ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 1 - NUMMER 2, OKTOBER 2002


## OPEN CURSOR

*Deze editie van 'Exploring DB2' wordt afgerond op het moment dat de herinneringen aan 11 september 2001 nog vers in onze geest staan.*
*Quasi verworvenheden worden terug in vraag gesteld; nieuwe accenten worden gelegd. We voelen de weerslag op onze dagelijkse leef- en werkomgeving.*
*De economische recessie die hierdoor nog geaccentueerd werd, heeft ook zijn impact gehad op de wereld van de ICT. Een gevoel van euforie heeft plaats gemaakt voor - overdreven? - pessimisme. De klemtoon in de slogan 'Back to Business' ligt dan ook niet op 'back', - nog meer van het zelfde - maar op 'Business' - wat wil de klant, wat is voor hem relevant?*
*Wij hopen met dit nummer een bescheiden bijdrage te leveren aan de technische ondersteuning van die Business.*

*Het ABIS DB2 team.*

## IN DIT NUMMER:

- Triggers - relatief nieuw in DB2 voor OS/390. Eerst eigen aan ERP implementaties als SAP, nu ook meer en meer gebruikt in traditionele DB2 omgevingen. In *Business logica in triggers - Pendant actions* wordt hiervan een voorbeeld gegeven.
- *Dossier 7*: zie de catalog groeien - over primary keys en unique constraints.
- *ROWIDs en RIDs - slechts 2 letters verschil?* Of is het toch meer dan dat? Kunnen ze beide rechtstreeks gebruikt worden in queries?
- *Cursusplanning okt-dec 2002.*

## CLOSE CURSOR

In het november nummer besteden we uitgebreid aandacht aan een nieuwe techniek om SQL statements te optimizeren: de DSN_STATEMNT_TABLE.

Tot dan!

# Business logica in triggers - Pendant actions

*Pieter Bedert (ABIS)*

## Business logica

Wanneer we op een “DB2-manier" over bedrijfslogica spreken, dan hebben we het over de gevolgen van je eigen bedrijfsnoden op database niveau. Bekende voorbeelden zijn: in het bankwezen kunnen bepaalde rekeningen nooit negatief zijn; een bepaalde klantengroep is nooit ouder dan x aantal jaar;... Deze logica kon m.b.v. constraints in onze database opgenomen worden.

Triggers geven ons nu een kans om meer dynamische bedrijfslogica te implementeren. Zoals bijvoorbeeld: automatisch prijstotalen laten berekenen, automatisch controlerijen toevoegen bij bepaalde acties, enz.

## Efficiënte applicatie ontwikkeling

Eén van de hedendaagse kreten bij applicatie ontwikkeling, is het belang van hergebruik van code. Hoewel dit vooral de bedoeling was van object georiënteerde talen, moet dit ook een zorg zijn voor de meer traditionele DB2-ontwikkelaar. Er gaat op dit ogenblik veel tijd verloren aan het implementeren en herhalen van bedrijfslogica in hun applicaties. Denk maar eens aan het aanpassen van een prijstotaal-kolom: in elk programma wordt dan een aantal lijnen code gewijd aan het herberekenen van een prijstotaal, terwijl DB2 ons, a.d.h.v. een trigger, de mogelijkheden geeft om dit bij elke manipulatie automatisch te herberekenen. Er wordt op deze manier een aanzienlijke tijdwinst geboekt bij het ontwikkelen.

## Pendant actions

We spreken hier specifiek over acties die moeten ondernomen worden tijdens het verwijderen van een laatste dependent rij horende bij één primary key.

Veel bedrijven hebben nood aan zo'n soort logica in de database. We geven enkele voorbeelden:

- Een bedrijf doet aan “Order management”: Er zijn twee tabellen, één tabel met de algemene bestelling (parent), een tweede tabel die voor elk product een orderlijn bevat (dependent). Als nu alle orderlijnen verwerkt zijn (verwijderd uit de dependent tabel) van één bestelling dan mag deze bestelling gefactureerd worden (verplaatst naar een andere tabel).

- Een bedrijf doet aan archivering: We spreken over 3 tabellen, één tabel zijn de huidige leveranciers (parent), de tweede tabel zijn ex-leveranciers, een derde tabel (dependent) is een productentabel die verwijst naar de leverancier. Wanneer een bedrijf nu geen enkel

product meer levert, dan mag dit bedrijf in de "archieftabel" geplaatst worden.

- Een bedrijf organiseert transport, seminaries,...: De ene tabel bevat het transport of het georganiseerde seminarie, de andere tabel bevat alle klanten die moeten betalen voor één transport of seminarie. Wanneer de laatste klant betaald heeft (verwijderd uit de betalingentabel) dan moet een “flag”, aanduiden dat het volledige transport/seminarie betaald is.

## Implementatie

Zoals we hierboven vermeld hebben, is het niet nodig om in alle programma's deze logica te herhalen. Wanneer we een trigger gaan definiëren op de dependent tabel dan zullen de "pendant actions" automatisch gaan gebeuren bij het verwijderen van de laatste dependent rij van één primary key. Vanaf dat moment hoeft geen enkele applicatie ontwikkelaar zich nog zorgen te maken over een delete uit de dependent table.

## Triggers

Triggers worden gedefinieerd in relatie met een insert, delete of update. Wanneer een trigger gecreëerd is, dan wordt deze altijd "afgevuurd" vóór of na een bijbehorende delete, insert of update. Bijvoorbeeld bij een update van een prijskolom, wordt een totaalprijs automatisch herberekend.

## Triggers creëren

Een trigger wordt gemaakt m.b.v. DDL. We hebben de keuze om de trigger vóór of na een actie te laten gebeuren (before / after). Aan de hand van een voorwaarde wordt beslist of de bijbehorende acties wel of niet uitgevoerd worden (when). Alle acties die moeten ondernomen worden, kunnen in SQL geschreven worden.

Een voorbeeld: *Bijlage* op p. 9.

## Trigger voor de "pendant actions"

```
CREATE TRIGGER
pendant_actions
AFTER DELETE ON orderlines
REFERENCING OLD AS old_row
FOR EACH ROW MODE DB2SQL
WHEN (0 = (select count(*)
          from orderlines
          where orderid =
          old_row.orderid))
BEGIN ATOMIC
        insert into
payments_table
select *
from order_table
where orderid =
old_row.orderid;
delete from order_table
where orderid =
old_row.orderid;
END
```

Deze trigger zal "afgevuurd" worden net na het verwijderen van de laatste dependent rij van één primary key waarde.

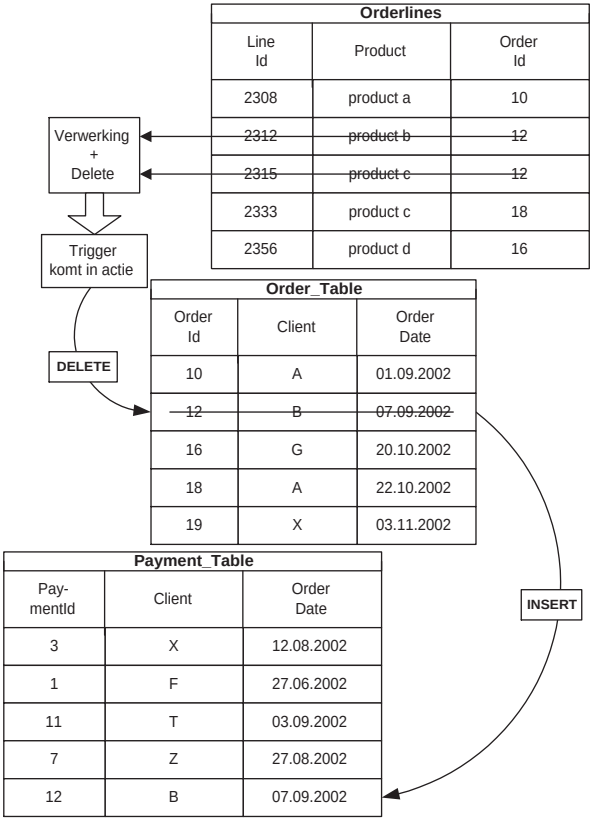
We werken het voorbeeld over "Order management" uit. We beschikken over twee tabellen, één tabel met de algemene bestelling (parent) deze tabel noemen we de "order_table", een tweede tabel met de aparte orderlijnen (dependent) deze noemen we de "orderlines". Wanneer we de laatste orderlijn horende bij één bestelling ver-

werkt/verwijderd hebben dan zullen we deze bestelling verplaatsen naar de "payments_table". De volgende trigger zal deze actie na de creatie perfect uitvoeren:

Wanneer de "WHEN"-conditie waar is, dan worden alle SQL-acties tussen "BEGIN ATOMIC" en "END" uitgevoerd. In dit specifiek geval wil dit zeggen dat na het verdwijnen van de laatste orderlijn verwijzend naar een zelfde order, de order verplaatst wordt naar de "payments_table".

Op dezelfde manier kunnen we te werk gaan voor het tweede voorbeeld. Het derde voorbeeld is zelfs eenvoudiger, want in de trigger moet enkel een update uitgevoerd worden.

*Figuur 1: Een voorbeeld van pendant delete*



# DOSSIER 7

## De catalog groeit!
## Deel 1 - (over primary keys en unique constraints)

Uiteraard groeit de catalog, of heeft u misschien geen activiteit in uw DB2 databases? Maar daarover gaat dit artikel niet, wel over nieuwe tabellen. Bij iedere nieuwe versie van DB2 komen er een aantal nieuwe catalog tabellen bij. Waar DB2 V6 nog 65 (62)catalog tabellen had, beschikt de DB2 V7 catalog over 87 (81) tabellen. In dit artikel bespreken we 2 nieuwe interessante tabellen: SYSIBM.SYSTABCONST en SYSIBM.SYSKEYCOLUSE en wijden we een beetje verder uit over constraints.

Ondertussen is iedereen het gewoon geworden: wanneer we informatie wensen over de primary key van een tabel moeten we de KEYSEQ kolom van SYSIBM.SYSCOLUMNS bekijken, maar voor menig beginneling leverde dit toch een zoektocht op naar de primary key. Foreign keys en check constraints hebben nooit een probleem gegeven, elk had zijn eigen tabel(len): respectievelijk de SYSIBM.SYSRELS en SYSIBM.SYSFOREIGNKEYS, en SYSIBM.SYSCHECKS. En hoe zit het met de unieke constraint? Een unieke constraint zonder unieke index werd beschreven in een extra rij in de SYSIBM.SYSCOLUMNS met COLNO 0. Eens de definitie volledig gemaakt werd door een unieke index aan te maken werd de constraint zelf nergens meer beschreven; wel vonden we een waarde 'C' in de kolom UNIQUERULE van de SYSIBM.SYSINDEXES tabel. Ook het gebruik van de kolommen STATUS en TABLESTATUS van SYSIBM.SYSTABLES is niet echt consequent. Deze worden alleen gebruikt om aan te geven of de primary key volledig gedefinieerd is en geven geen informatie over een unieke constraint.

Daar komt nu verandering in. Met de twee nieuwe tabellen SYSIBM.SYSTABCONST en SYSIBM.SYSKEYCOLUSE zijn de primary key en unieke constraints altijd duidelijk en gemakkelijk opzoekbaar. In de SYSTABCONST tabel wordt elke primary key en elke unieke constraint met 1 rij beschreven onafhankelijk van het bestaan van de bijhorende index. In SYSKEYCOLUSE vindt men een rij voor iedere kolom die deel uitmaakt van de primary key of unieke constraint.

Tegelijkertijd zijn er aanpassingen aan de kolommen STATUS en TABLESTATUS van SYSIBM.SYSTABLES. STATUS krijgt nog altijd de waarde I wanneer de definitie van de tabel niet volledig is, maar de oorzaak kan nu ook bij de unieke constraint liggen. TABLESTATUS kan nu naast de waarde 'P' ook een waarde 'U' of zelfs 'PU' bevatten. Dit geeft weer dat ofwel een index op de primary key, ofwel op de unieke constraint, ofwel op beide noodzakelijk zijn om de tabel definitie volledig te maken.

Nog een toemaatje: vanaf versie 7 worden constraints aangemaakt met een uniforme syntax die toelaat dat elke constraint een naam kan krijgen. Ook unieke constraints kunnen nu toegevoegd en verwijderd worden met het alter tabel statement. Ondersteunende indexen voor primary keys en unieke constraints kunnen niet meer gedropt worden zonder eerst de constraint te verwijderen. De weg naar uniformisering is echter lang: er is nog één probleem! De naam van een foreign key kan 8 karakters lang zijn die van de andere constraints 18...

*Bron: www.ibm.com/db2*

*Katrien Platteborze (ABIS)*

# ROWIDs en RIDs - slechts 2 letters verschil?

*Katrien Platteborze (ABIS)*

De invoering van de ROWID in Versie 6 van DB2 UDB voor OS/390 veroorzaakte bij velen verwarring. Kunnen we nu rechtstreeks met RIDs gaan werken was een veel gehoorde vraag. Om direct op deze vragen te antwoorden: Er is wel zeker een groter verschil dan alleen twee letters, en we kunnen niet rechtstreeks met RIDs werken.

## RIDs

RIDs zijn unieke identifiers die DB2 *intern* gebruikt om records te identificeren. RIDs vindt men terug op leaf pages in de index. Ze fungeren als pointer in de index en worden gebruikt om snel het record horend bij een key op te halen. In verschillende accessmethoden zoals list prefetch en multiple index access maakt DB2 gebruik van de waarde van de RID zelf.

Als een tabel gereorganiseerd wordt kunnen (en zullen) deze waarden veranderen.

Het RID concept is volledig transparant voor de applicatieontwikkelaar - rechtstreeks ernaar verwijzen bij select, update of delete operaties is niet toegestaan.

## ROWIDs

ROWIDs zijn waarden die een rij uniek maken en te samen met de rij bewaard worden. De ROWID waarden hebben het datatype ROWID. De waarden van een kolom met het ROWID datatype worden door DB2 zelf gegenereerd. Daartoe moet de ROWID gedefinieerd worden met ‘generated always’ of ‘generated by default’. De waarden zijn uniek, ze kunnen daarom eventueel gebruikt worden als primary key. Aangezien de waarden gebruikt worden om een rij uniek te identificeren moet DB2 de waarden behouden zelfs na het uitvoeren van een REORG.

ROWIDs zijn voornamelijk gekend omdat ze noodzakelijk zijn wanneer men met LOBs werkt. Nochtans kan men ze ook volledig los van LOBs gebruiken wanneer men een rij rechtstreeks wil ophalen zonder gebruik te maken van index access of table scan. Dit impliceert natuurlijk dat men de ROWID waarde kent. Een applicatie kan een rij ophalen waar onder andere een ROWID kolom inzit. De ROWID waarde kan daarna in andere statements (zoals update of delete) gebruikt worden om rechtstreeks de rij te identificeren. Deze methode wordt 'direct row access' genoemd en wordt in de plan_table aangeduid met ‘D’ in de kolom primary_accesstype.

RIDs en ROWIDs kunnen nooit te samen gebruikt worden in een toegangspad, het is ofwel het ene ofwel het andere.

Volgende regels gelden:

- Een ROWID kolom moet gedefinieerd worden met NOT NULL;
- Een tabel kan maar 1 ROWID kolom bevatten;
- Een ROWID kolom kan niet ge-update worden, ook niet door een trigger;
- Er kunnen geen check constraints gedefinieerd worden op een ROWID kolom;
- Een ROWID kolom moet gedefinieerd worden met 'generated always' of 'generated by default';
- Wanneer de ROWID kolom gedefinieerd werd met generated by default moet er een unieke index op gedefinieerd worden.

Onderstaande figuur vat een aantal rowid-gerelateerde operaties samen.

*Figuur1: Create syntax van een tabel met ROWID kolom, SQL statements in een COBOL applicatie en explain output. (De eerste rij is de output van het eerste select statement)*

```
create table metrowid
 (cola rowid not null generated always,
  colb smallint) in database abis;

insert into metrowid(colb) values (1);
```

```
         EXEC SQL DECLARE TB00020.METROWID TABLE
         ( COLA                     ROWID NOT NULL,
           COLB                           SMALLINT
         ) END-EXEC.
     01  DCLMETROWID.
         10 COLA                SQL TYPE IS ROWID.
         10 COLB                PIC S9(4) USAGE COMP.
         EXEC SQL
            INCLUDE SQLCA
         END-EXEC.
     PROCEDURE DIVISION.
     PROG-START.
         EXEC SQL
           SELECT COLA, COLB INTO :COLA, :COLB
           FROM METROWID
           WHERE COLB = 1
         END-EXEC.
         DISPLAY COLA
         DISPLAY COLB
         EXEC SQL
           SELECT COLB INTO :COLB
           FROM METROWID
           WHERE COLA = :COLA
         END-EXEC.
         STOP RUN.
```

```
TNAME      ACCESSTYPE  MATCHCOLS  ACCESSNAME         PRIMARY_ACCESSTYPE
METROWID            R          0
METROWID            R          0                       D
```

# CURSUSPLANNING OKT-DEC 2002

| | | |
|---|---|---|
| DB2 for OS 390, een totaal-overzicht | 1625 EUR | 04-08/11 (L), 18-22/11 (W), 09-13/12 (L) |
| DB2 UDB, een totaaloverzicht | 1625 EUR | 18-19 & 27-29/11 (W) |
| RDBMS concepten | 325 EUR | 04/11 (L), 18/11 (W), 09/12 (L) |
| Basiskennis SQL | 325 EUR | 05/11 (L), 19/11 (W), 10/12 (L) |
| DB2 for OS/390 basiscursus | 975 EUR | 06-08/11 (L), 20-22/11 (W), 11-13/12 (L) |
| DB2 UDB basiscursus | 975 EUR | 27-29/11 (W) |
| DB2 UDB concepten | 375 EUR | 12/11 (W), 03/12 (L) |
| SQL workshop | 700 EUR | 24-25/10 (W), 16-17/12 (L) |
| DB2 applicatieprogrammering voor gevorderden | 1050 EUR | 21-23/10 (W), 04-06/12 (L) |
| DB2 for OS/390: SQL performance | 1200 EUR | 13-15/11 (W), 16-18/12 (L) |
| Fysiek ontwerp van relationele databases | 700 EUR | 04-05/11 (L) |
| DB2 database administratie | 1600 EUR | 18-21/11 (L) |
| DB2 UDB database administratie | 1600 EUR | 28-31/10 (W), 12-15/11 (L) |
| DB2 UDB systeembeheer en performance | 400 EUR | 01/11 (W), 25/11 (L) |
| DB2 UDB for OS/390 V7 upgrade voor ontwikkelaars | 375 EUR | 25/10 (L), 05/12 (W) |
| DB2 UDB en zijn extenders: XML en text search | 200 EUR | 30/10 (W), 16/12 (L) |
| DB2 UDB integratie met MQSeries | 200 EUR | 28/10 (W), 16/12 (L) |
| Integrating IBM technologies - Web Services | 800 EUR | 07-08/11 (L), 12-13/12 (W) |

Na het succes van het up-to-date seminar 'Integrating IBM technologies' in het voorjaar 2002, presenteren we in het najaar een vervolg: toegespitst op Web Services, specifiek in een IBM omgeving met o.a. WebSphere en DB2 UDB.

*Plaats: L = Leuven; W = Woerden*
*Details, andere data en bijkomende 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

# Bijlage

### Example

```
create regular tablespace tsexploringdb2
managed by database using
(file 'db2\container1' 32,
 file 'db2\container2' 32,
 file 'db2\container3' 32)
extentsize 4 prefetchsize 8

create table order_table (
orderid smallint not null primary key,
client char(5),
orderdate date)
in tsexploringdb2

create table orderlines (
lineid smallint not null primary key,
product char(5),
orderid smallint not null references order_table on delete cascade)
in tsexploringdb2

create table payment_table (
paymentid smallint not null primary key,
client char(5),
orderdate date)
in tsexploringdb2

create trigger pendant_actions
after delete on orderlines
referencing old as old_row
for each row
mode db2sql
when (0 = (select count(*)
           from orderlines
           where orderid = old_row.orderid)
      )
begin atomic
insert into payment_table
select *
from order_table
where orderid = old_row.orderid ;
delete from order_table
where orderid = old_row.orderid ;
end

insert into order_table
values (1 , 'ABIS' , '01.08.2002')

insert into order_table
values (2 , 'KBC' , '12.09.2002')

insert into orderlines
values (1 , 'MAP' , 1)

insert into orderlines
values (2 , 'LAT' , 1)
```

```
insert into orderlines
values (3 , 'PEN' , 1)

insert into orderlines
values (4 , 'STOEL' , 2)

insert into orderlines
values (5 , 'PC' , 2)

select client, orderdate, O.orderid, lineid, product
from order_table O, orderlines L
where O.orderid = L.orderid

select * from payment_table
```