03-04&12-14/11 (L) |
| RDBMS concepten | 325 EUR | 13/10 (W), 03/11 (L), 08/12 (W) |
| Basiskennis SQL | 325 EUR | 14/10(W), 04/11(L), 09/12 (W) |
| DB2 for OS/390 basiscursus | 975 EUR | 15-17/10 (W), 05-07/11(L), 10-12/12 (W) |
| DB2 UDB basiscursus | 975 EUR | 20-22/10 (W), 12-14/11(L) |
| SQL workshop | 700 EUR | 29-30/09 (L), 27-28/10 (W), 17-18/11 (L), 18-19/12 (W) |
| DB2 for OS/390 programmering voor gevorderden | 1050 EUR | 27-29/10 (W), 03-05/12 (L) |
| DB2 for OS/390: SQL performance | 1200 EUR | 03-05/11(W), 15-17/12 (L) |
| DB2 UDB applicatieperformance | 400 EUR | 10/11 (W) |
| Database applicatieprogrammering met Java | 800 EUR | 20-21/10 (L), 13-14/11(W) |
| Fysiek ontwerp van relationele databases. | 700 EUR | 06-07/11(L) |
| DB2 for OS/390 database administratie | 1600 EUR | 20-23/10 (W), 02-05/12 (L) |
| DB2 for OS/390 restart and recovery | 1650 EUR | 01-03/10 (W) |
| DB2 UDB systeembeheer en performance | 400 EUR | 11/11 (W), 15/12 (L) |
| DB2 for OS/390 V7 upgrade voor ontwikkelaars | 375 EUR | 26/09 (L), 12/11 (W) |
| DB2 UDB en zijn extenders: XML en text search | 200 EUR | 23/10 (W), 16/12 (L) |
| DB2 UDB integratie met MQSeries | 200 EUR | 23/10 (W), 16/12 (L) |

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

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245691
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

## Opzet van logische partitionering

Onderstaand script maakt de test omgeving aan. De tabellen in de test worden opgevuld a.d.h.v. de inhoud van bestaande tabellen. Op alle relevante tablespaces wordt voorts ook het runstats-utility gedraaid.

```
--
-- Database en tablespaces
--
 create database parttst;
--
 create tablespace pj80p2 segsize 16 in parttst;
 create tablespace pj90p1 segsize 16 in parttst;
 create tablespace pj90p2 segsize 16 in parttst;
 create tablespace pj00p1 segsize 16 in parttst;
--
-- Niet gepartitioneerde table cours
--
 create table cours
 like tbaccad.tpvcourses
 in database parttst;
 insert into cours
 select * from tbaccad.tpvcourses;
--
-- 'Partities'
--
 create table sessj80p2
 (seno      integer not null,
 sesdate   date,
 setype    char(2),
 selang    char(1),
 seincomes integer,
 se_cno    smallint,
 constraint j80p2
   check (sesdate between '01.01.1985'and '31.12.1989'))
 in parttst.pj80p2;
--
 insert into sessj80p2
 select seno, sesdate, setype, selang, seincomes, se_cno
 from tbaccad.tpvsessions
 where sesdate between '01.01.1985' and '31.12.1989';
--
 create table sessj90p1
 (seno      integer not null,
 sesdate   date,
 setype    char(2),
 selang    char(1),
 seincomes integer,
 se_cno    smallint,
 constraint j90p1
   check (sesdate between '01.01.1990' and '31.12.1994'))
 in parttst.pj90p1;
--
 insert into sessj90p1
 select seno, sesdate, setype, selang, seincomes, se_cno
 from tbaccad.tpvsessions
 where sesdate between '01.01.1990' and '31.12.1994';
--
```

```
create table sessj90p2
(seno      integer not null,
sesdate   date,
setype    char(2),
selang    char(1),
seincomes integer,
se_cno    smallint,
constraint j90p2
  check (sesdate between '01.01.1995' and '31.12.1999'))
in parttst.pj90p2;
--
insert into sessj90p2
select seno, sesdate, setype, selang, seincomes, se_cno
from tbaccad.tpvsessions
where sesdate between '01.01.1995' and '31.12.1999';
--
create table sessj00p1
(seno      integer not null,
sesdate   date,
setype    char(2),
selang    char(1),
seincomes integer,
se_cno    smallint,
constraint j00p1
  check (sesdate between '01.01.2000' and '31.12.2004'))
in parttst.pj00p1;
--
insert into sessj00p1
select seno, sesdate, setype, selang, seincomes, se_cno
from tbaccad.tpvsessions
where sesdate between '01.01.2000' and '31.12.2004';
--
-- De ‘tabel’
--
create view sess
as (
select * from sessj80p2
where sesdate between '01.01.1985' and '31.12.1989'
union all
select * from sessj90p1
where sesdate between '01.01.1990' and '31.12.1994'
union all
select * from sessj90p2
where sesdate between '01.01.1995' and '31.12.1999'
union all
select * from sessj00p1
where sesdate between '01.01.2000' and '31.12.2004'
)
--
-- Indexes op slechts 2 partities
--
create index ixj80p2 on sessj80p2(sesdate);
create index ixj90p1 on sessj80p2(sesdate);
```