ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 2 - NUMMER 9, JUNI 2004

OPEN CURSOR

Dit is Exploring DB2 nr. 9, het laatste nummer van deze jaargang. De zomermaanden zijn immers in aantocht!
En dus ook: 2 maanden relatieve rust. Relatief, want DB2 staat natuurlijk niet stil en ABIS ook niet.
Inderdaad, de komende maanden staan voor ABIS extra in het teken van 'integratie'.
Integratie met andere RDBMS systemen, want DB2 is nog meer dan vroeger slechts één van de vele databases die in uw omgeving worden gebruikt. Maar ook integratie van DB2 met andere computersystemen - via MQSeries, IMS, .Net, XML. En we gaan nog een stap verder met de DB2 UDB Integration Server; en WebSphere Portal komt ook aan bod!
Dit alles vormt de basis voor een aantal artikels, die u in jaargang 3 van Exploring DB2 mag verwachten.
Prettige zomermaanden - tot in september!

IN DIT NUMMER:

CLOSE CURSOR

We beginnen een nieuwe reeks over DB2 en integratie: verschillen tussen belangrijke RDBMS systemen komen in detail aan bod. Onder andere.

Tot dan!

Extended SQL: onverwacht (?) efficiënt!

Eric Venmans (ABIS)

Inleiding

In de jaren '80 werden de eerste versies van DB2 geïntroduceerd. Sindsdien is er functioneel weinig toegevoegd aan de SQL-vraagtaal (lees: het SELECT statement). De SQL in DB2 heeft vooral een evolutie meegemaakt in de manier van formuleren, niet in het toevoegen van functionaliteit. Nogmaals, het gaat hier over het opvragen van informatie, niet over aanpassingen en niet over performance. In het oogspringende nieuwigheden waarover het hier wel gaat, zijn de OUTER JOIN (versie 4), de NESTED TABLE EXPRESSION (ook versie 4 en later herdoopt tot NESTED SELECTS) en een explosieve toename van SCALAR FUNCTIONS (versie 6 en 7). Enkel deze laatsten verleggen de grenzen van SQL (minder programmatie), maar zijn voor performance dan weer geen goede zaak.

OUTER JOINS en NESTED SELECTS zijn schijnbaar bedoeld om moeilijkere formuleringen te vereenvoudigen. Met OUTER JOINS worden UNIONS van verschillende SQL statements overbodig, met NESTED SELECTS worden dan weer een aantal VIEWs overbodig. Nu blijkt bij nader toezien dat ook de performance van deze nieuwe formuleringen erg gunstig beïnvloed wordt. We bespreken hier één voorbeeld.

Probleemstelling

Een programma moet op vraag van gebruikers een rapport maken van een 'reeks' producten die het bedrijf aanbiedt, aangeduid via een productklasse. Bij elk product uit deze klasse moet aangegeven worden hoeveel stuks er totaal in voorraad zijn. Dit laatste is telkens de som van de voorraden in verschillende verkoopspunten.

Figuur 1: Tabeloverzicht

```
Create TABLE Producten
(ProductID         char(6)     not null,
 PRKlasse          char(2)     not null,
 .....
 Primary Key (ProductID)

Create TABLE Voorraden
(VR_ProductID      char(6)     not null,
 VR_VerkoopPlaats  char(4)     not Null,
 VRAantalStuks     integer     not null with default,
 .....
 primary key (VR_ProductID, VR_VerkoopPlaats),
 foreign key FK1 (VR_ProductID)
 references Producten on delete cascade)
```

Voor het maken van het rapport, moet de SQL in het betrokken programma zich richten tot twee tabellen, beschreven in figuur 1.

Eerste traditionele oplossing

Onze applicatie kan zonder veel omwegen via een query aan de gevraagde informatie geraken - zie voorbeeld 1.

Voorbeeld 1

```
Select ProductID, PRNaam, PRBeschrijving, SUM(VRAantalStuks)
from  Producten, Voorraden
where ProductID = VR_ProductID
and PRKlasse = :WS-PRKlasse
GROUP BY ProductID, PRNaam, PRBeschrijving
```

Via EXPLAIN komen we te weten dat DB2 voor deze query een kostenschatting maakt van 68146 ms processor tijd. Dit getal komt uit de DSN_STATEMNT_TABLE (zie vorige nummers Exploring DB2).

De relatief hoge kost kan men verklaren na een analyse van het gebruikte toegangspad. DB2 gaat eerst 'joinen'. Daardoor komt elk geselecteerd product gemiddeld *n* maal voor in het gemaakte tussenresultaat. De *n* staat voor het gemiddeld aantal voorraad-rijen per product. In de testomgeving waar onze cijfers vandaan komen is *n* ongeveer 10. In een tweede fase voert DB2 de GROUP BY uit op dit tussenresultaat. Alle rijen met dezelfde productinformatie (gemiddeld een 10-tal) worden gegroepeerd. Het systeem berekent daarbij de SOM van het aantal stuks in voorraad.

Tweede oplossing: schijnbare verbetering

Men kan het maken van een tussenresultaat voorkomen door de query op te splitsen - zie voorbeeld 2.

Voorbeeld 2

```
Select ProductID, PRNaam, PRBeschrijving
from  Producten
where PRKlasse = :WS-PRKlasse

Select SUM(VRAantalStuks)
from Voorraden
where  VR_ProductID = :WS-ProductID
```

De eerste query geeft alle gevraagde producten. Bij elke fetch (het lezen van telkens één geselecteerd product) wordt de identificatie van dit product gebruikt om met een aparte query de voorraden-tabel te ondervragen. Dit levert telkens de gevraagde totale voorraad.

De geschatte kosten liggen nu heel wat lager. Nog 7502 ms voor het selecteren van producten, en nauwelijks iets (1 ms) voor het telkens opzoeken van de totale voorraad. Bij het echt uitvoeren van het programma en het meten van de daarbij gebruikte tijd, blijkt deze tweede oplossing echter minder snel. Reden: in de geschatte tijd wordt enkel rekening gehouden met de interne verwerkingstijd van DB2, niet met de communicatie tussen de applicatieomgeving en het DB2-

systeem. Er wordt ook geen rekening gehouden met de 'overhead' die gepaard gaat met het initialiseren van de DB2-omgeving voor het uitvoeren van elke query. In de tweede oplossing gaat dit zwaar doorwegen: voor elk geselecteerd product wordt een aparte query uitgevoerd op de voorradentabel.

Derde oplossing: betere performance

Men kan het maken van het tussenresultaat (zie allereerste oplossing) ook voorkomen zonder de query op te splitsen. We maken hierbij gebruik van de NESTED SELECT constructie - voorbeeld 3.

Voorbeeld 3

```
Select ProductID, PRNaam, PRBeschrijving, VRTotaalAantalStuks
from Producten,
     (Select VR_ProductID, SUM(VRAantalStuks) as VRTotaalAantalStuks
      from Voorraden
      GROUP BY VR_ProductID) as X
where ProductID = X.VR_ProductID
and PRKlasse = :WS-PRKlasse
```

Deze query geeft volgend, door DB2 berekend resultaat: 20528 ms.

De NESTED SELECT wordt eerst uitgevoerd. Het tussenresultaat is hier in omvang beperkt: door het groeperen van voorraden per product. Pas in een tweede fase wordt er 'gejoined': per product is er slechts één gerelateerde rij in het tussenresultaat. De interne processing is identiek aan wat in de vorige, 'geprogrammeerde' oplossing gebeurt, alleen is het aantal queries en de communicatie tussen programma en DB2 veel lager. Bij elke fetch komt samen met een geselecteerd product meteen ook de gevraagde totale voorraad binnen. Bij uitvoer stellen we ook vast dat het gebruik van de NESTED SELECT voorlopig de beste performance geeft.

Nadeel blijft hier het groeperen van alle voorraden (in de eerste fase), terwijl dit strikt genomen enkel voor de gevraagde klasse moet gebeuren. Ook dit laatste kan afgedwongen worden.

Voorbeeld 4

```
Select ProductID, PRNaam, PRBeschrijving, VRTotaalAantalStuks
from  Producten P,
     TABLE(Select VR_ProductID, SUM(VRAantalStuks) as VRTotAantalStuks
           from Voorraden
           where VR_ProductID = P.ProductID) as X
where PRKlasse = :WS-PRKlasse
```

Finale oplossing

Deze query geeft het beste resultaat. DB2 komt tot een berekende tijd van 9021 ms. Een echte uitvoering bevestigt deze berekening: het gevraagde rapport wordt nog sneller geleverd.

Door de 'correlatie' worden in de finale oplossing alleen nog maar voorraden van geselecteerde producten opgezocht en behandeld.

Extra complicatie

Op een bepaald ogenblik beslist men om ook producten te verkopen zonder deze in voorraad beschikbaar te houden. Ze worden enkel doorverkocht en gaan rechtstreeks van leverancier naar klant. Concreet betekent dit dat er rijen in de productentabel terecht komen, waarvoor geen 'dependent' rijen bestaan in de voorradentabel.

Veronderstel nu even dat we bij de derde oplossing zijn gestopt met ons optimalisatieproces. We moeten deze oplossing aanpassen om ook de nieuwe producten zonder voorraden te zien. Via de GROUP BY zien we immers alleen maar geregistreerde informatie, nooit ontbrekende. De aangepaste versie ziet er uit als volgt - zie voorbeeld 5.

Voorbeeld 5

```
Select ProductID, PRNaam, PRBeschrijving, VRTotaalAantalStuks
    from  Producten,
          (Select VR_ProductID, SUM(VRAantalStuks) as VRTotAantalStuks
            from Voorraden
            GROUP BY VR_ProductID) as X
    where ProductID = X.VR_ProductID
    and PRKlasse = :WS-PRKlasse
UNION ALL
Select ProductID, PRNaam, PRBeschrijving, 0 as VRTotAantalStuks
from  Producten P
where PRKlasse = :WS-PRKlasse
and not exists (Select  1
                from Voorraden
                where VR_ProductID = P.ProductID)
```

Door het toevoegen van deze subquery wordt de select meteen aanzienlijk duurder (36115 ms). We gebruiken hier nochtans de goedkoopste oplossing: een gecorreleerde subquery is meestal - en zeker in het gebruikte voorbeeld - merkelijk goedkoper dan de niet-gecorreleerde variant. We zijn echter wel uitgegaan van de derde oplossing. Voor de finale oplossing moeten we niet eens iets wijzigen. De gecorreleerde nested select geeft immers automatisch een 0 (zero) als er geen corresponderende rijen gevonden worden. Deze wordt niet duurder en is (hier toevallig of niet) bestand tegen een wijziging in de afspraken op databaseniveau (parent-rijen voortaan ook zonder dependents).

Besluit

Strikt genomen kunnen we ons probleem oplossen met een 'klassieke' SQL-formulering (zie eerste traditionele oplossing). Door nieuwere, 'geavanceerde' formuleringen te gebruiken kunnen we echter het toegangspad merkelijk verbeteren. Voor applicaties die veelvuldig worden uitgevoerd, is dit zeker gewenst.

Je zou kunnen verwachten dat de DB2 optimizer spontaan het optimale toegangspad kiest, maar voorlopig gebeurt dit niet. In afwachting moeten we zelf zoeken naar een optimale formulering. Rest de vraag: zijn applicatieontwikkelaars en/of DBA's voldoende vertrouwd met de nieuwe SQL-mogelijkheden om dit soort performance problemen 'correct' te behandelen?

EXTRA EDITORIAAL

Het blijft een goede praktijk cursoren zo laat mogelijk te openen, en zo snel mogelijk opnieuw te sluiten. Hetzelfde is waar voor de 'Open cursor' en 'Close cursor' stukjes aan het begin van elke Exploring DB2!

En toch wensen we nu wat ruimte te besteden aan een extra editoriaal.

Dit is het einde van de tweede jaargang 'Exploring DB2'. Het ideale moment om even stil te staan bij onze huidige werkwijze; en om na te denken over hoe we e.e.a. in de derde jaargang willen aanpakken. Ons doel blijft hetzelfde - u regelmatig een publicatie bieden die inhoudelijk op een hoog kwalitatief niveau staat. Een kwaliteit die u van ons gewoon bent geworden. Maar toch ook met oog voor een aantal concrete realiteiten waar ABIS mee wordt geconfronteerd.

Realiteiten die te maken hebben met de tijd die door ABIS medewerkers aan deze publicatie wordt besteed. Tijd voor het samenstellen van de artikels, de omvang van het finale 'editeer'-werk, en de coördinatie van de nummers. Om er maar een paar op te noemen. En natuurlijk is er ook het aspect 'kosten'.

Rekening houdende met onder andere deze factoren, hebben we beslist in de derde jaargang van 'Exploring DB2' een aantal wijzigingen door te voeren.

a) Het aantal nummers zal worden beperkt tot 5 per jaargang, i.p.v. de huidige 9 nummers. We hebben gekozen voor een nummer in september, november, januari, maart en mei;
b) Naast de artikels opgenomen in de papieren publicatie, zullen er extra artikels verschijnen op onze website. Deze extra artikels zullen in de gedrukte publicatie telkens ingeleid worden, met een verwijzing naar de volledige versie op www.abis.be.
c) De 'Dossier' artikels blijven uiteraard behouden;
d) Eens per jaargang blijven we een extra dik themanummer voorzien!

U merkt het: het is nog steeds onze bedoeling om u ook in de volgende jaargang verder op de hoogte te houden van de evolutie die DB2 doormaakt. Met specifieke aandacht voor beheer en applicatieontwikkeling.

En natuurlijk staan we open voor elk voorstel, elk idee dat volgens u, aan het nut en de efficiëntie van deze publicatie kan bijdragen.

Tot slot. Voor dit alles kunnen we uw hulp nog steeds gebruiken. Graag trekken we extra tijd uit, om ook uw artikel in 'Exploring DB2' te publiceren.

Het ABIS DB2-team.

DOSSIER 8

Meer stage 1 in DB2 V8 ...

Een algemene design-regel luidt dat datatypes van variabelen en kolommen moeten overeenstemmen. Op die manier vermijdt men een stage-2 predikaat in queries waardoor het gebruik van een index mogelijk wordt.

Maar bij het ontwikkelen van nieuwe applicaties wordt het steeds moeilijker deze regel te handhaven, zeker wanneer het gaat om 'nieuwerwetse' applicaties zoals deze geschreven in Java. We staan voor een dilemma: DB2 houdt wel van vaste lengte karaktervelden en Java houdt niet van vaste lengte karaktervelden (er bestaan alleen strings van variabele lengte).

In DB2 V8 worden predikaten met niet overeenstemmende datatypes stage 1 predikaten. In bepaalde gevallen zal een index gebruikt kunnen worden. Dit komt ook joins op kolommen met verschillende datatypes ten goede.

Volgende predikaten varen er wel bij - een paar voorbeelden:

kolom = 'Peeters'

kolom >= :namevar

kolom between '100' and '200'

tabel1.kolom = tabel2.kolom (de operatie <> geeft wel een stage-1 predikaat maar het is niet indexeerbaar).

Wanneer kan DB2 nu een index gebruiken? Wanneer bij het casten van de geïndexeerde kolom naar het andere datatype GEEN gegevens veloren gaan, voorbeeld: char(15) _kolom < :varchar(10)_var.

Ook de numerieke datatypes doen mee: een incomes-kolom met datatype decimal(9,4) kan vergeleken worden met een real of float host variabele. De precisie moet kleiner of gelijk zijn aan 15. Wat betreft string en date/time/timestamp vergelijkingen: er kan een index gebruikt worden wanneer de kolom het datum-tijd datatype heeft, omgekeerd niet.

Katrien Platteborze (ABIS)

Een systeemvergelijking - DB2 for LUW vs. DB2 for z/OS and OS/390

Kris Van Thillo (ABIS)

Vaak wordt DB2 UDB for z/OS and OS/390 (vanaf hier DB2 z/OS genoemd) en DB2 UDB for Linux, Unix en Windows (vanaf hier DB2 LUW genoemd) over dezelfde kam gescheerd. Alsof ze eigenlijk quasi-identiek zijn. In wat volgt doen we een poging om op systeemniveau een aantal verschillen op een rijtje te zetten.

SUBSYSTEEM of INSTANCE

In DB2 z/OS spreekt men over SUBSYSTEEM; DB2 LUW heeft het over INSTANCEs. Het gaat inderdaad in beide gevallen over één enkele, actieve en onafhankelijke DB2-omgeving, bekeken vanuit het standpunt van de traditionele systeemprogrammeur. Maar inhoudelijk en structureel zijn beide systemen toch wel zeer verschillend. Figuur 1 geeft een globaal overzicht van een aantal verschillen. Van een aantal elementen - clustering en partitioning, of het eigene van een gedistribueerde architectuur - wordt in dit artikel even abstractie gemaakt.

En toch vallen een aantal belangrijke gelijkenissen op.

- Het cruciale belang van de DB2-cataloog, en de relatie tussen het 'niveau' waarop de cataloog beschikbaar is, en het 'niveau' waarop de user aanlogt.
- Het belang van een aantal memory-allocaties voor efficiënt databeheer: o.a. bufferpools en extended storage/hyperpools.
- Het belang van integriteit en consistentie - dual logging, en de bijhorende administratieve structuren.
- Het belang (en de aanwezigheid van) configuratiefiles: dbm cfg, db cfg, DSNZPARM.

Belangrijkste conslusie is echter het verschillende hiërarchische niveau waarop al deze elementen beschikbaar zijn op beide systemen.

USERS

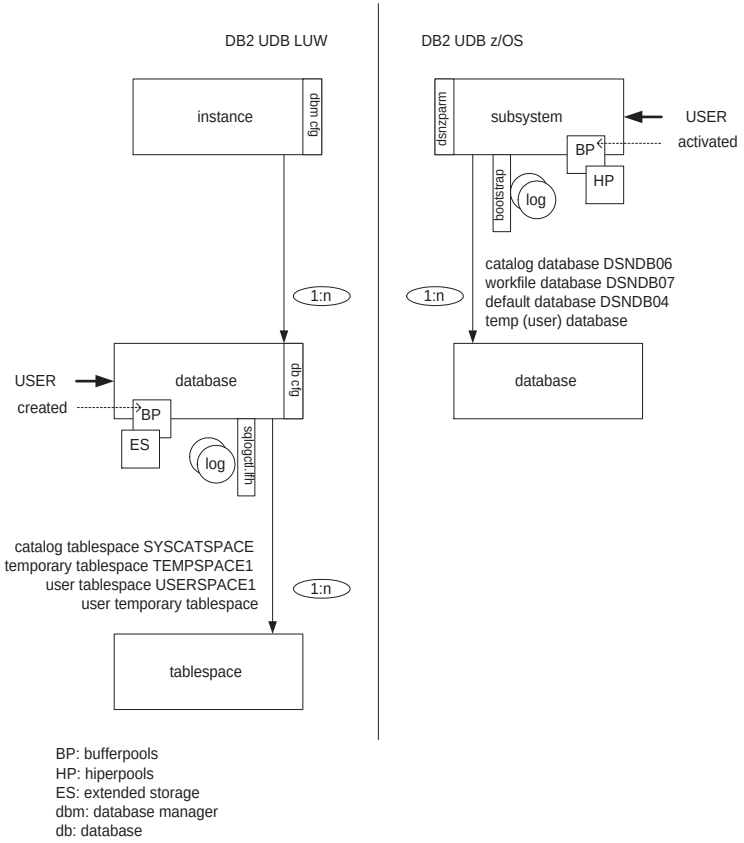
Gebruikers loggen aan. Op DB2 z/OS op het niveau van het subsysteem; op DB2 LUW aan een database. Logisch, aangezien op deze niveaus ook de DB2-cataloog beschikbaar is. Met als logisch gevolg dat applicaties op een DB2 z/OS systeem transparant toegang heb-

ben tot alle tabellen aangemaakt op het systeem (de database is transparant voor de applicatie). En dit is dus niet het geval voor DB2 LUW gebruikers.

DB2 LUW applicaties die data wensen te gebruiken uit verschillende databases binnen één instance, hebben behoefte aan een vooraf opgezette - zij het minimale - gedistribueerde architectuur.

Tenslotte nog dit - DB2 LUW users 'CONNECT-en' aan een database, en 'ATTACH-en' aan een instance, typisch voor beheer en configuratie van database-overschrijdende veiligheidsmaatregelen, configuratie van een gedistribueerde omgeving, etc.

Figuur 1: DB2 LUW versus DB2 z/OS structuur - overzicht



Memory allocaties

Zowel DB2 LUW als DB2 z/OS hechten veel belang aan bufferpools - beide database-systemen ondersteunen het gebruik van meerdere pools; en elke pool kan pagina's bevatten van een verschillende, standaardgrootte. Bufferpools zijn geassocieerd met tablespaces, waardoor de fysieke karakteristieken van de data in de data containers/datasets beïnvloed worden.

Merk op dat in een DB2 z/OS omgeving, bufferpools standaard bestaan, en via het '-ALTER BUFFERPOOL' commando (dus geen SQL-statement!) moeten worden geactiveerd. Op DB2 LUW is een formeel 'CREATE BUFFERPOOL' SQL-statement voorzien, dat bufferpools aanmaakt. Tijdens de creatie van een database wordt op dit platform automatisch een default, standaard bufferpool aangemaakt.

Opvallend is ook hier weer het verschillend 'hiërarchische' niveau waarop de bufferpool allocaties plaatsvinden - zie figuur 1.

In deze context is het voor DB2 LUW nuttig een belangrijk gevolg van de allocatie van deze memory-componenten op het niveau van de database te duiden. Op dit platform worden traditioneel databases gestopt en gestart op het moment dat de eerste gebruiker CONNECTeert, en de laatste gebruiker DISCONNECTeert. Vermits bufferpools gealloceerd worden op het niveau van de database, heeft dit een allocatie/deallocatie van bufferpools tot gevolg, met nadelige gevolgen voor het efficiënt gebruiken van in de bufferpool aanwezige data.

Directories

In beide omgevingen is vaak sprake over directory-structuren.

In DB2 z/OS verwijst men naar de DSNDB01-omgeving - een soort database met structuren die echter enkel door DB2 rechtstreeks zelf kunnen worden gemanipuleerd. De inhoud, conceptueel vergelijkbaar met bijvoorbeeld de inhoud van de cataloog, is niet aan de hand van SQL consulteerbaar. Het gaat hier concreet om een interne systeemstructuur, met gegevens opgeslagen in een intern, DB2-specifiek formaat.

In een DB2 LUW omgeving zijn directory-structuren een fundamenteel onderdeel van de gedistribueerde LUW-architectuur - beschouw het maar als configuratiefiles. De inhoud van deze structuren heeft niets te maken met de interne werking van DB2 als dusdanig. Opslag van deze gegevens in de cataloog is in de DB2 LUW context trouwens niet relevant, vermits deze zich op een te laag 'hiërarchisch' niveau bevindt.

Vervolg

In volgende nummers van Exploring DB2 gaan we verder in op een aantal verschillen tussen beide systemen, toegespitst op bepaalde zeer specifieke onderwerpen. En plaatsen we het verhaal in een nog groter kader, door ook aandacht te besteden aan andere, niet DB2, relationele database-systemen.

OVERZICHT VAN DE GEPUBLICEERDE ARTIKELS

Artikel	Nr.
Een systeemvergelijking - DB2 for LUW vs. DB2 for z/OS and OS/390	2-9
Extended SQL: onverwacht (?) efficiënt!	2-9
Tunen van Java/DB2 applicaties	2-2
Scrollable cursors in DB2 for z/OS	2-8
Alles over web services	2-7
Over web services, WORF, en DB2	2-7
Grid, web services en DB2 data	2-7
DB2 Data Provider verenigt .NET-clients en DB2	2-2
DB2 Data Provider verenigt .NET-clients - 2	2-5
DB2 Data Provider verenigt .NET-clients - 3	2-6
Over MQT - Materialized Query Tables	2-4
Triggers & redundancy	2-3
Partitioned tablespaces - er is een alternatief	2-1
Een buitenbeentje - LOBs en space management	2-6
Online Reorg in DB2, *Davy Goethals*, *Sidmar*	2-5
UDB federated databases - performance	2-8
DB2 performance - case 1: betere EXPLAIN, slechter toegangspad	2-1
DB2 performance - case 2: beter toegangspad, foutief resultaat	2-2
DB2 performance - case 3: duurder via een INDEX!	2-3
DB2 performance - case 4: probeer SQL varianten	2-4
Dossier 8	
Alles wordt groter - grenzen worden verlegd!	2-1
Over schema-evolutie	2-2
Multiple row insert	2-3
Multiple row select	2-4
Over stage 1 en distributiestatistieken	2-5
Op zoek naar extra veiligheid?	2-6
XML	2-7
Indexen op variabele lengte kolommen	2-8
Meer stage 1 in DB2 V8 ...	2-9

CURSUSPLANNING SEP - DEC 2004

DB2 concepten	375 EUR	27/09 (L), 29/11(W)
DB2 for OS/390, een totaaloverzicht	1625 EUR	20-24/09 (L), 04-8/10 (W), 15-19/11 (W), 06-10/12 (L)
DB2 UDB, een totaaloverzicht	1625 EUR	04-08/10 (W)
RDBMS concepten	325 EUR	20/09 (L), 04/10 (W), 15/11 (W), 06/12 (L)
Basiskennis SQL	325 EUR	21/09 (L), 05/10 (W), 16/11 (W) 07/12 (L)
DB2 for OS/390 basiscursus	975 EUR	22-24/09 (L), 06-08/10 (W), 17-19/11 (W), 08-10/12 (L)
DB2 UDB basiscursus	975 EUR	06-08/10 (W)
SQL workshop	700 EUR	21-22/10 (L), 22-23/11 (W), 22-23/12 (L)
DB2 for OS/390 programmering voor gevorderden	700 EUR	28-29/9 (L), 25-26/11 (L)
DB2 procedural extensions	350 EUR	30/09 (L), 30/11 (W)
DB2 for OS/390: SQL performance	1200 EUR	11-13/10 (L), 01-03/12 (W)
DB2 UDB applicatieperformance	400 EUR	22/12 (W)
Database applicatieprogrammering met JDBC	400 EUR	18/10 (L), 26/11 (L), 14/12 (W)
Fysiek ontwerp v. relationele DB's	700 EUR	07-08/10 (L)
DB2 for OS/390 database administratie	1600 EUR	25-28/10 (W), 14-17/12 (L)
DB2 for OS/390 operations and recovery	1500 EUR	08-10/11 (W)
DB2 UDB DBA 1 - Kernvaardigheid	1600 EUR	26-29/10 (W)
DB2 UDB DBA 2 - Advanced tuning	1200 EUR	13-15/12 (W)
DB2 up-to-date: extended SQL	400 EUR	15/09 (W), 18/10 (L)
DB2 up-to-date: XML in DB2	400 EUR	28/09 (W), 18/10 (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