ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 3 - NUMMER 5, JUNI 2005

OPEN CURSOR

Jaargang 3 zit er op - de zomermaanden staan voor de deur. Een periode van rust enerzijds, maar ook van diepgaand technisch onderzoek anderzijds, breken aan.
Een periode waar we DB2-technisch onze aandacht willen concentreren op een aantal relatief nieuwe aandachtsgebieden waar steeds meer interesse voor is. We denken aan content management, de problematiek van replicatie, en integratie overheen verschillende databronnen heen.
We houden u van al deze activiteiten natuurlijk op de hoogte. Via artikels in Exploring DB2. Maar ook via een aantal halve-dag specials, aangeboden in het najaar, die een aantal van deze onderwerpen op een gestructureerde wijze zullen uiteenzetten. Meer info op onze website!

Een verrijkende zomerperiode toegewenst - tot in september!

Het ABIS DB2-team.

IN DIT NUMMER:

- Over het belang van de mapping tussen relationele databases en objecten, in *Object-relational mapping met Hibernate.*
- En voor het laatst, in *DB2 utilities en applicaties - 4*, de mogelijkheid van remote utility uitvoer via DSNUTILS.
- In Dossier 8 bespreken we kort de mogelijkheid om *common table expressions* te definiëren.
- *Overzicht van de gepubliceerde artikels* in de 3e jaargang.
- *Cursusplanning augustus 2005 - november 2005.*

CLOSE CURSOR

Welke opties biedt DB2 als het gaat over datummanipulaties? En we beginnen een nieuwe reeks, afwisselend in deze publicatie en op het net - DB2 vs. Oracle vs. DB2.

Tot dan!

Object-relational mapping met Hibernate

Geert Vandevenne (ABIS)

In het verleden heeft ABIS reeds een speciaal Exploring DB2 nummer besteed aan de impact van Java op de DB2 professional. In dit nummer staan we even stil bij een framework dat als het ware een brugfunctie vervuld tussen enerzijds de database - bijvoorbeeld DB2 - en de object-georiënteerde applicaties.
We hebben hiervoor kunnen beroep doen op een ABIS -Java/WebSphere specialist van het eerste uur. Veel leesgenot!

O/R mapping

Hibernate is een van de O/R (Object-relational) mapping frameworks die vandaag de dag beschikbaar zijn om OO-Java applicaties te mappen op een relationele database. Dat dit geen eenvoudige klus is, bewijzen de vele pogingen om zulk een framework in elkaar te steken. Hoe gaat men overweg met hiërarchieën tussen entiteiten, die in OO-applicaties worden ondersteund, maar niet in relationele systemen? Hoe zit het met de multipliciteit van associaties tussen entiteiten? En wat aangevangen met navigabiliteit van associaties, iets waar OO-mensen mee geplaagd zitten, maar relationele mensen niet. Komen ook nog het probleem van locking, transacties, security, performance enz. om de hoek kijken. Hibernate lijkt in de lijst van frameworks een volwassen oplossing te worden. Tijd om de zaak eens van dichterbij te bekijken.

Positionering van Hibernate

Om Hibernate te kunnen positioneren binnen de Java-wereld, moeten we een onderscheid maken tussen technische frameworks, de componenttechnologieën en O/R mapping frameworks.

Technische frameworks benaderen relationele data op een nogal 'procedurele' wijze. De data worden opgehaald als resultsets die sequentieel worden doorgelopen, de SQL statements staan rechtstreeks in de Java-code. Voorbeelden hiervan zijn JDBC en SQLJ. JDBC is een onderdeel van de J2EE-standaard, SQLJ is dat niet.

De componenttechnologie die voor database-ontwikkelaars van belang is in het kader van J2EE, zijn de EJBs of Enterprise JavaBeans. Dit zijn server-side Java-componenten die binnen de context van een EJB-container draaien, en gebruik kunnen maken van de diensten die door deze container worden aangeboden. Naast persistentie van data, gaat het ook over diensten als ressource pooling, transaction management, life-cycle management, security, load balancing, transparant failover, ... Een nadeel aan deze techniek is dat componenten geen objecten zijn, en een aantal belangrijke mankementen

vertonen voor de echt OO-ontwikkelaar (oa. geen mogelijkheid tot inheritance en polymorfisme). EJB is onderdeel van de J2EE-standaard.

Een O/R mapping framework maakt het voor de Java-ontwikkelaar mogelijk om transparant Java-objecten weg te schrijven naar een database. Dit noemt men in de OO-wereld persisteren. Omdat er hier weer wel sprake is van 'echte' objecten, en geen componenten zoals met EJBs, wordt er hier soms nogal nostalgisch gewag gemaakt van POJOs (Plain Old Java Objects). Een voorbeeld van zulk een framework is JDO, dat geen onderdeel is van de J2EE-standaard, maar wel door Sun gestandaardiseerd. JDO mapt echter niet alleen naar relationele systemen, maar ook naar bv. OO-databases en XML-documenten. Daarnaast is er ook Hibernate. Dit is een open-source framework dat de afgelopen jaren bijzonder veel aandacht heeft gekregen in de Java-wereld omwille van zijn uitgebreide mogelijkheden voor mapping naar relationele systemen; en vormt dus het onderwerp van deze bespreking.

Hibernate onder de DB2-loep

Vragen die een DBA zich zou kunnen stellen zijn: moet het database-schema worden aangepast aan het Java-domeinmodel of andersom? Kunnen complexe datatypes in Java gemapt worden op database-types? Welke SQL wordt er gegenereerd en kunnen we die aanpassen? Komt het gebruik van statische SQL in aanmerking? Kunnen we iets doen aan de hoeveelheid SQL die traditioneel door OO-applicaties wordt gegenereerd? Welke locking-opties worden gebruikt en zijn deze gemakkelijk te sturen? Hoe wordt het transactioneel werken ondersteund?

Laten we bij het begin beginnen.

a. OO-domeinmodel vs. database-schema

Het database-schema moet in principe niet aangepast worden aan het OO-domeinmodel. Onder OO-domeinmodel wordt verstaan: 'het object-georiënteerd model van entiteiten in het probleemdomein'. We maken hierbij uiteraard abstractie van veranderingen aan het database-schema die nodig zijn om nieuwe functionaliteit in de applicatie te ondersteunen, maar dat staat los van de programmeertaal. Het is de verantwoordelijkheid van de mapping-laag om de object-relationele mismatch te overbruggen. Wat er concreet gebeurt is dat men het domeinmodel en het database-schema met elkaar gaat proberen verzoenen in een mapping-document. Dit document is een XML-file die beschrijft hoe de Java-klassen op de database-tabellen passen.

Een probleem dat zich stelt bij deze mapping is dat van de granulariteit: OO-domeinmodellen hebben de neiging om een hoge granulariteit te hebben (m.a.w. veel kleine objecten), terwijl database-schema's een lage granulariteit hebben: ze zijn gedeeltelijk gedenormaliseerd. Dit hoeft op zich geen probleem te zijn: Hibernate heeft constructies om dit probleem op te vangen.

Een typisch voorbeeld is dat van een bedrijven tabel met adresgegevens. In een OO-model zal men ervoor kiezen twee aparte klassen te maken: één klasse voor de bedrijven, en één voor de adressen. Via de <component> tag in het XML mapping-document kan men gemakkelijk deze twee klassen op één enkele tabel mappen. Een ander voorbeeld is het soort klassen die men zou kunnen catalogeren als complexe datatypes, en die men veel in OO-modellen tegenkomt, zoals Currency, VATNumber, PhoneNumber,... Ook hierin voorziet Hibernate in een mapping-constructie door UserType en ComplexUserType klassen aan te bieden waarop extensies kunnen gemaakt worden om de mapping te realiseren.

Wat betreft relaties tussen entiteiten, zit men soms met het probleem van de multipliciteit. Daar waar OO-applicaties totaal geen probleem kennen met many-to-many relaties, stelt dit probleem zich wel in de database: daar moet men gebruik maken van een tussentabel. Het mapping framework kan hier perfect mee overweg, en twee klassen met een many-to-many relatie worden zonder problemen op drie tabellen gemapt. Voor OO-mensen moeten we wel meegeven dat Hibernate geen managed associaties kent. Dit wil zeggen dat als je de link van object A naar object B legt, je manueel de link terug moet leggen van B naar A. Dit is iets waar database-mensen zich echter geen zorgen over moeten maken: Hibernate zorgt netjes voor die éne SQL die nodig is om de aanpassing in de database te realiseren.

Als laatste moeten we ook overerving aanhalen. In OO-applicaties hebben entiteiten hiërarchische relaties, die niet zomaar in een relationeel schema kunnen worden uitgedrukt. Hibernate ondersteunt hier de drie bekende oplossingen: een tabel per hiërarchie (i.e. één tabel waarin supertypes en subtypes samen worden gezet), een tabel per concreet subtype (i.e. alleen tabellen voor de subtypes, en geen voor het supertype) of een tabel per klasse (i.e. voor ieder super- en subtype een tabel). De te kiezen strategie hangt af van geval tot geval, en meerdere strategieën kunnen door elkaar gebruikt worden binnen één applicatie.

b. SQL

Vanuit Hibernate zijn er drie mogelijkheden om data op te vragen: Hibernate Query Language (HQL), Query By Criteria (QBC) en Query By Example (QBE). Bij elke methode wordt er dynamische SQL gegenereerd. Welke methode gebruikt wordt, maakt voor de DBA verder weinig verschil. Wat wel belangrijk is: kan deze SQL getraceerd worden op een eenvoudige manier, en kan ze dan verder getuned worden, m.a.w. is het mogelijk deze SQL te beïnvloeden? Op beide vragen is het antwoord ja. Traceren kan gemakkelijk in Hibernate door een switch om te zetten: alle SQL statements worden netjes gelogd en kunnen geanalyseerd worden. Tunen is ook mogelijk, en wel op verschillende manieren.

Manier één is gebruik maken van de ingebouwde fetch-strategieën. Een traditioneel probleem van OO-applicaties en mapping frameworks is dat er veel kleine queries naar de database gaan die telkens

zeer weinig data ophalen. Dit gebeurt als in de applicatie de objectgrafiek doorlopen wordt (object-hopping genoemd), en de informatie stuk voor stuk apart uit de database gehaald wordt. Hibernate heeft hier een intelligente oplossing voor bedacht: het is mogelijk om een default fetch-strategie te bepalen per associatie tussen klassen, lees: welke joins er moeten uitgevoerd worden indien er bepaalde data worden opgehaald. Deze defaults worden vastgelegd in het mapping-document. Bijvoorbeeld: telkens een persoon uit de persoonstabel wordt gehaald, worden ook zijn adresgevens mee opgehaald. Deze defaults kunnen indien nodig in de code weer overschreven worden. Zo wordt het mogelijk om op een intelligente manier, use-case per use-case, de aangepaste fetch-strategie te bepalen en dus ook de SQL die er gegenereerd zal worden. Per use-case zal men dus ook perfect weten welke SQL er uit de applicatie komt.

We moeten hierbij ook opmerken dat Hibernate zich bewust is van de database waar hij op draait: het is mogelijk een SQL-dialect in te stellen. Concreet: DB2-SQL voor een DB2 database. Het specificeren van het dialect gebeurt in de configuratie-file van Hibernate.

Indien bovenstaande niet voldoende zou blijken te zijn, bestaat er ook een mogelijkheid om rechtstreeks native SQL statements in het mapping-document te specificeren. Deze statements worden dan opgeroepen vanuit de Java-code, en Hibernate zorgt ervoor dat er objecten gegenereerd worden. Op deze manier is het mogelijk om van zeer DB2-specifieke SQL-eigenschappen gebruik te maken. Het zal geen verwondering wekken dat deze techniek bij Hibernate-mensen niet erg populair is.

Wat betreft statische SQL: dit wordt niet door Hibernate ondersteund. Hibernate maakt onderliggend gebruik van JDBC, en dat betekent dus dynamische SQL. Men zou natuurlijk kunnen overwegen om de SQL statements dan maar in stored procedures te steken, waar er, al dan niet via SQLJ, wèl gebruik gemaakt kan worden van statische SQL. Dit is een optie, maar stored procedures worden pas vanaf Hibernate versie 3 ondersteund (die pas sinds 08/05/2005 beschikbaar is in productie). Als men met een lagere versie van Hibernate werkt en toch absoluut wil gebruik maken van stored procedures, zal men moeten teruggrijpen naar JDBC. Het is mogelijk om aan Hibernate die JDBC-connectie te vragen. De applicatieprogrammeur wordt dan wel opgezadeld met de tijdrovende (en dus kostbare) taak om de resultset zelf te mappen op een objectgrafiek. Dit is duidelijk een pas achteruit vanuit OO-standpunt.

c. Transacties en locking

Hibernate voorziet in zijn eigen abstractie boven op het onderliggende transactiemechanisme. Dit kan zowel JDBC als JTA zijn, afhankelijk van de omgeving waarin de applicatie draait. Wat betreft transactie-isolatie ondersteunt Hibernate dezelfde oplossingen die JDBC voorziet, maar omdat Hibernate zich bewust is van de database waarvoor hij SQL genereert, kan er ook intelligent gebruik worden gemaakt van bijkomende mogelijkheden.

Wat betreft transactie-isolatie voorziet Hibernate in vier niveau's: read uncommited, read commited, repeatable read en serializable. Deze komen in DB2 overeen met respectievelijk uncommited read, cursor stability, read stability en repeatable read. U moet dus uitkijken dat u goed weet waarover u praat, want een repeatable read betekent in Hibernate heel wat anders dan in DB2! Deze isolatieniveaus worden in de Hibernate configuratie-file gespecificeerd, en vormen dus een globale optie die voor alle connecties en transacties dezelfde is.

Hibernate voorziet echter ook in de mogelijkheid om expliciet pessimistic locks te krijgen in de database, doordat Hibernate het onderliggende database-dialect kent. Dit betekent concreet dat Hibernate gebruik kan maken van het SELECT ... FOR UPDATE statement. Men moet wel in de applicatie aangeven dat men hiervan gebruik wil maken door een LockMode.UPGRADE te specificeren.

Een bijkomend probleem waar applicatieprogrammeurs soms tegenaan lopen, is het onderscheid maken tussen (fine-grained) databasetransacties, en (coarse-grained) applicatietransacties. Hibernate ondersteunt naast de database-transactie d.m.v. het Transaction object, ook het concept van applicatietransacties doorheen het Session object. Daarenboven maakt Hibernate ook gebruik van verschillende caching strategieën voor objecten door gebruik te maken van een first-level cache en optioneel ook een second-level cache. Deze onderwerpen liggen echter buiten het bestek van deze bespreking.

Tot slot

Er is in de Java community een hevige discussie aan de gang over 'het beste O/R mapping framework'. Het moge duidelijk zijn dat het zeer moeilijk is een winnaar aan te duiden aangezien veel verschillende aspecten meespelen in de eindbeoordeling. Toch is er veel opwinding ontstaan rond Hibernate de laatste jaren en verdient het zeker aandacht. Voor Hibernate spreken zijn vele uitgebreide mogelijkheden voor tuning van SQL en mapping. In het nadeel van Hibernate is dat het geen Sun-standaard is, maar een open-source project.

In de marge moet ook worden gezegd dat de EJB 3.0 die er zit aan te komen, veel van Hibernate geleerd heeft, en aldus opschuift in de richting van de O/R mapping frameworks. De mapping meta-data worden daar echter niet gespecificeerd in XML files, doch er wordt gebruik gemaakt van de nieuwe taalconstructies die met Java v5.0 zijn meegekomen: de meta-data worden als annotaties in de code zelf meegegeven. Maar dat zal stof zijn voor een volgend artikel.

DB2 utilities en applicaties - 4

Eric Venmans (ABIS)

Inleiding

In de vorige artikels uit deze reeks over utilities, is aangetoond dat ze zeer geschikt zijn voor 'zware' database-activiteiten. Wanneer we hun gebruik vergelijken met equivalente, geprogrammeerde SQL-oplossingen, vormen ze een aantrekkelijk alternatief. We hebben het daarbij in de eerste plaats over 'aantrekkelijk i.v.m. performance'. Implementatie is iets anders. Je moet als applicatieontwikkelaar meestal de hulp inroepen van een database administrator. Uiteindelijk zal men de gewenste JCL-instructies bij mekaar krijgen om programma- en utilities-uitvoeringen te integreren tot een job die efficiënt doet wat men ervan verwacht.

Er zijn echter situaties waarbij het gebruik van utilities niet meteen voor de hand ligt, ook al is dit wenselijk voor performance. We ontwikkelen bijvoorbeeld applicaties in een remote omgeving: Java-programma's ondersteund door een WebSphere Application Server.

Probleemomschrijving

Stel dat men vanuit de remote Java-omgeving geregeld massaal informatie moet verwerken in een centraal DB2-systeem. Kan dit nog gestuurd worden vanuit die remote Java-omgeving, terwijl de uitvoering gedaan wordt door DB2 utilities in het centrale systeem?

Nemen we eerst een concreet voorbeeld om van te vertrekken.

Een langlopende Java-applicatie heeft veel gegevens nodig die in een centraal DB2-systeem worden beheerd. Er zijn echter een reeks traditionele CICS-transacties die diezelfde gegevens gebruiken. Om interferentie van de Java-applicaties met de CICS-transacties te vermijden, besluit men de Java-toepassingen te laten werken op replicaties binnen het gebruikte DB2-systeem.

Dit is niet uitzonderlijk. Mainframe DBA's staan dikwijls wantrouwig t.o.v. alles wat met Java en een object-georiënteerde benadering te maken heeft. Of dit terecht is of niet laten we hier buiten beschouwing.

De Java-ontwikkelaars accepteren de oplossing (werken met replicaties) op voorwaarde dat ze zelf kunnen bepalen wanneer deze replicaties 'gerefreshed' worden, bijvoorbeeld bij het opstarten van de betrokken applicatie.

Pogingen tot oplossing

Een eerste mogelijkheid om te 'refreshen' is het gebruik van SQL vanuit de Java-omgeving. De JDBC interface, of beter nog de SQLJ-extensie ervan (leve 'static SQL', weet je wel) komt hiervoor in aanmerking. Om geen interactie te hebben met de bestaande CICS-transacties moet het wel gebeuren met de 'uncommitted read' optie. Geen echt probleem volgens de Java-ontwikkelaars. De tijd echter die nodig is om de replicatie beschikbaar te krijgen is te lang. Een alternatieve werking dringt zich op.

Het gebruik van een stored procedure om het werk te doen biedt geen soelaas. Door het gebruik ervan wordt de communicatie tussen de Java-omgeving en het DB2-systeem wel tot een minimum beperkt (één CALL van de remote procedure met in de procedure het lokaal uitvoeren van de SQL op mainframe), maar bij het repliceren is de communicatie niet het probleem. Immers via een beperkt aantal SQL-instructies worden gegevens intern binnen DB2 gekopiëerd vanuit de productietabellen naar de replicatiestructuren.

Het is het opnemen van deze informatie in de replicatietabellen dat de doorlooptijd bepaalt. Er zijn een reeks indexen voorzien die het verwijderen van de 'verouderde' informatie en het toevoegen van de nieuwe actuele informatie sterk vertragen. Het zijn weer de random 'in flight' indexaanpassingen die voornamelijk de verwerkingstijd bepalen. Dus het gebruik van utilities als oplossing (met sequentiële index-aanpassingen) duikt weer op. Alleen willen we dit nu sturen vanuit een remote omgeving.

Gebruik van DSNUTILS

Het uitvoeren van een DB2 voor z/OS utility vanuit een remote omgeving is niet evident. Wat hebben we in een ‘traditionele’ uitvoering nodig?

- Utility Control Statements.
  Deze duiden aan welke acties het systeem moet uitvoeren (LOAD, REORG, ...) en met welke objecten (TABLE(SPACE), PARTITION, ...).
- Job Control Statements (JCL).
  Deze verwijzen naar een specifiek DB2-systeem, naar de bibliotheken (partitioned datasets) met systeemmodules, naar tijdelijke workfiles en naar eventueel input en/of output datasets.

Een Java-ontwikkelaar kan zich wel iets voorstellen bij 'utitlity control statements' (acties met gegevens), maar JCL begrijpen is allicht te veel gevraagd. Bovendien moet men voor het uitvoeren van de JCL-instructies (= uitvoeren van de aangeduide utility) en het bekijken van het resultaat (via bv. SDSF) de nodige authorisaties en kennis hebben. En dit is nog maar een halve oplossing. We willen de uitvoering van de utilities integreren met aanverwante Java-activiteiten.

Een handige manier om ook dit laatste mogelijk te maken, is het gebruik van DSNUTILS. Het is een DB2 stored procedure die het uitvoeren van utilities ondersteunt. Als de systeemverantwoordelijken hun

DB2-omgeving configureren voor het uitvoerbaar maken van deze stored procedure, moeten de gebruikers (de Java-ontwikkelaars in ons voorbeeld) enkel nog de 'utility statements' kunnen formuleren. Dit is zeker doenbaar als ze in ruil de gewenste performance verbetering krijgen.

Systeemconfiguratie

Voor het uitvoeren van de DSNUTILS stored procedure is een WLM-omgeving nodig. WLM staat voor Work Load Management. De traditionele SPAS (Stored Procedure Address Space) komt niet in aanmerking. Bovendien moet de gebruikte WLM-omgeving aan een aantal vereisten voldoen:

- alle bibliotheken moeten APF authorized zijn (systeemverantwoordelijken weten dit wel te interpreteren);
- UTPRINT, DSSPRINT, SYSIN en SYSPRINT moeten als DD statement aanwezig zijn; SYSIN en SYSPRINT zijn datasets om tijdelijk de utility INPUT en OUTPUT te bevatten;
- de nodige authorisaties moeten toegekend worden.

De stored procedure zelf moet via SQL (een 'CREATE PROCEDURE' statement) gedefinieerd worden. Voor een verkorte weergave, zie voorbeeld - in totaal kunnen 41 parameters worden meegegeven, meestal ‘workfiles’ gerelateerd.

Voorbeeld

```
CREATE PROCEDURE DSNUTILS
  ( IN  UTILTY_ID VARCHAR(16),
    IN  RESTART VARCHAR(08),
    IN  UTSTMT VARCHAR(32704),
    OUT RETCODE INTEGER,
    IN  UTILITY_NAME VARCHAR(20),
    IN  RECDSN VARCHAR(54),
    IN  RECDEVT CHAR(08),
    IN  RECSPACE SMALLINT,
    .....
    IN  FILTRSPACE SMALLINT)
   DYNAMIC RESULT SET 1
   EXTERNAL NAME DSNUTILS LANGUAGE ASSEMBLER
   COLLID DSNUTILS
   PARAMETER STYLE GENERAL
   MODIFIES SQL DATA
   COMMIT ON RETURN NO
   WLM ENVIRONMENT TBD2WSYS

CALL SYSPROC.DSNUTILS(
   :UtilityID, :Restart, :UtilityStatements, :ReturnCode, :UtilityName,
   :RecordSN  ,:RecordDEVT  ,:RecordSPACE,   .....,
   :FilterDSN,:FilterDEVT,:FilterSPACE)
```

CALL van DSNUTILS

Het oproepen van de stored procedure gebeurt via een CALL waarbij variabelen worden doorgegeven die één op één matchen met de gedefinieerde parameters.

De belangrijkste variabelen zijn:

- UtilityID
  Unieke identificatie die gegeven wordt aan de utility. De identificatie moet maar uniek zijn binnen alle utilities die in uitvoering zijn (of na een onderbreking wachten op een restart).
- UtilityStatements
  De utility control statements worden gebruikt om het systeem duidelijk te maken welke acties uit te voeren met welke objecten.
- ReturnCode
  Dit is de hoogste returncode van de utility die wordt doorgegeven via deze stored procedure parameter. Als de procedure zelf problemen te melden heeft gebeurt dit via de SQLCODE die bij de CALL van de stored procedure hoort.

Voor het correct invullen van alle nodige parameters zal een applicatieontwikkelaar allicht beroep moeten doen op een DB2-specialist. Maar als éénmaal een test heeft uitgewezen dat alles naar behoren functioneert, heeft de ontwikkelaar een middel in handen om autonoom te beslissen over het uitvoeren ervan.

Controleren van de uitvoering

Het uitvoeren van DSNUTILS geeft normaal (als er geen onverwachte problemen opduiken) SQLCODE +466: PROCEDURE ... RETURNED ... QUERY RESULTS SETS. Immers, het rapport dat de utility genereert, komt terecht in een 'temporary table' die de naam SYSIBM.SYSPRINT heeft. De stored procedure opent voor deze table volgende cursor:

```
DECLARE SYSPRINT CURSOR
WITH RETURN FOR
SELECT SEQNO, TEXT
FROM SYSPRINT
ORDER BY SEQNO
```

Daardoor komt het resultaat van deze query (= het utility rapport) als resultset beschikbaar in de applicatie die de stored procedure heeft opgeroepen. Indien de ReturnCode van de utility duidt op een probleem bij het uitvoeren, kan de inhoud van het rapport gebruikt worden voor verder onderzoek en correctie. Hierbij zal men als Java-ontwikkelaar dikwijls de hulp kunnen gebruiken van een DB2 DBA.

Besluit

Het uitvoeren van een utility via de DSNUTILS is geen eenvoudige zaak. Systeem- en DBA-gerelateerde kennis moet gecombineerd worden met programmeertechnieken (CALL van een stored procedure). Bovendien zijn de meeste applicatieontwikkelaars niet meteen op de hoogte van de alternatieve werking en mogelijkheden van utilities, zeker niet als het Java-ontwikkelaars zijn zonder mainframe verleden. We moeten dus al met zware performance problemen zitten vooraleer we tot oplossingen komen zoals hier beschreven. Vroeg of laat zal DB2 zelf 'smart' genoeg zijn om waar nodig, utilities (of aanverwante middelen) in te schakelen ter vervanging van de 'gewone' SQL-verwerking. Tot zolang blijft het behelpen met de DSNUTILS een mogelijkheid.

DOSSIER 8

Common table expressions

DB2 voor z/OS versie 8 biedt de mogelijkheid één of meer common table expressions te definiëren in een SQL statement. Concreet gaat het om tijdelijke tabel constructies, aangemaakt in het SQL statement zelf, waarvan het SQL statement zelf gebruik maakt. Belangrijkste verschil met views of nested table expressions is dat de resultaatset van deze laatste 2 typisch wordt bepaald op moment van gebruik; de resultaatset van een common table expression wordt éénmalig, op het moment van specificatie, bepaald. De inhoud blijft dan ook constant gedurende de volledige levensduur van het SQL statement waarin deze expressie is aangemaakt.

Deze techniek wordt vaak gebruikt in de context van recursieve SQL, bill-of-materials, etc.

De syntax reeds geruime tijd beschikbaar in DB2 UDB voor LUW wordt hiertoe aangewend - WITH <common table expressie> FULLSELECT.

Kris Van Thillo (ABIS)

OVERZICHT VAN DE GEPUBLICEERDE ARTIKELS

CURSUSPLANNING AUG - NOV 2005

DB2 concepten	375 EUR	26/09 (L), 16/11 (W)
DB2 for OS/390, een totaaloverzicht	1700 EUR	05-09/09 (W), 19-23/09(L), 17-21/10 (W), 28/11-01/12 (W)
DB2 UDB, een totaaloverzicht	1625 EUR	17-26/10 (W), 28/11-02/12 (W)
RDBMS concepten	325 EUR	05/09 (W), 19/09 (L), 17/10 (W), 21/11 (L), 28/11 (W)
Basiskennis SQL	325 EUR	06/09 (W), 20/09 (L), 18/10 (W), 22/11 (L), 29/11 (W)
DB2 for OS/390 basiscursus	1050 EUR	07-09/09 (W), 21-23/09 (L), 19-21/10 (W), 30/11-02/12 (W)
DB2 UDB basiscursus	975 EUR	24-26/10 (W), 30/11-02/12 (W)
SQL workshop	700 EUR	29-30/09 (W), 10-11/10 (L), 14-15/11 (W)
Extended SQL in DB2	400 EUR	17/10 (L)
Gebruik van DB2 procedural extensions	400 EUR	18/10 (L)
DB2 for OS/390 programmering voor gevorderden	750 EUR	19-20/09 (W). 17-18/11 (L)
DB2 for OS/390: SQL performance	1200 EUR	12-14/10 (L)
Database applicaties bouwen met Java	2800 EUR	19-28/10 (L), 18-30/11 (W)
XML in DB2	400 eur	30/09 (W)
DB2 Content Manager	200 EUR	21/10 pm (L), 15/11 pm (W)
WebSphere Integrator	200 EUR	20/10 pm (L), 18/11 pm (W)
DB2 for OS/390 database administratie	1600 EUR	03-06/10 (W)
DB2 for z/OS in een Java omgeving	400 EUR	07/10 (W)
DB2 for OS/390 operations and recovery	1500 EUR	07-09/11 (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