ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 1 - NUMMER 8, MEI 2003

OPEN CURSOR

Op het moment dat we de laatste hand leggen aan dit nummer vernemen we via de pers het overlijden van Dr. Codd, vader van het relationele database model. Alle bestaande relationele databases zijn afgeleid van zijn initiële ideeën m.b.t. relationele theorie en verzamelingenleer.
Het toeval wil dat juist dit nummer de nadruk legt op een aantal elementen die recent zijn toegevoegd aan DB2 om 'tekortkomingen' eigen aan het relationele model weg te werken. Tekortkomingen die te maken hebben met het procedureel benaderen van data in verzamelingen.
Slechts weinigen hebben een zo grote, praktische invloed gehad op IT. Als er ooit een Nobelprijs voor IT wordt in het leven geroepen, moet die zonder twijfel postuum aan Dr. Codd worden toegekend.

Het ABIS DB2-team.

IN DIT NUMMER:

CLOSE CURSOR

Volgende maand gaan we in op de relatie tussen WebSphere Application Server en DB2. We staan ook even stil bij de integratie tussen de XML- en text-extenders. En we publiceren een laatste 'Dossier 7'.

Tot dan!

De CASE-expressie

Pieter Bedert (ABIS)

Welke ontwikkelaar is nog nooit gestruikeld over het gebrek aan een 'als-dan'-achtige constructie binnen SQL? Of wie heeft nog nooit moeten knoeien met ellenlange SQL statements die zesendertig keer 'UNION' gebruiken? DB2 biedt ons een elegante oplossing voor deze problemen: de CASE-expressie. Expressie (Eng: Expression) is een veelbetekenend woord in de DB2-wereld en om de CASE-expressie beter te begrijpen kunnen we misschien eerst het begrip expressie kort toelichten.

Expressie

We spreken over een expressie wanneer door een bewerking per rij één waarde gegenereerd wordt. Bekende voorbeelden van expressies zijn: kolommen, functies, berekeningen tussen verschillende kolommen, concatenaties van verschillende kolommen,...

Een expressie kan in de SELECT-instructie worden gebruikt.

Voorbeeld 1

```
SELECT voornaam CONCAT familienaam, YEAR(geboortedatum)
FROM persoonsinfo
```

Ook in de where-conditie is een expressie mogelijk en sinds kort is een expressie zelfs toegelaten in de 'order by'.

Voorbeeld 2

```
SELECT product, aantal, aantal * prijs AS totale_prijs
FROM bestelling
WHERE aantal * prijs > 100000
ORDER BY aantal * prijs
```

Eenvoudige CASE-expressie

De CASE-expressie kan op twee manieren gebruikt worden. De 'eenvoudige' manier evalueert een gelijkheid; de 'conditie'-manier evalueert condities. In beide gevallen zal het resultaat één enkele waarde genereren per rij.

```
CASE keuze-expressie
WHEN exp-1 THEN res-expr-1
WHEN exp-2 THEN res-expr-2
...
ELSE else-res-expr
END
```

Wanneer de ‘keuze-expressie' gelijk is aan 'exp-n' dan is het resultaat van de CASE-instructie de waarde van 'res-expr-n'. Wanneer de 'keuze-expressie' echter aan geen enkele expressie gelijk is dan is het resultaat van de CASE-instructie de waarde van de 'else-res-expr'. Als de ELSE niet vermeld is, dan is de resultaatwaarde NULL.

Voorbeeld 3

```
SELECT voornaam, naam, CASE landcode
                       WHEN 'BE' THEN 'België'
                       WHEN 'NL' THEN 'Nederland'
                       WHEN 'LU' THEN 'Luxemburg'
                       END AS land,
       landcode
FROM persoonsinfo
==========================================
VOORNAAM    NAAM           LAND        LANDCODE

Mark        Kop            Nederland   NL
Joop        Jansma         Nederland   NL
Gianni      Monza          -           IT
Jan         Janssen        Belgie      BE
Ben         Aat            Belgie      BE
Jean        Michel         Luxemburg   LU
Erich       Pfaff          -           DE
```

Conditie CASE-expressie

Naast een vergelijking kunnen we ook condities laten evalueren door de CASE-expressie. De syntax ziet er dan net iets anders uit.

```
CASE
WHEN cond-1 THEN res-expr-1
WHEN cond-2 THEN res-expr-2
...
ELSE else-res-expr
END
```

Wanneer aan 'cond-n' voldaan is, dan is het resultaat van de CASE-instructie de waarde van 'res-expr-n'. Als meerdere condities waar zijn, dan wordt de eerste 'res-expr' gekozen waarvoor de conditie waar is. Indien echter aan geen enkele conditie voldaan is, krijgen we 'else-res-expr' als resultaat of NULL wanneer geen ELSE gecodeerd werd.

Voorbeeld 4

```
SELECT productnaam,
       CASE
       WHEN aankoopprijs < 10000 THEN aankoopprijs
      WHEN aankoopprijs BETWEEN 10000 AND 50000 THEN aankoopprijs * 0.95
       ELSE aankoopprijs * 0.9
       END AS nieuwe_aankoopprijs
FROM bestellingen
```

Nuttige toepassingen en voorbeelden

De CASE-instructie kan in zeer veel situaties nuttig worden aangewend. Zo kan deze bijvoorbeeld gebruikt worden om zich te wapenen tegen uitzonderingen - het vermijden van delingen door nul is er zo één van.

Zo worden in voorbeeld 5 alle datums geselecteerd in 2003 waarvoor een bestelling gemiddeld minder dan 1000 euro opbrengt. Het gemiddelde mag niet berekend worden voor de dagen waarop geen enkele bestelling gedaan is.

Voorbeeld 5

```
SELECT datum
FROM audit_tabel
WHERE CASE
      WHEN aantal_bestellingen = 0 THEN 0
      ELSE totaal_daginkomsten / aantal_bestellingen
      END < 1000
AND year(datum) = 2003
```

De CASE-expressie is ook bijzonder nuttig om UNION-expressies te vermijden - voorbeeld 3 vereist zonder CASE-expressie 4 SELECTs!

Ook bij het berekenen van percentages kan de CASE-expressie zijn nut bewijzen. Voorbeeld 6 berekent het aandeel van de maand januari in de totale inkomsten van 2003. Daarvoor delen we de som van alle daginkomsten in januari door de som van de daginkomsten van het gehele jaar.

Voorbeeld 6

```
SELECT SUM(CASE
           WHEN month(datum) = 1 THEN totaal_daginkomsten
           ELSE 0
           END)  /  SUM(totaal_daginkomsten) * 100 AS aandeel_januari
FROM audit_tabel
WHERE year(datum) = 2003
```

Performance van de CASE-expressie

Het gebruik van de CASE-expressie in de SELECT is bijna gratis en men moet zich dus op het gebied van efficiëntie geen zorgen maken. De CASE in een WHERE-conditie is echter een stage-2 predikaat, waarvoor geen indexgebruik in aanmerking komt. Daar moet dus voorzichtig mee omgesprongen worden, tenzij deze conditie goed omringd wordt door stage-1 condities met een hoge filterfactor.

Er zijn zelfs gevallen waarbij de CASE-expressie aanzienlijke performance verbeteringen kan opleveren. Voorbeeld 6 is daar een illustratie van. Bij de query die gebruik maakt van de CASE-expressie, moet DB2 slechts eenmaal de tabel lezen. In elke andere query die je kunt bedenken om dezelfde vraag op te lossen moet DB2 twee keer de tabel lezen.

Wanneer de CASE-expressie kan gebruikt worden om een bestaande functie in DB2 te vervangen levert dat een performancevoordeel op. De overhead om de functie binnen DB2 te gaan oproepen gaat namelijk verloren. Een voorbeeld daarvan is het vervangen van de SIGN-functie door een CASE-instructie (voorbeeld 7).

Voorbeeld 7

```
SELECT CASE
       WHEN getal > 0 THEN   1
       WHEN getal < 0 THEN  -1
       ELSE 0
       END
FROM getallen
```

Besluit

Men beweert wel eens dat de CASE-expressie, die toch al ingevoerd is sinds versie 5 van DB2, het meest ondergewaardeerde statement is in SQL. Het is nu wel duidelijk dat het in bepaalde gevallen een sneller resultaat zal leveren en/of, bijvoorbeeld bij vervanging van een UNION-query, een meer leesbare SQL kan opleveren. Deze argumenten moeten voor elke DB2-ontwikkelaar of DBA voldoende motivatie bieden, om met meer dan gewone interesse naar de CASE-expressie te kijken!

Test

Bent u ook overtuigd?

Probeer eens even of u met de CASE-expressie overweg kan!

Bedenk een query (met dezelfde tabel en kolommen als voorbeeld 6) die als resultaat de maanden geeft die een aandeel van meer dan 20% in de totale omzet van het jaar 2003 hebben. Probeer uiteraard gebruik te maken van de CASE-expressie.

Oplossingen kunt u mailen naar training@abis.be. Eeuwige roem zal u deel zijn wanneer uw naam samen met de juiste oplossing wordt gepubliceerd in het volgende nummer van Exploring DB2!

SQL/PL - een beknopt overzicht

Eric Everaert (ABIS)

Het heeft een hele tijd geduurd vooraleer IBM een procedurele databasetaal heeft ter beschikking gesteld van DB2-ontwikkelaars: SQL/PL. Concurrenten Oracle, Sybase, en SQL Server beschikken reeds verschillende jaren over een transactie-georiënteerde taal, die de mogelijkheden van SQL aanzienlijk uitbreidt.

Dit artikel vraagt uw aandacht voor een aantal belangrijke karakteristieken van SQL/PL. Naast eerder conceptuele elementen, besteden we ook aandacht aan basissyntax, uiteengezet aan de hand van een voorbeeld.

SQL/PL

SQL/PL is een procedurele taal - een uitbreiding op SQL. De taal is gebaseerd op de SQL/PSM (Persistent Storage Module), een onderdeel van de SQL3-standaard. De taal is beschikbaar op elk DB2-platform, maar eigen aan IBM zijn een aantal implementatiedetails platform-afhankelijk.

Gebruik

SQL/PL bevat een aantal 'SQL controlestatements' - denk bijvoorbeeld aan een 'IF THEN ELSE'-constructie, een FOR-lus. Deze instructies kan men als volgt gebruiken.

- SQL/PL-instructies kan men toevoegen aan SQL (stored) procedures. Tijdens het ontwikkelproces worden deze omgezet in C-code. Het traditionele applicatieontwikkelproces moet worden geëerbiedigd: compilatie, link, bind zijn dus vereist. Een C/C++ compiler is inderdaad noodzakelijk. We hebben te maken met statische SQL.

- SQL/PL-instructies zijn echter ook bruikbaar in triggers of SQL-functies. Ze kunnen inderdaad worden gebruikt in dynamic compound statements - zeg maar geplaatst tussen 'begin atomic' en 'end' statements, bijvoorbeeld rechtstreeks uitgevoerd aan de prompt. In deze context is van een applicatieontwikkelproces geen sprake meer: eens (dynamisch) geoptimizeerd, worden ze onmiddellijk uitgevoerd. We hebben te maken met dynamische SQL.

De DB2-optimizer houdt rekening met het verschil tussen beide omgevingen. Zonder in detail te willen treden, wordt bijvoorbeeld een SQL-based FOR-lus met toegevoegd INSERT statement in een stored procedure omgebouwd volgens de regels van de traditionele cursorafhandeling. Diezelfde 'OR-lus wordt in de compound omgeving gewoon vertaald naar een INSERT AS SELECT, uiteraard veel sneller en efficiënter.

Niet alle instructies zijn bruikbaar in beide boven geschetste omgevingen. Consulteer de relevante IBM manuals voor meer informatie: *http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main*.
SQL/PL-informatie vindt u terug in de SQL reference manuals.

Voorbeeld? zie *Een voorbeeld van SQL/PL* op p. 13

In wat volgt besteden we aandacht aan een aantal eerder syntax-gerelateerde karakteristieken van SQL/PL. Ze worden geïllustreerd aan de hand van codefragmenten uit één groot voorbeeld. De letters tussen haakjes verwijzen naar de individuele instructies in de codefragmenten.

Basisstructuur

Globaal gesproken bestaat een SQL/PL-procedure uit een aantal blokken, typisch minstens één. Alle uit te voeren instructies zijn vervat binnen een 'BEGIN...END'-constructie. Blokken kunnen worden genest, dan wel naast mekaar worden gebruikt. Elk blok bevat typisch een aantal instructies, optioneel ook declaraties van variabelen, foutafhandelroutines, etc.

Bij het nesten van blokken gelden strikte regels wat betreft overerving: definities opgenomen in geneste blokken, hebben in geval van (naam)-conflict, steeds voorrang. Ook het onder controle houden van de 'procedure flow' is onder deze omstandigheden geen sinecure.

Declaratie van variabelen

Zoals elke procedurele taal vereist ook SQL/PL het gebruik en dus ook de declaratie van variabelen. Zoals steeds hebben ze als taak tijdelijke waarden te bevatten. Deze variabelen zijn bruikbaar in de procedure statements, maar ook in SQL-statements in SQL/PL vervat.

Merk op dat:

- de datatypes die SQL/PL ondersteunt voor de definitie van variabelen, identiek zijn aan deze die DB2 onderkent (**a**);
- variabelen met 'default' waarden kunnen worden aangemaakt (**b**);
- variabelen NULL kunnen bevatten - een concept dat andere procedurele talen niet ondersteunen.

Voorbeeld 1: declaraties

```
    CREATE PROCEDURE tbaccad.newenrol (thePersonNr SMALLINT,
                                       theSessionNr SMALLINT)
    BEGIN
(a)     DECLARE price SMALLINT;
(b)     DECLARE prijs DECIMAL(9)DEFAULT 000000000.;
        DECLARE companyNr SMALLINT;
        DECLARE noOfEnrolments SMALLINT;
        DECLARE dubbelfout INTEGER;
        DECLARE vpno SMALLINT;
    ...
```

Procedurele logica

Zoals boven reeds uiteengezet bestaat SQL/PL typisch uit een aantal blokken - de code wordt toegevoegd tussen een BEGIN en END statement (**c**). De traditionele syntaxconstructies vinden we hier terug: selecties (CASE, IF THEN ELSE), herhalingen (LOOP, FOR LOOP, REPEAT UNTIL, WHILE), en sequentiecontrole (GOTO, ITERATE, LEAVE). Voor een volledige beschrijving van deze instructies verwijzen we naar de betreffende manual.

Voorbeeld 2 bevat een CASE-instructie (**d**).

Voorbeeld 2: procedurele logica

```
(c) BEGIN
    ...
(d)    CASE
           WHEN
             noOfEnrols IN (1,2) THEN SET redPrice = price * 0.98;
           WHEN
(e)         noOfEnrols BETWEEN 3 AND 6  THEN SET redPrice = price * 0.95;
           ELSE
             SET reducedPrice = price * 0.92;
       END CASE;
    ...
(c) END;
```

Foutafhandeling

Foutafhandeling is zeer belangrijk in elke procedurele taal en zeker wanneer die taal kan gebruikt worden voor het aanmaken voor stored procedures. Belangrijk voor deze procedures is met name de vraag of fouten moeten aanleiding geven tot het beëindigen van het programma, het uitvoeren van automatische correcties, dan wel het registreren van de fout gevolgd door het aanvatten van de volgende verwerking; stored procedures zijn tenslotte 'batch'-georiënteerd.

Foutafhandeling in SQL/PL stoelt op het concept van fout-'handlers': stukjes logica die automatisch worden uitgevoerd als een bepaalde 'handler'-specifieke fout optreedt. Elke 'handler' heeft een naam en kan worden geassocieerd met (zie voorbeeld 3):

- een SQLSTATE (**f**);
- een logische fout gedefinieerd door de gebruiker - een speciale instructie 'signal' is voorzien om 'handlers' te activeren;
- gereserveerde, generieke condities SQLWARNING, SQLEXCEPTION en NOT FOUND.

Elke 'handler'-definitie bevat daarnaast ook een aanduiding hoe, volgend op de fout, de procedure moet worden verdergezet, *na* het uitvoeren van de 'handler'-code (**g**). Mogelijke opties zijn:

- CONTINUE, waarbij de procedure hervat met de instructie volgend op diegene die de 'handler' heeft opgeroepen;

- EXIT, waarbij de procedure hervat met de instructie volgend op het blok dat de uitgevoerde 'handler' heeft gedeclareerd;
- UNDO, waarbij alle instructies die deel uitmaken van het blok, aan een rollback worden onderworpen.

Voorbeeld 3: foutafhandeling

```
(f)  DECLARE dubbel CONDITION FOR SQLSTATE '23505';
(g)  DECLARE CONTINUE HANDLER FOR dubbel
       BEGIN
          INSERT INTO result
          VALUES ('problem: combinatie '||CHAR(thePersonNr)||'-'||
              CHAR(theSessionNr)||' bestaat!',NULL,CURRENT TIMESTAMP);
       SET dubbelfout=1;
       END;
```

Foutafhandeling wordt extra uitdagend waneer bepaalde fout-'handlers' niet zijn voorzien in het blok waarin de fout is opgetreden, maar in een parentblok - de programflow verplaatst zich immers naar dit blok. Ook het doormekaar gebruiken van de generieke 'handler'-condities met specifieke SQLSTATE-'handlers' kan tot aanzienlijke verwarring leiden.

Gebruik van SQL

Zoals te verwachten, zijn alle relevante SQL-instructies op een eenvoudige wijze te integreren in een SQL/PL-procedure (zie voorbeeld 4). Het gaat hier dan zowel om SQL-statements in de strikte zin van het woord (**h**), als om SQL-achtige constructies toegevoegd aan traditionele procedurele logica.

Voorbeeld 4: SQL in SQL/PL

```
   BEGIN
   ...
(h)   SELECT max(eno) + 1 INTO nextNr FROM tbaccad.enrolments
      WHERE e_sno = theSessionNr;
(i)   IF nextNr IS NULL THEN SET nextNr = 0; END IF;
   ...
(j)   SET vpnaam = (SELECT plname FROM tbaccad.tutpersons
                    WHERE pno = thePersonNr;
   ...
(k)   FOR i AS cur1 CURSOR FOR
        SELECT epay,s_cid
        FROM tbaccad.enrolments,tbaccad.tutsessions
        WHERE sno=e_sno AND ecancel IS NULL AND scancel IS NULL
      DO
        SELECT caprice INTO prijs
        FROM tbaccad.tutcourses
        WHERE cid=i.s_cid;
        IF i.epay < prijs
        THEN SET totalreduct=totalreduct+(prijs-i.epay);
        END IF;
      END FOR;
   ..
   END;
```

Met SQL-achtige constructies verwijzen we bijvoorbeeld naar het gebruik van traditionele WHERE-constructies, rechtstreeks in de procedurele logica (**i**). Deze NULL-test is inderdaad veel intuïtiever dan de traditionele 'indicator'-tests eigen aan andere procedurele talen, als C en COBOL. Zie ook voorbeeld 2, (**e**).

Deze techniek maakt het werken met SQL/PL zeer eenvoudig en intuïtief.

Een aantal opmerkingen voegen we graag toe.

- Het resultaat van een SQL SELECT-statement dat slechts één rij bevat, kan zonder het gebruik van een cursor aan SQL/PL-variabelen worden toegekend. Twee technieken zijn beschikbaar: de traditionele 'SELECT .. INTO'-techniek, gekend van embedded SQL (**h**), dan wel de SET-techniek. Hierbij wordt het resultaat van het SELECT-statement toegekend aan een variabele (**j**). Merk op dat in dit laatste geval, het niet vinden van een rij, geen fout zal opleveren - de waarde NULL wordt toegevoegd aan de variabele. In het eerste geval geeft dit echter aanleiding tot een 'NOT FOUND'-exceptie.

- SQL/PL is in zijn huidige vorm nog steeds een eenvoudige procedurele taal. Multi-dimensionele variabelen (rijen) bestaan (nog) niet. Dit heeft als gevolg dat SELECT-statements die data ophalen, nog steeds van cursoren moeten gebruik maken indien ze meer dan één rij ophalen uit de database. De traditionele cursorwerking is uiteraard ondersteund - declaratie van cursoren, alsook OPEN, FETCH' en CLOSE zijn toegelaten instructies. Zeer mooi is echter de techniek waarbij de cursorafhandeling quasi automatisch door SQL/PL wordt afgehandeld in een FOR-loop (**k**). De cursor moet vooraf niet worden gedeclareerd: het openen, sluiten en lezen gebeurt 'vol-automatisch' tijdens het traditionele 'loop-en' doorheen de FOR-lus. Merk trouwens ook op dat de FOR-lus index automatisch wordt aangemaakt als een record, met evenveel velden als er kolommen worden geselecteerd in de SELECT-instructie. De traditionele 'record.veld' notatie kan worden gebruikt om de aldus gelezen waarden in de rest van de procedure aan te wenden.

Besluit

SQL/PL is zeker geen alternatief voor applicaties ontwikkeld in COBOL of C. Maar het biedt de applicatieontwikkelaar wel de mogelijkheid om bepaalde herbruikbare stukjes code, op een platform onafhankelijke en transparante manier in de database op te nemen. Naarmate de mogelijkheden van SQL/PL doorheen de tijd verder uitbreiden, kan verwacht worden dat ook het gebruik ervan zal toenemen.

DOSSIER 7

Leafnear en leaffar

Leaffar en leafnear zijn twee nieuwe kolommen in SYSIBM.SYSINDEXPART. Deze twee nieuwe kolommen kunnen mee bijdragen tot de beslissing om over te gaan tot een herorganisatie van de index.

Leaffar en leafnear proberen een idee te geven van de fysische afwijking van de logische structuur van de index. Leaf-pages die in de logische structuur mooi naast mekaar zitten, worden in realiteit niet altijd naast mekaar bewaard. Bij een page-split bijvoorbeeld is er een grote kans dat de nieuwe pagina zich fysiek niet naast de originele pagina bevindt. DB2 moet dan over een aantal pages "springen" om de logisch volgende page te bereiken. Het aantal keer dat DB2 moet springen vindt men terug in deze kolommen en zou dus klein moeten zijn. In dat geval is de fysische volgorde waarin dat de pages bewaard zijn bijna dezelfde als de logische volgorde. Met het verschil tussen leafnear en leaffar wil men aangeven of DB2 binnen of buiten de prefetch quantity springt. Leaffar is daarom belangrijker als leafnear. In de leafnear kolom staat dus een getal dat weergeeft hoe dikwijls dat de logisch volgende page niet terug te vinden is als fysisch volgend page, maar wel ergens binnen dezelfde prefetch quantity. Het getal in de leaffar kolom geeft het aantal keer dat het niet terug te vinden was als volgende page en ook niet binnen de prefetch quantity.

Een idee over het al dan niet goed bewaard zijn van de leaf-pages kon men vroeger al verkrijgen met behulp van de kolom leafdist, ook uit SYSIBM.SYSINDEXPART. Het verschil is dat deze twee nieuwe kolommen intuïtief veel duidelijker zijn. Leafdist zegt ook niets over het aantal sprongen maar wel over het gemiddeld aantal pages (x 100) dat tussen twee logische aansluitende pages voorkomt. Zo zou het kunnen dat men voor een zelfde leafdist waarde een beperkt aantal sprongen heeft in één index, maar een redlijk groot aantal in een andere index.

Bekijken we een fictief voorbeeld. Een index heeft 10 leaf-pages. In het eerste geval springen we om de logische volgorde te behouden naar page 6, we slagen 4 pages over, voor het vervolg keren we terug naar de tweede leaf page, we slagen 3 leaf pages over. De volgende pages staan in volgorde, alleen moeten we nog over page 6 springen. Dit betekent dat we drie keer gesprongen hebben (dus een leafnear waarde 3 en leaffar 0 hebben indien we ons steeds binnen de prefetch quatity bevinden) met een leafdist waarde van ((4 + 3 + 1) / 10) x 100= 80. Met zes sprongen waarbij we van page 1 naar 4 gaan, dan terug keren naar 2, dan 3 (niets aan de hand), dan 4 overslaan en nog een paar keren springen bekomt men juist dezelfde leafdist: ((2+1+1+2+1+1) / 10) x 100 = 80 maar het aantal sprongen is twee maal zo groot (leafnear wordt 6 en leaffar 0 indien we ons steeds binnen de prefetch quatity bevinden).

Katrien Platteborze (ABIS)

CURSUSPLANNING MEI - JUN 2003

DB2 for OS/390, een totaaloverzicht	1625 EUR	22-28/05/2003 (W), 02-06/06 (L)
DB2 UDB, een totaaloverzicht	1625 EUR	22-23/5 & 02-04/06 (W)
RDBMS concepten	325 EUR	22/05 (W), 02/06 (L)
Basiskennis SQL	325 EUR	23/05 (W), 03/06 (L)
DB2 for OS/390 basiscursus	975 EUR	26-28/05 (W), 04-06/06 (L)
DB2 UDB basiscursus	975 EUR	02-04/06 (W)
DB2 UDB concepten	375 EUR	05/06 (L)
SQL workshop	700 EUR	12-13/06 (W), 16-17/06 (L)
DB2 for OS/390 programmering voor gevorderden	1050 EUR	19-21/05 (L)
DB2 for OS/390: SQL performance	1200 EUR	11-13/06 (L)
DB2 for OS/390 system programming	1650 EUR	03-05/06 (L)
DB2 for OS/390 V7 upgrade voor ontwikkelaars	375 EUR	06/06 (L)
DB2 week - een overzicht van uitdagende nieuwe features in DB2!		
° The procedural DBA	400 EUR	16/06 (W)
° Extended SQL in DB2	400 EUR	17/06 (w)
° XML in DB2	400 EUR	18/06 (W)
° Federated databases	400 EUR	23/06 (W)
° JAVA and .NET in a DB2 environment	400 EUR	24/06 (w)

Plaats: L = Leuven; W = Woerden
Details, andere data en bijkomende cursussen: www.abis.be.
De planning voor het najaar is online beschikbaar vanaf 23 mei 2003.

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

Een voorbeeld van SQL/PL

```
CREATE PROCEDURE tbaccad.newenrol (thePersonNr SMALLINT, theSessionNr
SMALLINT)
BEGIN
    DECLARE price SMALLINT;
    DECLARE prijs DECIMAL(9)DEFAULT 000000000.;
    DECLARE companyNr SMALLINT;
    DECLARE noOfEnrolments SMALLINT;
    DECLARE reducedPrice DECIMAL(9);
    DECLARE nextNr SMALLINT;
    DECLARE totalreduct DECIMAL(9);
    DECLARE SnoOrPnofout INTEGER;
    DECLARE dubbelfout INTEGER;
    DECLARE vpno SMALLINT;


    DECLARE dubbel CONDITION FOR SQLSTATE '23505';
    DECLARE CONTINUE HANDLER FOR dubbel
        BEGIN
            INSERT INTO tbaccad.result
              VALUES ('problem: combinatie '||CHAR(thePersonNr)||'-'||
                       CHAR(theSessionNr)||' bestaat al',NULL,CURRENT
TIMESTAMP);
            SET dubbelfout=1;
        END;
   SET SnoOrPnofout=0;
   SET dubbelfout=0;

  BEGIN
        DECLARE EXIT HANDLER FOR NOT FOUND
            BEGIN
                INSERT INTO tbaccad.result
                    VALUES ('problem:Cursus nr of sessie num onbekend',
                                NULL,CURRENT TIMESTAMP);
                SET SnoOrPnofout=1;
            END;
        -- determining course price
        SELECT caprice INTO price FROM tbaccad.tutcourses
                WHERE cid = (SELECT s_cid FROM tbaccad.tutsessions
                                WHERE sno = theSessionNr);

  END;
   BEGIN
        DECLARE EXIT HANDLER FOR NOT FOUND
            BEGIN
                INSERT INTO tbaccad.result
                        VALUES ('problem:Person ombekend',
                                NULL,current timestamp);
                SET SnoOrPnofout=1;
            END;
        -- determining course price
        SELECT pno INTO vpno FROM tbaccad.tutpersons
                WHERE pno=thePersonNr;
      --  SET vpno=(SELECT pno FROM tbaccad.tutpersons
      --                   WHERE pno=thePersonNr);
  END;
```

```
-- determining number of enrolments for company
    SELECT e_cono, count(*) INTO companyNr, noOfEnrolments
            FROM tbaccad.tutenrolments
                WHERE e_cono = (SELECT pa_cono FROM tbaccad.tutpersons
                                        WHERE pno = thePersonNr)
                and ecancel is null
        GROUP BY e_cono;

-- calculating reduction
    CASE
      WHEN
        noOfEnrolments in (1,2) THEN SET reducedPrice = price * 0.98;
      WHEN
        noOfEnrolments BETWEEN 3 AND 6  THEN SET reducedPrice = price
* 0.95;
      ELSE
        SET reducedPrice = price * 0.92;
    END CASE;

-- determining the enrolmentís number
    SELECT max(eno) + 1 INTO nextNr FROM tbaccad.enrolments
        WHERE e_sno = theSessionNr;
    IF nextNr is null THEN SET  nextNr = 0; END IF;

-- inserting data
    IF SnoOrPnofout <> 1 THEN

        INSERT INTO tbaccad.enrolments
                (e_sno, eno, e_pno, epay, e_cono, ecancel,einv_cono)
                VALUES(theSessionNr, nextNr, thePersonNr,
                        reducedPrice, companyNr, null, companyNr);
        IF dubbelfout = 0 THEN
                INSERT INTO tbaccad.result(col_1, col_2,col_3)
                      SELECT plname||pfname, price -reducedPrice, 'ok'
                                FROM tbaccad.tutpersons
                                WHERE pno = thePersonNr;
        END IF;

    SET totalreduct=0;
    FOR l1 AS cur1 CURSOR FOR
        SELECT epay,s_cid
                FROM tbaccad.enrolments,tbaccad.tutsessions
               WHERE sno=e_sno AND ecancel IS NULL AND scancel IS NULL
      DO
        SELECT caprice INTO prijs FROM tbaccad.tutcourses WHERE
cid=l1.s_cid;
        IF l1.epay < prijs --(select caprice from tbaccad.tutcourses
where cid=l1.s_cid)
                THEN SET totalreduct=totalreduct+(prijs-l1.epay);
        END IF;
      END FOR;
    INSERT INTO tbaccad.result VALUES ('Reduction
total',totalreduct,current timestamp);
    END IF;
END
```