ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 3 - NUMMER 4, MEI 2005

OPEN CURSOR

DB2 wordt steeds vaker gebruikt als de universele data store - alle data, in alle mogelijke formaten, moeten vanuit een veelheid van applicaties en aplicatietypes benaderbaar zijn. Dit vergt van de database beheerder en van de ontwikkelaar vaak kennis van een reeks nieuwe en/of gevorderde technieken om dit te bewerkstelligen. Technieken waar we u o.a. via Exploring DB2 op willen wijzen.
Zoals het gebruik van utilities als alternatief voor applicatieontwikkeling.
In deze editie besteden we - in diezelfde context - ook aandacht aan één van de vaak 'verborgen schatten' van DB2 op het UDB voor LUW platform: de OLAP analytische functies, waardoor rapportage een totaal andere betekenis krijgt; waardoor bestaande rapporten eenvoudiger, leesbaarder en efficiënter worden.
Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- Over het belang van utilities in applicatieontwikkeling - *DB2 utilities en applicaties - 3*.
- Nieuwe rapportagemogelijkheden in DB2 UDB voor LUW - OLAP analytische functie, in *OLAP functies in DB2 UDB LUW versie 7 en 8*.
- En een kort *Dossier 8* heeft het over de DB2 universele client.
- *Cursusplanning mei 2005 - juni 2005*.

CLOSE CURSOR

In het volgende nummer hebben we onder andere over het herstarten van DB2 applicaties.

Tot dan!

DB2 utilities en applicaties - 3

Eric Venmans (ABIS)

Inleiding

Wanneer we veel gegevens moeten toevoegen aan een DB2-tabel, kunnen we kiezen uit twee mogelijkheden: we gebruiken ofwel het SQL INSERT statement, ofwel de LOAD utility. Het gebruik van de LOAD utility laat op zijn beurt een aantal keuzes toe: REPLACE of RESUME en vrij recent ook SHRLEVEL NONE of CHANGE. De specificaties van het 'toevoeg'-proces bepalen voor een deel de keuze die we maken. Een andere beïnvloeder van de keuze is de performance. In dit artikel geven we een voorbeeld om aan te tonen dat het werken met de LOAD utility soms aantrekkelijke perspectieven biedt.

Het probleem: periodiek toevoegen van nieuwe gegevens

Stel, we moeten een proces automatiseren, waarbij dagelijks informatie, verzameld in files, moet toegevoegd worden aan bestaande gegevens in een tabel.

De informatie komt uit een 'gecontroleerde' omgeving. Dit wil zeggen dat de gegevens correct zijn. Hetgeen vooral gevraagd wordt, is het efficiënt toevoegen van de gegevens zonder de beschikbaarheid van de 'target table' voor andere processen sterk te verminderen.

De SQL INSERT oplossing

Als we opteren voor het werken met een programma waarin we het toevoegen coderen via SQL INSERTS, hebben we twee basisopties:

- ofwel kiezen we voor een zeer beperkte beschikbaarheid van de gegevens tijdens het proces, maar houden we dit proces zelf zo kort mogelijk,
- ofwel kiezen we voor een hoge beschikbaarheid, maar verhogen daardoor de duur van het proces.

1. Beperkte beschikbaarheid.

We vragen DB2 de betrokken tabel 'exclusief' te locken. Daarna 'inserten' we de gegevens (gelezen als input-records uit een file) zonder, of met een beperkt aantal synchronisatiemomenten. Afhankelijk van de werkomgeving is dit synchronisatiemoment een 'SQL commit', een 'IMS checkpoint', een 'CICS syncpoint', ... Door de 'exclusieve' lock beperken we de beschikbaarheid van de tabel (enkel nog toegankelijk via 'uncommitted read' acties), maar we beperken ook het werk van DB2. De communicatie met de LOCK manager (IRLM)

wordt tot een minimum beperkt. Bovendien reduceert men bijkomend het CPU-verbruik door het beperken van het aantal synchronisatiemomenten.

2. Hoge beschikbaarheid.

Als het proces, zelfs met minimaal CPU-verbruik, toch te lang de betrokken tabel isoleert, kunnen we een radicaal andere oplossing kiezen. We vragen kleine locks (page-locks voor fysiek sequentiële toevoegingen, row-locks voor fysiek random toevoegingen) en synchroniseren met een hoge frequentie.

Een test met deze oplossing geeft voor onze testomgeving aan dat we nu zowat een verdubbeling krijgen van de uitvoeringstijd. Uit verder onderzoek (andere locking, maar gelijke COMMIT-frequentie) blijkt 3/4 van de toename te wijten aan de verhoogde COMMIT-frequentie (1/100 inserts i.p.v. 1/1000 inserts). De rest kunnen we toeschrijven aan de verhoogde activiteiten i.v.m. locking.

Uiteraard zijn deze cijfers enkel exemplarisch. Ze worden beïnvloed door o.a.:

- uitvoeromgeving (batch, IMS restartable BMP, ...)
- aantal indexen
- clustering sequence
- volgorde van de inputgegevens
- volume van inputgegevens
- concurrerende processen
- systeeminstellingen (bufferpools, lockmax, ...)
- ...

De LOAD utility oplossing

Laten we nu even het alternatief bekijken, waarbij we het toevoegen laten uitvoeren door de DB2 LOAD utility.

1. Geen beschikbaarheid.

Als we de utility uitvoeren met SHRLEVEL NONE is er uiteraard tijdens het uitvoeren geen toegang mogelijk tot de betrokken tabel. Ook leesacties met 'uncommitted read' maken hier geen kans.

```
LOAD DATA RESUME YES LOG YES
SHRLEVEL NONE
INTO TABLE <target-table>
(.....)
```

De snelste manier van toevoegen is deze in bijgevoegde syntax boxen besproken. Ofwel kiezen we voor een RESUME YES LOG YES optie; ofwel voor een RESUME YES LOG NO. In dit laatste geval is men natuurlijk verplicht een IMAGE COPY te maken van de betrokken tablespace.

Opmerking hierbij: als de toegevoegde informatie relatief klein is t.o.v. de bestaande informatie, zal LOG YES efficiënter zijn dan LOG NO en de bijhorende IMAGE COPY.

```
LOAD DATA RESUME YES LOG NO
SHRLEVEL NONE
INTO TABLE <target table>
(.....)
```

De duur van het proces wordt in vergelijking met de SQL INSERTS sterk gereduceerd. In onze testomgeving vielen we terug op +/- 50% van de tijd die nodig was voor de applicatie met INSERTS, de exclusieve LOCK en het beperkt aantal synchronisatiemomenten. De tijdswinst komt door de manier waarop het systeem de INDEXEN aanpast (waar hebben we dit nog gehoord?).

De LOAD utitlity gaat namelijk de indexen niet 'in flight' aanpassen. Eerst worden de nieuwe rijen toegevoegd aan de betrokken tablespace. Tijdens dit proces noteert de utitlity alle relevante informatie i.v.m. indexen. Na het sorteren van deze index-informatie, wordt deze 'gemerged' met de bestaande indexen. Dit is een sequentieel proces, geen random zoals bij de meeste 'in flight' aanpassingen.

Locking zal uiteraard ook geen 'overhead' creëren omwille van SHRLEVEL NONE.

2. Hoge beschikbaarheid.

Sinds versie 7 kunnen we de LOAD utility ook uitvoeren met SHRLEVEL CHANGE.

```
LOAD DATA RESUME YES LOG YES
SHRLEVEL CHANGE
INTO TABLE <target-tabel>
(.....)
```

De variante met LOG NO komt hier niet in aanmerking omdat de te verwerken informatie moet kunnen 'geshared' worden met andere processen.

Wanneer we de performance onderzoeken van deze oplossing (SHRLEVEL CHANGE) merken we geen groot verschil met de geprogrammeerde versie (deze met de hoge beschikbaarheid). De utility gaat de input-records namelijk als rijen INSERTEN. Hierbij worden de indexen 'in flight' aangepast. Bovendien kunnen locking-conflicten (bijvoorbeeld t.g.v. 'lock-escalatie') de uitvoering van de utility onderbreken.

Het voordeel t.o.v. SQL INSERT is hier hoofdzakelijk herleid tot 'het niet moeten programmeren' van het process.

Andere scenario's

INSERTS zijn moeilijker te vervangen door utitlity-activiteiten dan bijvoorbeeld DELETES. We vertrekken namelijk van externe informatie die na opname 'geldig' moet zijn voor DB2 en voor de applicaties. Geldig zijn voor DB2 is niet het echte probleem. De LOAD utility kan de controles doen die voor DB2 belangrijk zijn:

- het kenmerk UNIQUE wordt automatisch gecontroleerd,
- via ENFORCE CONSTRAINTS wordt R.I. (Referential Integrity) gecontroleerd,

- via eveneens ENFORCE CONSTRAINTS worden CHECK constraints geëvalueerd.

Alleen TRIGGERS stellen een probleem. Zij worden enkel uitgevoerd tijdens een LOAD met SHRLEVEL CHANGE. Dit is de uitvoering die geen of nauwelijks performance winst levert.

Bovendien moet de informatie ook geldig zijn voor applicaties (of de gebruiker ervan). Hier horen ook controles bij die niet door DB2 gebeuren. Immers zelden worden alle relaties geïmplementeerd via R.I.. Zelfs als dit het geval is, is de DB2 controle dikwijls te beperkt. Bijvoorbeeld: DB2 kan wel nagaan of een PRODUCT dat besteld wordt, bestaat, maar als een status aanduidt dat het PRODUCT uit de verkoop is gehaald, wordt dit niet via R.I. regels opgevangen.

Dit betekent dat dikwijls tijdens het toevoegen, een programma extra controles moet uitvoeren. Werken via de LOAD utility wordt dan een heel stuk moeilijker.

Toch twee scenario's om aan te tonen dat met de nodige creativiteit (en kennis van het DB2-systeem) alternatieven te vinden zijn. Dit kan nodig zijn wanneer gewone INSERTS een te zware impact hebben op de uitvoeringstijd van een proces.

1. Scenario met voorwaardelijke toevoegingen.

Rijen worden via de LOAD utility toegevoegd aan de 'target table'. Met SHRLEVEL NONE gebeurt dit met de efficiënte manier van indexaanpassingen. De tabel zelf wordt voorzien van een indicator-kolom. Voor nieuwe rijen bevat deze een specifieke waarde (NULL, blanks, 0, ...). Via een VIEW kunnen deze nieuwe, niet-volledig gecontroleerde gegevens, afgeschermd worden bij 'gewoon' tabel gebruik. Een controleprogramma, een eigen geschreven intelligente variante op de CHECK DATA utility, kan on-line de nieuwe rijen opzoeken, controleren en vrijgeven (update van de niet-geïndexeerde indicator-kolom). Bij eventueel ongeldige nieuwe informatie, kan men afhankelijk van de vastgestelde 'fout', de rij in kwestie verwijderen, corrigeren of enkel rapporteren en 'ongeldig' laten.

Het controle-programma heeft een lage impact op het gebruik van de betrokken tabel. Enkel rijen die niet kunnen benaderd worden door de 'gewone' applicaties worden gelezen, aangepast (indicator-kolom) en uitzonderlijk (tenzij er veel foute invoer is) verwijderd. Enkel dit laatste heeft via indexaanpassingen enige impact op andere processen.

Nadelen van dit scenario:

- tijdelijk onbeschikbaarheid van de tabel tijdens het uitvoeren van de LOAD utility (SHRLEVEL NONE)
- niet onmiddellijk beschikbaar zijn van de toegevoegde rijen (enkel na update van de indicator-kolom).

2. Scenario met een tussentabel.

Rijen worden ingevoerd met de LOAD utility, maar in een eerste faze in een tussentabel (een die geen indexen nodig heeft).

Daar kan een controleprogramma de nodige acties uitvoeren om de informatie 'geldig' te maken. Om bij het verplaatsen van de gegevens naar de 'target table' geen 'in flight' aanpassingen te hebben van indexen kunnen we weer beroep doen op utilities. We kunnen hierbij een nieuwe mogelijkheid van de LOAD utility gebruiken, namelijk de 'INCURSOR'.

```
EXEC SQL
   DECLARE COPYDATA CURSOR
FOR
   SELECT  *
   FROM  <tussen-tabel>
   WHERE IndicatorKolom =
   'GELDIG'
ENDEXEC
LOAD DATA
RESUME YES LOG YES
SHRLEVEL NONE
 INCURSOR COPYDATA
 INTO TABLE <target table>
```

Het gebruik van de 'INCURSOR' kan eventueel vervangen worden door eerst de UNLOAD utility uit te voeren en de output hiervan als input te gebruiken voor een 'klassieke' LOAD.

Een andere optie is het uitbreiden van de INCURSOR mogelijkheden. Als de uit te voeren controles niet te complex zijn, kunnen we na de LOAD in de tussentabel, onmiddellijk de INCURSOR LOAD uitvoeren, bijvoorbeeld als volgt:

```
EXEC SQL
   DECLARE COPYDATA CURSOR
FOR
  SELECT  *
  FROM  <tussen-tabel> TS
  WHERE EXISTS (SELECT 1
        FROM <any-tabel> AT
      WHERE TS.KEY = AT.KEY
AND ...)
ENDEXEC
.....
```

Besluit

Bij het toevoegen van veel gegevens aan een lege tabel, wordt vrij snel naar de LOAD utility gegrepen. Dit gebeurt minder als het gaat om het toevoegen van informatie aan een tabel die reeds heel wat gegevens bevat. Eisen i.v.m. beschikbaarheid en uit te voeren controles beperken de inzetbaarheid van de LOAD utility. Als echter de performance in het gedrang komt, kan men mits enige creativiteit en wat extra inspanningen toch vaak terugvallen op de efficiënte werking van de DB2 utilities, in dit geval de LOAD utility.

OLAP functies in DB2 UDB LUW versie 7 en 8

Kris Van Thillo (ABIS)

Relationele database systemen worden ingezet voor een veelheid van applicatietypes: OLTP, batch, DSS en Data Warehousing. Vooral aan deze laatste categorie applicaties schenken database vendors steeds vaker extra aandacht. Naast ondersteuning van STAR en SNOWFLAKE optimisatie in database design, de mogelijkheid tot automatisch opslaan en beheer van redundante informatie, en een nog verder doorgedreven query optimisatie, zijn sinds DB2 UDB LUW versie 7 een aantal belangrijke nieuwe OLAP SQL features aan DB2 toegevoegd. Een kort inleidend overzicht.

Inleiding

OLAP SQL wordt door DB2 UDB LUW geëvalueerd op het moment dat de SQL SELECT instructie wordt uitgevoerd: dus juist voor de ORDER BY, na GROUP BY en HAVING. Concreet: de traditionele structuur en inhoud van de resultaat set, wordt nog steeds door de 'traditionele SQL' bepaald. Het verschil - nieuwe OLAP SQL - wordt dus toegevoegd in de SELECT instructie van het SELECT statement.

```
<analytische OLAP functie>
(<parameters>)
OVER
(<substatement>)
```

De syntax box beschrijft de algemene structuur van een OLAP SQL statement. Karakteristiek is de OVER-instructie: deze geeft aan dat we met een OLAP SQL statement te maken hebben. Voorts onderkennen we 3 belangrijke bestanddelen, in wat volgt even kort toegelicht.

- De eigenlijke functie. Sinds DB2 UDB LUW versie 7 onderkennen we 3 soorten functies. In eerste instantie hebben we de traditionele scalaire en groeps- of kolomfuncties, reeds lang in elke versie van DB2 gekend. Analytische functies daarentegen kunnen best worden beschreven als een combinatie van scalaire en kolom functies: ze genereren een waarde per rij in de resultaatset ('scalaire' karakteristiek); maar de waarde zelf wordt bepaald door de rij 'groep' waar ze toe behoort ('kolom' karakteristiek).

- Parameters. Afhankelijk van o.a. de gebruikte analytische functie zijn vaak een aantal parameters vereist (ook zonder parameters zijn haakjes verplicht!).

- Substatements bepalen uitvoermodaliteiten. We denken hierbij aan:

  - ORDER substatements - sorteren de data als vaak voorgeschreven door de gekozen analytische functie;

- PARTITION substatements - verdelen de rijen opgeleverd door het originele SQL statement in virtuele groepen of partities, waarop de analytische functies worden toegepast;
- WINDOW substatements - maken het mogelijk bepaalde rijen in groepen - WINDOWS - te verdelen, die dan onderling kunnen worden vergeleken, verschuiven (running totals), etc.

In wat volgt bespreken we een aantal belangrijke OLAP SQL functies. En leggen we een aantal boven besproken concepten even nader uit.

Ranking

RANKING functies laten toe aan bepaalde gegevens een 'rangorde' toe te kennen, op basis van een gevraagde sorteervolgorde, en dit binnen een bepaalde resultaatset - zie voorbeeld 1. Merk op:

- standaard maken alle opgeleverde rijen deel uit van de resultaatset; gebruik PARTITION BY om deze resultaatset in logische groepen te delen;
- het aangeven van een sorteervolgorde is verplicht; zowel ASC als DESC kan worden gespecificeerd, alsook hoe NULL waarden moeten worden behandeld;
- zowel RANK als DENSE_RANK worden ondersteund; bij de eerste functie bepaalt het aantal gelijk geordende rijen de rangorde van de logisch volgende rij.
- vermits het SQL ORDER BY statement nog moet worden uitgevoerd, is sorteren achteraf op een RANK column steeds mogelijk.

Voorbeeld 1: Ranking

```
select se_cno, year(sesdate) as jaar, count(*) as tot,
       rank() over (order by count(*)) as r1,
       dense_rank() over (order by count(*)) as r2,
      rank() over (partition by year(sesdate) order by count(*)) as r3
from sessions
where year(sesdate) in (1995, 1996)
group by se_cno, year(sesdate)
having count(*) > 20;

SE_CNO JAAR        TOT   R1    R2    R3
------ ----------- ----- ----- ----- -----
   399 1996        22     1     1     1
   475 1996        22     1     1     1
   142 1995        23     3     2     1
   528 1996        23     3     2     3
    13 1996        24     5     3     4
   565 1996        24     5     3     4
   682 1996        27     7     4     6
   401 1996        28     8     5     7
    13 1995        30     9     6     2

R1: maximale rangorde is 9 - we hebben immers 9 rijen in de set
R2: maximale rangorde is 6 - we hebben immers 6 distincte rijen in de set
R3: rangorde wordt bepaald door partitie 1995 dan wel 1996
```

Bovenstaand voorbeeld geeft ook het belang weer van PARTITIONS in deze context. Gelijk welk criterium kan nu immers worden gebruikt om, onafhankelijk van mekaar, de opgeleverde rijen in totaal onafhankelijke groepen in te delen. Onafhankelijk sorteren is dus ook steeds mogelijk.

Rijnummering

De functie ROW_NUMBER genereert een nummer voor elke opgeleverde rij. Strikt genomen ook een 'soort' rangorde dus. ORDER BY is niet verplicht - noch in de SQL SELECT instructie, noch in de OLAP functie. Gevolg is dan wel dat de rij NUMBERs dus totaal arbitrair worden toegekend - niet steeds (of beter: steeds niet) gewenst!

Voorbeeld 2: Rij nummers

```
select cno, cworktitle, cdur, ckind,
       row_number() over() as r1,
       row_number() over(order by cno) as r2,
       row_number() over(order by cdur) as r3,
       row_number() over(order by cworktitle) as r4,
       row_number() over(partition by cdur order by cworktitle) as r5
from tptcourses
where csequence is not null
and cdur > 0
and ckind = 'C'
and cworktitle like 'DB2%'
order by 1;

CNO    CWORKTITLE CDUR  CKIND  R1     R2    R3    R4    R5
------ ---------- ----- ----- ----- ----- ----- ----- -----
    13 DB2OVER    5.0 C        13      1    13     9     1
    92 DB2ADV     2.0 C         7      2     7     1     1
   290 DB2APPER   3.0 C        10      3    10     2     1
   451 DB2CON     1.0 C         1      4     1     4     1
   476 DB2DBA     4.0 C        12      5    12     5     1
  1061 DB2V7UP    1.0 C         5      6     5    12     5
  1067 DB2BAS     3.0 C        11      7    11     3     2
  1199 DB2TOP     5.0 C        14      8    14    11     2
  1209 DB2EXTSQL  1.0 C         2      9     2     6     2
  1211 DB2XML     1.0 C         6     10     6    14     6
  1217 DB2JAVATUN 2.0 C         8     11     8     8     2
  1225 DB2V8UP    2.0 C         9     12     9    13     3
  1242 DB2SPROC   1.0 C         4     13     4    10     4
  1299 DB2JAVADBA 1.0 C         3     14     3     7     3

R2: ordening van OLAP functie komt overeen met ordening van SELECT
R5: nummering volgens partitionering op cursus duur
```

Analytische functies

Afhankelijk van de beschouwde versie van DB2 UDB LUW zijn een aantal functies beschikbaar. De meeste van deze functies kunnen zowel als kolomfunctie dan als OLAP-functie worden gebruikt. Voor een volledig overzicht verwijzen we graag naar de 'DB2 UDB SQL Reference Guide'. Denk hierbij bijvoorbeeld aan volgende functies: CORRELATION, COVARIANCE, GROUPING, SUM, AVG, etc.

Windowing

PARTITIONS laten toe logische rijsets te definiëren, en dit op basis van de waarden van bepaalde kolommen. WINDOWING daarentegen biedt ons de mogelijkheid relatief ten opzichte van de huidige rij, vorige of volgende rijen in een bepaalde berekening mee te nemen. We hebben concreet 2 opties om deze WINDOWs te definiëren.

- a.d.h.v. de ROWS-instructie geven we aan dat we een aantal rijen voorafgaand aan en/of volgend op de huidige rij willen beschouwen als onderdeel van de WINDOW;
- a.d.h.v. de RANGE-instructie geven we aan dat we bepaalde rijen in een result set willen betrekken als de rij na toepassing van de range-berekening binnen de range zelf valt.

Zowel wat betreft de ROWS- als de RANGE-instructie kunnen een hele reeks specifieke grenswaarden worden aangegeven. Voor een volledig overzicht verwijzen we naar de 'DB2 UDB SQL Reference Guide'. Het spreekt voor zich dat in deze context de ORDER BY instructie een belangrijke invloed heeft op de wijze waarop de resultsets uiteindelijk worden opgebouwd.

Voorbeeld 3: Windows

```
select year(sesdate) as jaar, count(*) as tot,
       sum(count(*))
        over(order by year(sesdate) rows unbounded preceding) as sum_1,
       sum(count(*))
        over(order by year(sesdate) rows unbounded following) as sum_2,
       sum(count(*)) over() as sum_a0,
       sum(count(*))
          over(order by year(sesdate)
          rows between 2 preceding and 2 following) as sum_a1,
       sum(count(*))
          over(order by year(sesdate)
          range between 3 preceding and 3 following) as sum_a2
from tptsessions
where se_cno=13
and year(sesdate) between 1995 and 2005
group by year(sesdate);

JAAR        TOT  SUM_1   SUM_2  SUM_A0  SUM_A1  SUM_A2
------      ----  ----    ----    ----    ----    ----
   1995     30     30     135     135      74     74
   1996     24     54     105     135     100     74
   1998     20     74      81     135     115    100
   2001     26    100      61     135      98     74
   2003     15    115      35     135      81     61
   2004     13    128      20     135      61     61
   2005      7    135       7     135      35     35

SUM_A1: som van huidge rij waarde, 2 voorgaande en volgende waarden
SUM_A2: som van huidge rij waarde, voorgaande en volgende als binnen
set jaar + 3
```

Besluit

Analytische OLAP-functies bieden een breed scala aan bijkomende rapportage mogelijkheden: mogelijkheden, waarvoor vroeger vaak gevorderde en dure bijkomende tools vereist waren. En nu zijn een aantal dus standaard onderdeel van DB2 UDB LUW - zeker de moeite om naar te kijken!

Maar inderdaad, op dit moment zijn deze features nog niet ondersteund op DB2 UDB for z/OS. Een specifieke planning wanneer dit wel het geval zou zijn is niet bekend.

DOSSIER 8

Universele DB2 UDB Client - Common client

In het verleden was het inderdaad zo dat verschillende DB2 database clients - DB2 UDB for LUW dan wel DB2 UDB for z/OS - onderling verschillende database access protocols hanteerden voor access naar de DB2 UDB server. Voor elke protocol zijn dan op hun beurt een aantal accessroutines en -methoden gedefinieerd.

DB2 UDB versie 8 implementeert de Open Group Technical Standard DRDA Version 3 - de zogenaamde 'common client' infrastructuur. Dit komt oa. tot uiting als volgt:

- een C - common client voor ODBC
- een Java common client voor SQLJ en JDBC
- een gemeenschappelijke en geïntegreerde administratieve client

Belangrijk voordeel van deze integratie is uiteraard de toegenomen portabiliteit: DB2 UDB applicaties moeten nu immers met een minimum aan inspanningen kunnen worden gemigreerd van het ene naar het andere DB2 platform.

Koen De Backer, ABIS

CURSUSPLANNING MEI - JUN 2005

DB2 concepten	375 EUR	26/05 (W)
DB2 for OS/390, een totaaloverzicht	1625 EUR	09-13/05 (L), 13-17/06 (W)
DB2 UDB, een totaaloverzicht	1625 EUR	20-24/06 (L)
RDBMS concepten	325 EUR	09/05 (L), 13/06 (W), 20/06 (L)
Basiskennis SQL	325 EUR	10/05 (L), 14/06 (W), 21/06 (L)
DB2 for OS/390 basiscursus	975 EUR	11-13/05 (L), 15-17/06 (W)
DB2 UDB basiscursus	975 EUR	27-29/06 (L)
SQL workshop	700 EUR	23-24/05 (L), 27-28/06 (W)
Gebruik van DB2 procedural extensions	350 EUR	21/06 (W)
DB2 for OS/390: SQL performance	1200 EUR	23-25/05 (W)
DB2 UDB applicatieperformance	400 EUR	09/05 (W)
Database applicatieprogrammering met JDBC	400 EUR	09/05 (L)
Fysiek ontwerp v. relationele DB's	700 EUR	30-31/05 (L)
DB2 for OS/390 database administratie	1600 EUR	13-16/06 (L)
DB2 for OS/390 operations and recovery	1500 EUR	22-24/06 (W)
DB2 UDB DBA 2 - Advanced tuning	1200 EUR	17-19/05 (W)
DB2 up-to-date: extended SQL in DB2	400 EUR	29/06 (W)
DB2 up-to-date: XML in DB2	400 EUR	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