ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 3 - NUMMER 3, MAART 2005


## OPEN CURSOR

*Het sneeuwt - een understatement - maar het sneeuwt. Hard. Een deel van het openbare leven is hier in Nederland zelfs ontwricht - Schiphol sluit.*
*Wat kan er met mijn DB2 systeem gebeuren, met gelijkaardige gevolgen? Mag er wel iets gebeuren met dergelijke gevolgen?*
*Beschikbaarheid, betrouwbaarheid, onderhoudbaarheid staan nog steeds bovenaan op het verlanglijstje van menig applicatie- en systeem-DBA. Bij elke nieuwe release worden features voorzien die hier moeten toe bijdragen. Een van dergelijke features wordt onder andere in dit nummer verder ingeleid. Ook in volgende nummers komen naast applicatiegerichte topics features aan bod die het de DB2 specialist toelaten meer tijd uit te trekken voor andere dingen - bijvoorbeeld sneeuw!*

*Veel leesgenot!*
*Het ABIS DB2-team.*

## IN DIT NUMMER:

- We gaan verder met de bespreking van het belang van utilities in de context van applicatieontwikkeling - in *DB2 utilities en applicaties - 2*.
- Wat is het nut van RTS - in *Realtime statistics voor DB2 for z/OS.*
- En *Dossier 8* gaat in op REOPT(ONCE).
- *Cursusplanning maart 2005 - juni 2005*
- Web-only: *Mock objects as testing method* - een ABIS presentatie op de Javapolis 2004, Antwerpen (zie *http://www.abis.be/resources/presentations/javapolis2004mockobjects.pdf*).

## CLOSE CURSOR

In het volgende nummer aandacht voor OLAP in DB2 UDB. En gaan we verder met onze reeks over utilities. Onder andere.

Tot dan!

# DB2 utilities en applicaties - 2

*Eric Venmans (ABIS)*

## Inleiding

Wanneer veel rijen uit een grote tabel moeten verwijderd worden tijdens een zogenoemd 'opkuis' proces, is het gebruik van utilities een aantrekkelijk alternatief voor het normaal werken met SQL statements. Vooral als de duurtijd van het verwerkingsproces moet beperkt blijven, is het alternatief het overwegen meer dan waard. In dit artikel geven we in dit verband twee voorbeelden.

Het eerste is een eenvoudig 'recht aan, recht door' voorbeeld. Het tweede is moeilijker, maar sluit nauwer aan bij de realiteit.

## Voorbeeld 1: onafhankelijke tabel

Neem volgende situatie als uitgangspunt.

Een onafhankelijke tabel (dikwijls 'stand-alone table' genoemd) krijgt gedurende elke werkdag duizenden nieuwe rijen toegevoegd. Dit gebeurt online in pakketjes gaande van 10 rijen tot soms meer dan 1000. Deze nieuwe rijen worden geclusterd op hun timestamp, zodat in de dataset(s) alle rijen telkens aan het einde sequentieel worden toegevoegd.

Ook de verwerking van de nieuwe informatie gebeurt online. Om een dubbele verwerking te vermijden, krijgt een verwerkte rij een indicatie (een 'flag'). Onmiddellijk online deleten kan niet omdat men de verwerking de dag zelf nog moet kunnen 'corrigeren'. Na het afsluiten van de dagactiviteiten mogen (en moeten) alle verwerkte rijen (+/- 90%) verwijderd worden.

Dit laatste proces, zou men via een eenvoudig programma met het volgend SQL statement kunnen uitvoeren - code 1.

*Code 1*

```
DELETE FROM X
WHERE VerwerkingsStatus = 'In Orde'
```

De kolom ‘VerwerkingsStatus’ bevat de indicatie (of 'flag'), met X de onafhankelijke tabel waarvan boven sprake.

Bovenvermeld proces vraagt echter veel tijd, omdat bij elke rij die geschrapt wordt:

- de operatie gelogged wordt
- en vooral omdat de indexen immers zoals steeds 'in flight' worden aangepast (in ons voorbeeld zijn er 4 indexen betrokken bij dit proces).

Vormen geen probleem:

- locking (indien tenminste DB2 spontaan of expliciet gevraagd, een 'table' lock aanwendt)
- en RI controle (het is immers een 'stand-alone table').

Na het delete proces is ook nog een uitvoering van de REORG utility nodig om de resterende rijen en index-entries te reorganiseren (te 'comprimeren' in ons geval).

Voor het inschatten van hoe zwaar bovengeschetste oplossing weegt, werd in eerste instantie een beroep gedaan op de DB2 Estimator.

De berekeningen gebeurden op basis van kenmerken die in de uiteindelijke productieomgeving verwacht werden. De DB2 Estimator gaf een duurtijd aan van ongeveer 1 uur voor de delete-actie en slechts 2 minuten extra voor de daaropvolgende reorganisatie. Men kan echter de delete-acties ook laten uitvoeren tijdens de reoraganisatie i.p.v. ervoor. De REORG utility voert het verwijderen uit i.p.v. het programma (met de SQL delete). Dit gaat via volgend eenvoudig voorbeeld - code 2.

*Code 2*

```
REORG TABLESPACE dbx.tbx
SHRLEVEL NONE DISCARD
FROM TABLE X
WHEN (VerwerkingsStatus = 'In Orde')
```

Een berekening via DB2 Estimator geeft aan dat dit process om en bij de 3 minuten uitvoertijd vraagt. Een erg spectaculaire tijdwinst. Dit wordt bevestigd door testen, gedaan op een tabel die zowat 10% van de uiteindelijke productiegegevens bevat. Het verwijderen van rijen via de REORG utility verbruikt ongeveer 10% van de CPU die de SQL delete nodig heeft. De uitvoertijd (de 'elapsed time') valt terug op ongeveer 5% van de oorspronkelijke tijd.

Het verschil is hier zo groot omwille van de 4 indexen die bij de SQL delete 'in flight' worden bijgewerkt. Dezelfde delete (met SQL) op dezelfde tabel, maar dit keer zonder indexen, geeft een berekende tijd van minder dan 1 minuut!

## Voorbeeld 2: gerelateerde tabellen

In het tweede, meer realistische voorbeeld is de informatie in twee tabellen geplaatst.

De rijen van een eerste tabel X zijn nu dependent van rijen uit een parent tabel Y. Gemiddeld zijn er zo'n 100 dependent rijen per parent. Een delete-actie gebeurt steeds vanuit de parent table. Daar worden dagelijks zowat 90% van de rijen geschrapt. Hun dependents verdwijnen automatisch door de cascade delete rule, tussen tabel X en tabel Y gedefinieerd.

Om de delete met SQL uit te voeren, zou volgend statement in aanmerking komen (enkel parent rijen waarvan alle dependent rijen verwerkt zijn, mogen en moeten verwijderd worden) - zie code 3.

*Code 3*

```
DELETE FROM Y
WHERE Id NOT IN (SELECT Y_Id
                 FROM   X
                 WHERE  VerwerkingsStatus = 'Nog te doen')
```

Deze SQL is echter ongeldig omdat de tabel in de subquery 'delete-dependent' is van de 'target table' van de delete-actie. Maar we kunnen via een cursor select dit probleem eenvoudig omzeilen - code 4.

*Code 4*

```
DECLARE TeVerwijderenRijen
CURSOR FOR
SELECT *
FROM Y
WHERE Id NOT IN (SELECT Y_Id
                 FROM   X
                 WHERE  VerwerkingsStatus = 'Nog te doen')
FOR UPDATE
```

DB2 Estimator is niet meer geschikt om voor dit soort acties rechtstreeks uitvoertijden te berekenen. Er is immers geen ondersteuning voor 'referential integrity' acties.

Het meten van de uitvoertijd in onze testomgeving gaf ten opzichte van het eerste voorbeeld, volgend resultaat: 10% meer CPU verbruik en dubbele uitvoertijd. Het aanpassen van de indexen is zo dominant dat er geen al te spectaculaire verschillen zijn. Het delete process wordt door de cursor-benadering wel opgesplitst in deelopdrachten. Dit verklaart grotendeels de langere doorlooptijd.

We zouden ook hier kunnen trachten via utilities sneller te werken, maar dit is niet zomaar te doen. We kunnen via DISCARD in het REORG process niet aanduiden welke rijen wel en welke niet moeten verwijderd worden. In de REORG kan geen informatie uit verschillende tabellen gecombineerd worden. Een mogelijkheid die overblijft is het 'prepareren' van de tabellen. Dit kan als volgt - zie code 5.

*Code 5*

```
UPDATE Y
SET Status = 'Moet weg'
WHERE Id NOT IN (SELECT Y_Id
                 FROM   X
                 WHERE  VerwerkingsStatus = 'Nog te doen')
.....
UPDATE X
SET Status = 'Moet weg'
WHERE.Y_Id IN (SELECT   Id
               FROM     Y
               WHERE    Status = 'Moet weg')
```

Nu kan de reorganisatie uitgevoerd worden met volgende utility-statements - zie code 6.

Code 6

```
REORG TABLESPACE dbx.tbx
SHRLEVEL NONE DISCARD
FROM TABLE X
WHEN (Status = 'Moet weg')

REORG TABLESPACE dbx.tby
SHRLEVEL NONE DISCARD
FROM TABLE Y
WHEN (Status = 'Moet weg')
```

Na metingen komen we tot de volgende vaststellingen: via het prepareren met SQL en het schrappen via de REORG utility vallen het CPU verbruik en de totale uitvoertijd beide terug tot ongeveer 20% van de oorspronkelijk gemeten waarden.

De verschillen zijn minder spectaculair dan in het eerste voorbeeld, maar het alternatief via de REORG utility blijft zeer aantrekkelijk voor '(te) langdurende' activiteiten.

Wie creatief is, kan nog andere scenario's bedenken om via utilities te werken. Bovendien zullen deze scenario's ingewikkelder worden naarmate de dataorganisatie ingewikkelder wordt. Ook de aard van de verwerking speelt hierbij een rol.

Algemene regel is wel: gebruik SQL om de informatie aan te duiden die moet verdwijnen (via update van een niet-geïndexeerde kolom) en gebruik de REORG utility met de DISCARD optie om de rijen definitief te verwijderen.

## Besluit

Het verwijderen van rijen via de REORG utility is een zeer aantrekkelijk alternatief voor het SQL delete statement, wanneer het gaat om veel rijen uit een tabel met meerdere indexen. Zoveel te hoger het % te verwijderen rijen en zoveel te hoger het aantal indexen, zoveel te groter de besparing op CPU en uitvoertijd. De aangehaalde resultaten (terugvallen op 5 tot 20% van de oorspronkelijke tijd) zijn zeker geen uitzonderingen. Ze komen bovendien uit een omgeving die goed georganiseerd was en ideaal geclusterd voor de uitgevoerde activiteiten. Indien dit niet het geval is, zijn de verschillen nog groter. Dus weg met SQL en leve de UTILITIES, tenminste in situaties vergelijkbaar met de aangehaalde voorbeelden.

# DOSSIER 8

## Just ONCE please ...

SQL statements - zowel statische als dynamische - moeten door de DB2 optimizer worden bewerkt, om een zo efficiënt mogelijk access-pad te kunnen bepalen. Een belangrijke rol is hier weggelegd voor de host-variabelen. Inderdaad, naast de inhoud van de variabele is ook de frequentie waarmee deze variabele aan veranderingen is onderworpen, zeer belangrijk. Immers, een eerste optimaal access-pad kan op basis van zich wijzigende host-variabelen aanleiding geven tot het suboptimaal worden van datzelfde pad bij herhaaldelijke query uitvoer.

Reeds lang beschikt DB2 hiertoe over een BIND parameter REOPT. Globaal geldt het volgende.

- REOPT(NONE) - dit is een nieuwe term in V8 die equivalent is met de bestaande NOREOPT(VARS) - geeft aan dat het standaardgedrag van DB2 van toepassing zal zijn. Concreet: optimisatie geschiedt op het moment van de BIND. Aanpassing van het access-pad op een later tijdstip gebeurt dus NIET meer.
- REOPT (ALWAYS) - dit is een nieuwe term in V8 die equivalent is met de bestaande REOPT(VARS) - geeft aan dat DB2 het toegangspad met host-variabelen of 'parameter markers' zal opnemen in de statement cache, en bij elke uitvoer opnieuw zal optimizeren. Dit geldt dus nog steeds voor zowel dynamische als statische SQL. Gewijzigde inhoud van host-variabelen kunnen aldus in rekening worden gebracht bij het bepalen van optimale toegangspaden.
- REOPT(ONCE) is nieuw voor V8 en geldt enkel voor dynamische SQL. Statements met 'parameter markers' worden slechts eenmalig geoptimiseerd: de eerste maal op het moment van de run, op basis van de dan voorhanden zijnde waarden van parameters. Het resultaat wordt opgenomen in de dynamic statement cache, en bij elke nieuwe run herbruikt.

Deze laatste optie is vooral interessant als enerzijds de inhoud van de variabele en/of parameter marker een belangrijke invloed kan hebben op het gekozen toegangspad, maar anderzijds niet of nauwelijks structureel wijzigt doorheen de tijd. Een voortdurende heroptimisatie, zeker voor relatief omvangrijke en complexe SQL statements (maximale lengte van een SQL statement is opgetrokken tot 2 MB), bij gebruik van REOPT(ALWAYS) verbruikt immers te veel CPU cycles.

Merk op dat REOPT(ONCE) ook nuttig kan zijn wanneer bij de eerste run zeker waarden worden meegegeven, waarvan men weet dat het resulterende accesspad 'goed' zal zijn.

*Tom Avermaete, ABIS*

# Realtime statistics voor DB2 for z/OS

*Kris Van Thillo (ABIS)*

'Lights out' - daar gaat het om in de wereld van de relationele databases! Downtime minimaliseren, DBA interventies vermijden, en dus, globaal gesproken, beschikbaarheid vergroten.

DB2 wordt daartoe SMART (System Managed and Ressource Tuned), of 'autonoom'. Oracle gebruikt hiervoor Adam - geschreven ADDM (Automatic Database Diagnostic Monitor) die tot doel heeft autonoom potentiële problemen te identificeren en te verhelpen.

Of dit alles kan worden gerealiseerd, hangt af van de beschikbaarheid van accurate, realistische en live-statistieken, die de huidige toestand van uw database omgeving weergeven. Deze statistieken vormen de input voor de boven geschetste analyse; als systeembeheerder gaat u zelf met SQL aan de slag om de voor u relevante statistische info op te vragen; of u schaft third-party tools aan die e.e.a. voor u op een visueel aantrekkelijke manier in kaart brengen. Of u gebruikt eventueel native door de vendor geleverde tools - onze vriend Adam hierboven is er een van. De statistieken zelf worden meestal opgeslagen in een afzonderlijke database: IBM heeft het over de RTS-database (Real-Time Statistics database), Oracle over de AWR (Automatic Workload Repository).

Merk op dat functioneel de bedoeling van beide databases niet dezelfde hoeft te zijn: waar DB2 zich in eerste instantie richt op statistieken ten behoeve van onderhoud en beheer, richt Oracle zich voornamelijk op statistieken die systeem- en applicatie-performance gericht zijn. Uiteraard 'in eerste instantie' en 'voornamelijk'.

In wat volgt wensen we onze aandacht te vestigen op de DB2 RTS-database.

## Inleiding

De bedoeling van de RTS-database is real-time statistieken op te slaan die de DBA kunnen bijstaan bij het bepalen van de noodzaak tot uitvoeren van onderhoud op DB2 objecten. Deze statistieken worden door DB2 in-memory bijgehouden, en op bepaalde momenten naar DB2 tabellen weggeschreven. Men speek van 'counters', een term die DB2 LUW specialisten bekend in de oren klinkt. Vermits het hier gaat om traditionele tabellen, kunnen deze met SQL worden benaderd.

Merk op dat het hier dus niet gaat om statistieken die door de DB2 optimizer zullen worden gebruikt om het optimale DB2 access-pad naar de data vast te stellen! De hier bedoelde statistieken worden trouwens ook niet in de traditionele cataloogtabellen DSNDBO6 bijgehouden.

## Initiële set-up

DB2 berekent en bewaart de hier bedoelde statistieken in memory voor volgende objecten:

- simple en segmented tablespaces: info wordt bijgehouden op het niveau van de betreffende tablespace;
- partitioned tablespaces: info wordt bijgehouden op het niveau van de individuele partities.

Om deze statistieken te kunnen bewaren, moet het DB2 systeem hierop worden voorbereid. Volgende procedure wordt uitgevoerd.

▪ Stap 1 - aanmaken van objecten. Een speciale RTS-database, met specifieke tablespaces, tabellen en indexen moet worden gecreeerd. Hiervoor zijn de nodige sample DDL-instructies beschikbaar in de SAMPLE-dataset. Een aantal, voornamelijk fysieke karakteristieken van deze objecten kunnen worden aangepast indien gewenst. Vaak wordt aangeraden deze objecten eveneens te associëren met een eigen, specifieke bufferpool. Het spreekt voor zich dat het aanmaken van deze objecten de nodige authorisaties vereist.

Voor een volledige beschrijving van beide hier aangehaalde tabellen verwijzen we naar de 'DB2 voor z/OS Utilities Guide and Reference'.

▪ Stap 2 - bepalen van het download/insert interval. Om de 30 minuten worden standaard de memory gegevens 'gedumpt' naar de in stap 1 aangemaakte objecten. Deze 'default' waarde kan bij installatie dan wel achteraf online worden aangepast.

▪ Stap 3 - starten van de RTS-database, via de traditionele START DATABASE commando's, in read-write mode.

De overzichtsfiguur op volgende pagina geeft een algemeen inzicht in de globale RTS-architectuur. Tabel sysibm.tablespacestats bevat een rij per tablespace (of tablespace-partitie); tabel sysibm.indexspacestats een rij per index (of indexspace-partitie). Voor de statistische tablespace zelf worden echte geen statistieken bijgehouden, zelfs niet indien ze het onderwerp uitmaken van utility operaties.

## Aanpassing van RTS counters

Welke acties geven aanleiding tot het aanpassen van de hier bedoelde RTS counters? Een kort overzicht - voor detailinformatie verwijzen we u graag naar de IBM manuals, bijvoorbeeld de traditionele 'DB2 voor z/OS Utilities Guide and Reference'.

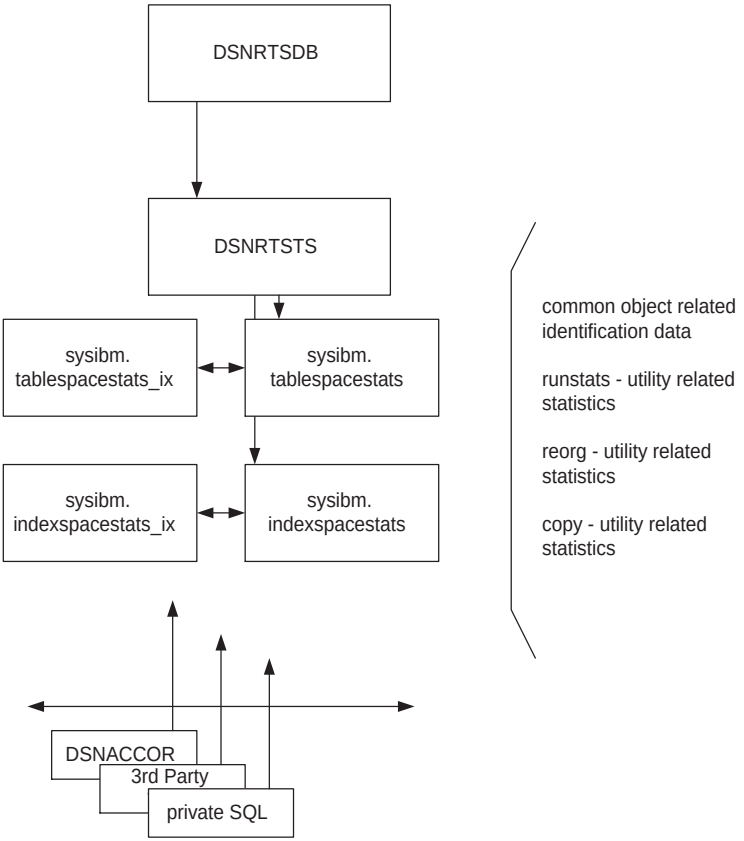
▪ alle traditionele UPDATE-, INSERT- en DELETE-operaties wijzigen (i.e. ophogen) RTS-statistieken; ROLLBACK-operaties wijzigen de annulatieoperaties. Voor belangrijke, speciale operaties zijn ook speciale statistieken beschikbaar (e.g. MASS-DELETE-operaties);

▪ LOAD, REORG, REBUILD INDEX, RUNSTATS UPDATE en COPY utilities hebben allemaal een specifieke impact op de RTS counters. Welke counters worden aangepast, en de impact van de aanpassing, is afhankelijk van de aard van het utility, en vaak ook van de moda-

liteiten waarmee het utility wordt uitgevoerd. Een aantal voorbeeldjes:

- bij het uitvoeren van een LOAD-operatie worden counters TOTALROWS, NACTIVE, PACE, EXTENTS aangepast;
- bij het uitvoeren van een REORG-operatie is de inhoud van bijvoorbeeld REORGDELETES en REORGINSERTS counters afhankelijk van de SHRLEVEL setting van de REORG zelf. Enkel bij SHRLEVEL REFERENCE/CHANGE zullen deze waarden niet '0' zijn (ten gevolge van het bijwerken van de shaduwtabellen op basis van de log na het switchen naar de schaduw).

*Overzichtsfiguur*




Objecten in RTS-structuur - de DSNRTSDB database, DSNRTSTS tablespace, en de boven aangehaalde tabellen tablespacestats en indexspacestats. Rechts: een overzicht van de typische counter informatie en de utilities die ze aanwenden.
Onderaan: een overzicht van de typische access technieken.

## Vatsleggen van counter-waarden in de RTS-database

Waneer worden de in-memory statistieken weggeschreven naar de DSNRTSDB database?

- bij het verstrijken van het ingestelde, boven besproken, tijds-interval;
- bij het uitvoeren van bepaalde specifieke utilities, als boven aangehaald. Dit is uiteraard niet het geval bij het uitvoeren van traditionele DML operaties;
- bij het stoppen van een tablespace;
- bij het stoppen van de DSNRTSDB database;
- bij het stoppen van het DB2-subsysteem in QUIESCE mode. Stoppen in FORCE mode schrijft geen statistische informatie.

## Uitlezen van data

Hoe kunnen we nu gebruik maken van deze statistische informatie? Eigenlijk is e.e.a. zeer eenvoudig: aan de hand van SQL moeten we gewoon de gewenste statistische gegevens uit de beide tabellen ophalen. En dan natuurlijk relateren en interpreteren - geen eenvoudige zaak.

We beschikken over een aantal alternatieve mogelijkheden.

▪ Schrijf eigen rapporten, op basis van eigen SQL queries. Zo is het niet ondenkbaar dat een aantal DBA's reeds gebruik maken van een aantal statistische gegevens die vandaag reeds in de 'gewone' cataloog worden opgenomen om te beslissen of bepaalde tablespaces moeten worden gereorganiseerd, bijvoorbeeld in de context van clustering. Deze praktijk kan aan de hand van de RTS counters worden uitgebreid. Een kleine greep uit de geboden mogelijkheden (counter namen in hoofdletters):

- gebruik de relatieve verhouding COPYUPDATEDPAGES en COPYCHANGES tot NPAGES om na te gaan of een tablespace moet worden ge-COPY-ed; en of dit al dan niet INCREMENTAL kan;
- gebruik STATSINSERT, STATSDELETES, STATSUPDATES om na te gaan hoeveel procent van een tabel gewijzigd is sinds het laatst uitvoeren van het RUNSTATS utility - misschien toch maar even opnieuw draaien?
- gebruik REORGUNCLUSTINS in verhouding tot TOTALROWS om na te gaan hoeveel rijen werden toegevoegd aan de tabellen in een suboptimale, niet geclusterde positie. REORGNEARINDREF en REORGFARINDREF duiden dan weer op het verplaatsen van rijen na update. Ook informatie met betrekking tot mass-deletes en pseudo-deletes kan worden geanalyseerd;

▪ Third-party tools. Een groot aantal vendors biedt al dan niet grafische implemenaties die het moeten mogelijk maken om de RTS counter gegevens op een aantrekkelijke manier te visualiseren. Vaak

wordt dit gecombineerd met uitgebreide analyse-tools, rapportage-mogelijkheden, etc.

▪ Gebruik een speciale DB2 stored procedure DSNACCOR. Deze stored procedure, aangeleverd door IBM, heeft tot doel:

- de inhoud van de RTS counters beschikbaar te maken:
- op basis van de inhoud, onmiddellijk en automatisch voorstellen te doen wat betreft onderhoud van de betreffende DB2-objecten.

DB2 UDB voor LUW specialisten herinneren zich in deze context ongetwijfeld het REORCHECK utility.

Met betrekking tot de DSNACCOR stored procedure even volgende opmerkingen:

- aan de hand van een hele reeks inputparameters kan worden aangegeven wat van de procedure wordt verwacht: zo moeten bijvoorbeeld de tresholds van toepassing worden aangegeven. Om de boven aangehaalde voorbeelden verder te zetten: Vanaf welk percentage page wijzigingen wenst u een COPY? En vanaf welk percentage moet dit een FULL COPY zijn? Voorts kan men aangeven welke analyses men wenst uit te voeren: analyses m.b.t. RUNSTATS, COPY, etc.;
- de formules aan de grondslag van DSNACCOR zijn publiek beschikbaar;
- de output van de DSNACCOR-procedure wordt ter beschikking gesteld via een result-set; deze moet op een traditionele manier worden uitgelezen.
- een speciale exceptietabel kan worden aangemaakt die door DSNACCOR zal worden gebruikt om het gegenereerde rapport aan te passen.

Info over gebruik exceptietabel? zie *Exceptietabel en RTS* op p. 13

## Besluit - Een meer gevorderde aanpak

De mogelijkheden door RTS counters geboden zijn natuurlijk omvangrijk. Een tweetal punten om over na te denken als besluit.

▪ De informatie opgeslagen in de RTS counters is natuurlijk 'current' informatie - historieken zijn niet beschikbaar. En dus maakt u een tweede tabel aan, die aan de hand van een job met bepaalde frequentie data van de RTS counters kopieert naar deze tweede tabel. Op basis van aldus beschikbare historieken kunt u het verleden analyseren EN de toekomst voorspellen. Plaats deze data op een DB2 UDB voor LUW en een omvangrijke set aan OLAP tools wordt beschikbaar - een onderwerp voor een volgende Exploring DB2.

▪ En dan hebben we natuurlijk nog de DSNUTILS stored procedure, die ons de mogelijkheid biedt DB2 utilities te starten vanuit stored procedures. Waarom DSNACCOR en DSNUTILS niet combineren in een eigen procedure? Als een bepaalde suggestie is gedaan (DSNACCOR) voer dan de relevante procedure met specifieke voorwaarden en opties uit (DSNUTILS). Lights out!

# CURSUSPLANNING MRT - JUN 2005

| | | |
|---|---|---|
| DB2 concepten | 375 EUR | 14/04 (L), 26/05 (W) |
| RDBMS concepten | 325 EUR | 14/03 (L),<br>11/04 (W), 09/05 (L), 13/06 (W),<br>20/06 (L) |
| Basiskennis SQL | 325 EUR | 15/03 (L),<br>12/04 (W), 10/05 (L), 14/06 (W),<br>21/06 (L) |
| DB2 for OS/390 basiscursus | 1050 EUR | 16-18/03 (L), 13-15/04 (W),<br>11-13/05 (L), 15-17/06 (W) |
| DB2 UDB basiscursus | 975 EUR | 13-15/04 (W), 27-29/06 (L) |
| SQL workshop | 700 EUR | 25-26/04 (W), 23-24/05 (L),<br>27-28/06 (W) |
| DB2 for OS/390 programmering voor gevorderden | 750 EUR | 10-11/03 (L), 09-10/06 (W) |
| Gebruik van DB2 procedural extensions | 375 EUR | 08/03 (L), 21/06 (W) |
| DB2 for OS/390: SQL performance | 1200 EUR | 11-13/04 (L), 23-25/05 (W) |
| DB2 UDB applicatieperformance | 400 EUR | 09/05 (W) |
| Database applicatieprogrammering met JDBC | 400 EUR | 14/03 (L), 18/04 (W), 09/05 (L) |
| Tunen van Java applicaties voor DB2 | 800 EUR | 24-25/01 (W), 21-22/04 (L) |
| Fysiek ontwerp v. relationele DB's | 750 EUR | 30-31/05 (L) |
| DB2 for OS/390 database administratie | 1600 EUR | 21-24/03 (W), 13-16/06 (L) |
| DB2 for OS/390 operations and recovery | 1500 EUR | 22-24/06 (W) |
| DB2 UDB DBA 1 - Kernvaardigheid | 1600 EUR | 25-28/04 (W) |
| DB2 UDB DBA 2 - Advanced tuning | 1200 EUR | 17-19/05 (W) |
| Extended SQL in DB2 | 400 EUR | 07/03 (L), 29/06 (W) |
| XML in DB2 | 400 EUR | 11/04 (L), 20/06 (W) |

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

## Exceptietabel en RTS

De TS exceptietabel heeft tot doel de gebruiker toe te laten ZELF informatie toe te voegen aan de result set van de output van de DSNACCOR stored procedure. Deze tabel bevat volgende kolommen:

- DBNAME, de database naam;
- NAME, de tablespace/indexspace naam;
- QUERYTYPE, een korte 'free text', die letterlijk wordt overgnomen in een van de kolommen van de result set van de stored procedure.

Concreet: op basis van een in deze tabel opgegeven combinatie databasenaam - tablespacenaam, dan wel databasenaam - indexspacenaam, wordt voor de overeenkomede combinatie in de betreffende outputkolom:

- 'YES' geplaatst als de QUERYTYPE kolom null is;
- de 'free text' geplaatst, als QUERYTYPE 'free text' bevat.

Hoe kunnen we deze techniek op een intelligente manier gebruiken? Twee mogelijkheden springen in het oog:

- voeg een afgesproken filtercriterium toe als 'free text': op basis daarvan kan bij het selecteren van relevante info van dit criterium worden gebruik gemaakt in de where condities;
- voeg in de 'free text' area gegevens toe die door zelf geschreven stored procedures kunnen worden gebruikt om run time opties te beïnvloeden.