ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 1 - NUMMER 6, MAART 2003

OPEN CURSOR

Onzekerheid troef! De huidige politieke en socio-economische toestand maakt het voor de meeste bedrijven erg moeilijk om ver vooruit te kijken. Belangrijke beslissingen - ook in de IT - worden steeds opnieuw uitgesteld.
Keuzes voor nieuwe platformen en infrastructuur, operating systemen of applicatieontwikkelomgevingen zijn nu even niet aan de orde.
Als ze toch aan de orde zijn, blijkt een fundamentele mentaliteitsverandering: de ROI van de klant, van de 'business case' staat terug centraal. Ook in IT!
Wat dan indien u gemotiveerd bent om met nieuwe technologieën aan de slag te gaan? Wel, u kan zich alvast voorbereiden: om technologieën te kunnen evalueren, en later te gebruiken, hebt u alvast nood aan relevante kennis
Hopelijk draagt Exploring DB2 hier haar steuntje toe bij!
Veel leesgenot.
Het ABIS DB2 team.

IN DIT NUMMER:

CLOSE CURSOR

Het volgende nummer is volledig gewijd aan Java, of meer concreet: hoe elke DB2-specialist zich moet bewust zijn van de impact van Java op DB2. Topics als JDBC, SQLJ en typische data-access-architecturen zullen worden bekeken. En we eindigen met een FAQ.

In een extra dik nummer - niet te missen - tot dan!

Large Objects in DB2 for OS/390 - 1

Diane Hendrix (ABIS)

In dit eerste artikel bespreken we enkele algemeenheden over LOB's:

- wat zijn LOB's?
- hoe worden ze bewaard?
- hoe kunnen we tabellen met LOB-kolommen aanmaken?
- hoe gaan we LOB's gebruiken?

Een meer technische bespreking zal volgen in een tweede artikel dat verschijnt in één van de volgende nummers van Exploring DB2. Hierin zullen we bespreken:

- wat zijn LOB-locators en hoe worden ze gebruikt?
- hoe zit het met LOB's en locking?
- DB2 utilities en LOB's?

Wat zijn LOB's?

Sinds V6 van DB2 UDB for OS/390 bestaat de mogelijkheid om met Large Objects (LOB's) te werken in DB2 databases. LOB's kunnen gebruikt worden om grote dataobjecten, zoals geluidsbestanden, beeldfragmenten, foto's of grote tekstbestanden op te slaan. Als we even vergelijken: voor DB2 V6 was de maximale grootte van een kolomwaarde 32 KB (LONG VARCHAR datatype). Met de komst van de LOB datatypes kan die grootte oplopen tot 2 GB. Op dit moment is het echter wel zo dat LOB's meestal gebruikt worden voor het opslaan van kleinere hoeveelheden data.

Naargelang het soort data dat men in een LOB wil opslaan, kan men gebruik maken van drie verschillende LOB-datatypes:

- CLOB (Character Large OBject): een character string van een variabele lengte, geassocieerd met een welbepaalde code page (CCSID: Coded Character Set IDentifier). CLOB's worden voornamelijk gebruikt voor het opslaan van grote tekstdocumenten, die geschreven zijn in 1 CCSID (XML-documenten, tekstfiles, ..).

Maximale grootte: 2 147 483 647 bytes (2 GB - 1 byte).

- BLOB (Binary Large OBject): een binaire string van een variabele lengte, niet geassocieerd met een code page. De data wordt zonder dataconversie in DB2 opgeslagen. BLOB's worden voornamelijk gebruikt voor het opslaan van beeldmateriaal, geluidsfragmenten of tekstfiles afkomstig van andere platformen. Ook data afkomstig van UDF's (User Defined Functions) en UDT's (User Defined Types) kunnen bewaard worden in BLOB's.

Maximale grootte: 2 147 483 647 bytes (2 GB - 1 byte).

▪ DBCLOB (Double Byte Character Large OBject): een double-byte-characterstring van een variabele lengte, geassocieerd met een specifieke codepage. DBCLOB's worden gebruikt voor het opslaan van grote tekstdocumenten waarvoor een single-byte-datatype niet volstaat.

Maximale grootte: 1 073 741 824 DBCS characters.

Naast deze drie LOB-datatypes, zijn er sinds DB2 V6 ook nog twee nieuwe datatypes beschikbaar die gebruikt worden om LOB's te manipuleren:

▪ LOB-locators: de drie types van LOB-locators (CLOB-, BLOB-, en DBCLOB-locator) worden gebruikt in applicaties om niet rechtstreeks met de LOB-data te werken, maar met een referentie naar de LOB-data. Dit heeft voornamelijk een invloed op de performance (zie verder in het vervolgartikel over LOB's).

▪ ROWID's: datatype dat gebruikt wordt om een rij in een DB2-tabel uniek te maken. De ROWID's worden door DB2 zelf gegenereerd. Een tabel met een LOB-kolom moet verplicht een kolom van het ROWID-type hebben.

Om het werken met LOB's te vereenvoudigen, zijn er een aantal DB2 Extenders voorzien. Deze DB2 Extenders (o.a. Text, Audio, Image en Video Extender) voorzien alle methodes die nodig zijn om op een relatief eenvoudige manier LOB's te manipuleren. Met name voorzien zij in een aantal UDF's, stored procedures en triggers.

In dit artikel zal voornamelijk besproken worden hoe men LOB's benadert en manipuleert zonder gebruik te maken van DB2 Extenders. Graag verwijzen wij naar een aantal andere nummers van 'Exploring DB2', waarin aan deze extenders reeds aandacht werd besteed.

Hoe worden LOB-data bewaard in DB2?

In DB2 is de grootte van een tablespace beperkt. Ook de lengte van een record kan niet groter zijn dan 32 KB. Wanneer men LOB-kolommen in een tabel opneemt, zou men al snel in problemen komen met de eerder vermelde beperkingen. Om deze reden (en natuurlijk ook om redenen van performance) wordt een LOB-waarde niet bewaard in de eigenlijk tabel (base table) zelf, maar in een hulptabel (auxiliary table). In de basistabel wordt enkel een kolom voorzien met de LOB-descriptor (ook wel LOB-indicator genoemd). Deze bevat controle-informatie die, samen met de ROWID-waarde, gebruikt wordt om de eigenlijke LOB-waarde in de hulptabel te benaderen. De hulptabel en basistabel zitten elk in hun eigen tablespace, en behoren verplicht tot dezelfde database.

Indien men een basistabel zonder partities heeft, moet er 1 LOB tablespace en 1 hulptabel per LOB-kolom van de basistabel voorzien worden. Is de basistabel wel gepartitioneerd, dan is er een 1 LOB-tablespace en 1 hulptabel vereist voor elke partitie en per LOB-kolom.

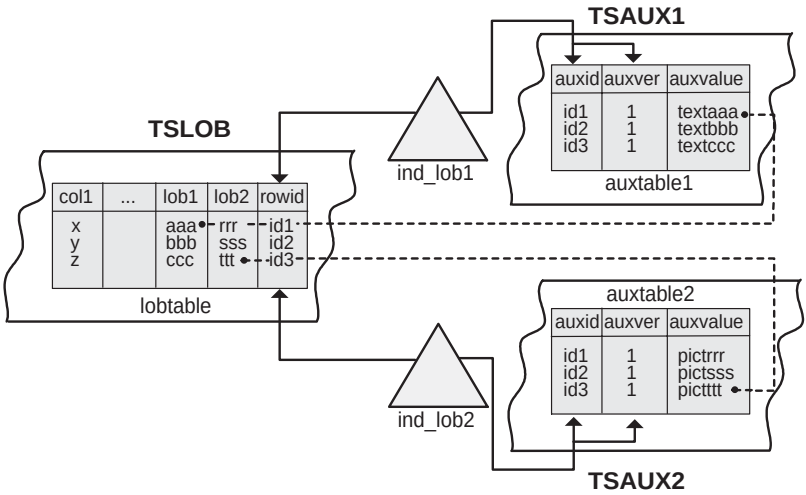
Een LOB-waarde zelf kan maximaal tot 2 GB groot zijn en zal dus meestal verschillende pages innemen. Voor een LOB-tablespace worden er dan ook zogenaamde chunks gealloceerd (groep van 16 aaneensluitende pages). Indien de nodige ruimte geen veelvoud is van 16 kunnen er bovendien enkelvoudige pages toegewezen worden.

De maximale grootte van een niet-gepartitioneerde LOB-tablespace, en dus de maximale ruimte die 1 LOB-kolom inneemt is 16 TB. Voor een gepartitioneerde tablespace kan dit oplopen tot 4064 TB!

Indien een DB2 Extender gebruikt wordt om een LOB te benaderen, kunnen de data in sommige gevallen bewaard worden in hun origineel formaat buiten DB2 (bijvoorbeeld JPG, GIF, EPS, Microsoft Word). De informatie i.v.m. de lokatie van de bestanden moet dan wel opgeslagen worden in supplementaire tabellen (de zogenaamde support tables).

Figuur 1 geeft een overzicht van de relatie tussen een basistabel 'lobtable' in tablespace 'tslob' en zijn hulptabellen 'auxtable1' en 'auxtable2' in respectievelijk 'tsaux1' en 'tsaux2'. De basistabel bevat 2 LOB-kolommen, en de verplichte kolom van het type 'rowid'. Deze laatste laat toe om via een LOB-index ('ind_lob1' en 'ind_lob2'), de LOB-waarde te vinden in de hulptabel. Beide LOB-kolommen bevatten de boven besproken 'LOB-descriptors'.

Figuur 1: relatie tussen basistabel en hulptabel



Hoe wordt een tabel aangemaakt met 1 of meer LOB-kolommen?

Wanneer men een tabel wil aanmaken met 1 of meerdere LOB-kolommen, moet men als volgt te werk gaan.

- Maak een basistabel aan (in een eigen tablespace). Deze basistabel bevat volgende kolommen:

1 primary key kolom en eventuele andere kolommen met de 'gewone' datatypes; 1 kolom van het type ROWID (liefst gedefinieerd als GENERATED ALWAYS en altijd NOT NULL); 1 of meer LOB-kolommen met type en grootte (eventueel NOT NULL).

Na het uitvoeren van het CREATE statement wordt in de basistabel elke LOB-kolom aangeduid met een LOB-descriptor (varchar 4). Deze bevat (eventueel) een NULL-indicator, een lengte-indicator, flaginfo en een versienummer van de LOB. De flaginfo op zijn beurt bestaat uit een NULL flag, een zero-length flag en een invalid-LOB flag en wordt door DB2 gebruikt om niet onnodig de hulptabel te moeten benaderen wanneer 1 van deze drie flags opstaat.

- Maak voor elke LOB-kolom van de basistabel een tablespace aan van het LOB-type.

- Maak voor elke LOB-kolom een hulptabel aan in zijn eigen LOB-tablespace, waarbij verwezen wordt naar de juiste basistabel en LOB-kolom. Elke hulptabel heeft dezelfde drie kolommen AUXID, AUXVER en AUXVALUE: AUXID bevat de ROWID-waarde van de overeenkomstige rij in de basistabel; AUXVER bevat het huidig versienummer van de LOB (nummer van de op dit moment opgeslagen versie van de LOB); AUXVALUE bevat de eigenlijke LOB-data.

- Definieer voor elke hulptabel een unieke hulpindex (auxiliary index). Opnieuw bepaalt DB2 zelf op welke kolommen de index gedefinieerd wordt (AUXID en AUXVER), en hoeft de creator geen kolomnamen mee te geven.

Voorbeeld: zie *Create van een tabel met LOB column* op p. 13.

Bij dit scenario wordt ervan uitgegaan dat CURRENT RULES(DB2) operationeel is. Indien voor CURRENT RULES(STD) gekozen wordt, zorgt DB2 zelf voor de LOB-tablespace, de hulptabel en de hulpindex.

Hoe worden LOB's gebruikt?

LOB-kolommen kunnen benaderd worden via de gewone SQL statements (SELECT, INSERT, UPDATE, DELETE). In het SELECT statement is het aantal functies echter beperkt. Men kan stellen dat alle clausules waarvoor DB2 de LOB-waarden moet vergelijken, zoals GROUP BY, HAVING, ORDER BY, niet toegelaten zijn voor een LOB-kolom. Wel toegelaten zijn o.m. CONCAT, SUBSTR, LENGTH, VALUE, COALESCE, IFNULL, POSSTR, LIKE en casting van LOB's naar andere datatypes.

Een DELETE van een rij met 1 of meerdere LOB-kolommen, resulteert in een deallocatie van de LOB-datapages. De fysieke verwijdering en dus hergebruik van de LOB-datapages kan pas plaatsgrijpen wanneer er geen enkele applicatie die data nog benadert.

Een UPDATE van een LOB-waarde wordt door DB2 vertaald in een DELETE-actie, gevolgd door een INSERT-actie. Hierbij worden de LOB-datapages eerst gedealloceerd en vervolgens wordt de nieuwe LOB-waarde op dezelfde datapages weggeschreven. Dit kan enkel wanneer er geen enkele applicatie terzelfdertijd dezelfde LOB-waarde benadert via een SELECT. Indien dit het geval is, zal de nieuwe LOB-waarde op andere LOB-datapages weggeschreven worden.

Als je vanuit een applicatie een LOB-waarde wil benaderen kan je op twee manieren te werk gaan:

- Ofwel breng je de volledige LOB-waarde binnen in een host variabele. Die moet dan wel groot genoeg zijn om de LOB-waarde te bevatten. Wanneer men op deze manier te werk gaat, brengt DB2 de volledige LOB-waarde via de bufferpool binnen in de address space van de gebruiker. Als het over een grote LOB gaat neemt dit niet alleen veel ruimte in beslag, maar zal ook de performance verminderen!

- Ofwel maak je gebruik van een LOB-locator. Door gebruik van een LOB-locator wordt slechts een 4-byte variabele gebruikt als een soort van interne referentie naar de echte LOB-waarde. In de applicatie kan perfect met de LOB-locator gewerkt worden overal waar men ook met een host variabele kan werken.

Besluit

De mogelijkheid om LOB's te gebruiken in DB2 for OS/390 kadert in de zogenaamde 'object-relational extensions' die vanaf V6 aan DB2 for OS/390 werden toegevoegd. Andere extensions zijn de UDF's, UDT's, triggers en extenders. Het gebruik van deze nieuwe features creëert een aantal bijkomende mogelijkheden voor DB2 databeheer en voor applicaties die deze databases benaderen. LOB's vormen onder andere een oplossing voor het probleem van 'te grote' data.

Let echter wel op: het gebruik van de LOB-functionaliteit in een database systeem leidt in vele gevallen tot een daling van de performance. Gebruik LOB's dus enkel wanneer het echt noodzakelijk is, en dus zeker niet voor gegevens die de limiet van een normale kolom net overschrijden. In dit kader dient zeker opgemerkt te worden, dat LOB's geen alternatief vormen voor normalisatie!

ABIS IN MUNCHEN

U kent ABIS als een bedrijf dat zich reeds meer dan 10 jaar verdiept in DB2. Permanent volgen we de technologische evolutie binnen DB2 van zeer nabij op. Dit alles uitgediept aan de hand van praktische ervaring opgedaan bij interne projecten of bij adviesopdrachten bij klanten.

Deze kennis komt tot uiting in onze cursussen, in seminars, en in de nummers van 'Exploring DB2'. ABIS medewerkers nemen echter ook deel aan internationale events. Graag brengen wij volgende presentatie onder uw aandacht.

Event: IBM Software Symposium

Datum: 19 - 23 mei, 2003, Munchen

Titel: WORF: Het ontwikkelen van DB2 UDB-based web services op een Websphere Application Server

Track: E-business integratie

Spreker: Pieter Bedert

Abstract:

De IT uitdaging van vandaag - het realiseren van een integratie tussen DB2 data en procedures enerzijds, en e-business applicaties anderzijds. Deze presentatie legt de nadruk op DB2 Versie 8 georiënteerde oplossingen voor de ontwikkeling van Web Services op een WebSphere applicatieserver. Alle vereiste ontwikkelcomponenten komen aan bod: WORF, XML, DB2 stored procedures, en natuurlijk DADX. De verschillende stappen in het ontwikkel- en deployementproces worden behandeld en geïllustreerd aan de hand van een concreet voorbeeld.

De DB2 XML Extender - 2

Tom Avermaete (ABIS)

Of hoe DB2 omgaat met XML-documenten...

In een eerste deel van de bespreking van de DB2 XML Extender hebben wij de Columnmethode uitgebreid besproken. Bij gebruik van deze methode wordt het volledige XML-document - inclusief tags - in een speciale XML-kolom bewaard. DB2 beschikt hierbij over de mogelijkheid om een XML-document te valideren aan de hand van een DTD (Document Type Definition) waarin de structuur beschreven staat. Deze DTD's worden bewaard in de DTD repository. Bij het inserten van een XML-document in de speciaal hiervoor bestemde DB2-kolom kunnen bepaalde velden uit het XML-document naar DB2-tabellen, zogenaamde side-tables, worden overgezet. Deze overzetting gebeurt aan de hand van een DAD (Document Access Definition) bestand waarin de instructies staan over hoe een XML-document gemapt moet worden naar deze side-tables. Om de velden binnen het XML-document aan te duiden, wordt gebruik gemaakt van de XML-querytaal XPath.

In dit deel willen we dieper ingaan op het gebruik van de Collectionmethode.

Een document toevoegen met de Collectionmethode

Na het 'enablen' van de database - een feature uiteengezet in Exploring DB2 nummer 4 - maakt DB2 het schema DB2XML aan met de bijhorende User Defined Types (UDT), User Defined Functions (UDF) en stored procedures die gebruikt worden in de Collectiontechniek. Ook de tabellen DTD_REF en XML_USAGE worden gedefinieerd.

In tegenstelling tot de Columnmethode wordt het XML-document zelf niet bewaard in speciale DB2-tabellen. Ook de data die in het XML-document vervat zitten, worden niet in speciale tabellen opgeslagen. Deze techniek lijkt dus zeer geschikt in die situaties waarbij XML als techniek wordt gebruikt om op een gestructureerde wijze gegevens uit te wisselen.

Deze techniek heeft dan ook als gevolg:

- dat we geen speciale XML-datatypes moeten gebruiken om met de Collectionmethode te werken: XML-documenten worden immers niet bijgehouden;
- dat de bekende 'side-tables', eigen aan het werken met de Columnmethode, niet zullen worden gebruikt. De gegevens worden met de Collectionmethode immers rechtstreeks bewaard in gewone DB2-tabellen.

Net zoals bij de Columnmethode wordt de mapping tussen het XML-document en de DB2-tabellen beschreven in een DAD-document

(voorbeeld 1). De structuur van deze DAD ligt vast en wordt gevalideerd door het bestand DAD.DTD.

Voorbeeld 1: een typische DAD

```
<DAD>
<Xcollection>
<SQL_stmt>select RTRIM(name) as name, RTRIM(location, firstname)
          from seminars, speakers
          where seminars.id = seminar_id
          and company_name = 'ABIS'
</SQL_stmt>
</Xcollection>
...
</DAD>
```

Eenmaal de DAD aangemaakt kan de Collection geënabled worden - dit is echter optioneel. We beschikken dus over 2 opties:

- Ofwel specificeren we de te gebruiken DAD bij het oproepen van Collection-specifieke stored procedures - Shred, Insert, etc. Enablen van de Collection is niet vereist.
- Ofwel kiezen we voor het enablen van de Collection. Vervolgens kan de naam van de Collection worden gebruikt bij het aanroepen van specifieke procedures. De informatie die aangeleverd wordt door de DAD, wordt dan opgeslagen in de DB2XML.USAGE-tabel.

Een document ophalen met de Collectionmethode

Niet enkel inserten is mogelijk met behulp van de Collection techniek. Ook het omgekeerde, het zogenaamde composen of samenstellen is mogelijk. Ook hiervoor wordt gebruik gemaakt van een DAD waarin ondermeer SQL gebruikt wordt om de relaties te leggen tussen de DB2-tabellen en het XML-document. Misschien overbodig hierbij te zeggen dat gelijk welke gegevens uit gelijk welke DB2-tabellen kunnen gebruikt worden om dit XML-document samen te stellen. Voorbeeld 2 toont een eenvoudige SQL om een XML-document te composen.

Voorbeeld 2: een XML document genereren aan de hand van een SQL

```
select '<P> <name>', name, '</name>', '<firstname>', fname,
'</firstname> </P>'
from persons
where PNO = 1
```

Meer complexe scenario's zijn natuurlijk ook mogelijk met behulp van meer ingewikkelde SQL. Situaties, waarbij niet alleen het XML-document wordt samengesteld op basis van data die zich bevinden in verschillende tabellen, maar waarbij ook de XML-documentstructuur in detail kan worden geëxpliciteerd. In DB2 kan dit gebeuren aan de hand van een DAD (voorbeeld 3). Merk op dat het originele document niet meer kan worden opgehaald uit de databases - het werd immers alsdusdanig niet opgeslagen!

Voorbeeld 3: een XML document genereren aan de hand van een DAD

```
...
<Xcollection>
<SQL_stmt>select plname, coname, cltitle, sdate from
tbaccad.tutenrolments, tbaccad.tutsessions, tbaccad.tutcourses,
tbaccad.tutpersons, tbaccad.tutcompanies where e_sno = sno and s_cid =
cid and e_pno = pno and e_cono = cono and pno = 104 and sno = 1 order
by plname, coname;</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Confirm SYSTEM
"d:\library\udb\XMLextender\xml\composingXML\confirm.dtd"</doctype>
<root_node>
<element_node name="Confirm">
  <element_node name="Enrollee">
    <element_node name="LastName">
      <text_node>
       <column name="plname"/>
      </text_node>
...
```

Zijn XML-extenders noodzakelijk?

Vooraleer u vol ijver met deze nieuwe XML-functionaliteiten in DB2 aan de slag wil, misschien eerst nog enkele kritische kanttekeningen die ook nuttig zijn voor OS/390-gebruikers of voor gebruikers van andere databases zonder XML opties. Zowel voor de Collection- als voor de Columnmethode bestaan er immers heel wat alternatieven.

Een XML-document kan bijvoorbeeld helemaal 'genormaliseerd' worden naar een drietal DB2-tabellen. Eén tabel zal hierbij een rij hebben voor elk XML-document, een tweede tabel zal een rij hebben voor elk XML-element en een derde tabel zal een rij hebben voor elk attribuut. In de tabel met de XML-elementen ligt er een foreign key naar de eerste tabel met de XML-documenten. In de tabel met de attributen ligt een foreign key naar de tweede tabel met de XML-elementen. Elk XML-element zal bovendien ook twee nummers hebben die de plaats aangeven van de openings- en sluitingstag in het document. De SQL-statements die deze misschien wat ingewikkelde inserts gaan beschrijven kunnen gegenereerd worden met behulp van XSLT, een XML-taal die toelaat om XML-documenten - in dit geval het XML-document dat we willen gaan inserten - om te zetten naar SQL. Het XML-document kan terug samengesteld worden met behulp van een (complex) SQL-statement.

Moet onze database dan eigenlijk wel belast worden met al die XML-tags? Voor data-georiënteerde XML-documenten kan de XML-markup gemakkelijk uitgeschakeld worden omdat de structuur en de datatypes redelijk vast liggen. Voor document-georiënteerde gegevens zoals teksten, documentatie of html-pagina's liggen de kaarten anders. Met een te strakke structuur zouden onze eindgebruikers immers teruggebombardeerd worden naar het pre-tekstverwerker tijdperk. Een gulden middenweg is dus ook hier weer de beste oplossing.

Meer info: *http://www7b.software.ibm.com/dmdd/library/techarticle/0203lima/0203lima.html*

DOSSIER 7

Row Expressions

Verwacht u ook een negatieve sqlcode bij het uitvoeren van volgend SQL-statement?

```
select * from sysibm.syscolumns
where (tbname, tbcreator) in (select name, creator
        from sysibm.systables where dbname = 'ABIS')
```

In DB2 V7 voor OS/390 zal u die niet krijgen. Let wel op; dit geeft niet hetzelfde resultaat als:

```
select name, tbname, tbcreator from sysibm.syscolumns
where tbname in (select name from sysibm.systables
                 where dbname = 'ABIS')
and tbcreator in (select creator from sysibm.systables where dbname= 'ABIS');
```

Een eenvoudig voorbeeld maakt bovenstaand verschil duidelijk. Tabelsub (een tabel met twee kolommen) bevat de waarden (a,3), (b,3), (b,4). Tabelhoofd bevat de waarden (a,3), (a,4).

Bekijk nu de volgende 3 query's:

Query 1:
```
select * from tabelhoofd
  where (col1, col2) in (select col1, col2 from tabelsub);
```

Query 2:
```
select * from tabelhoofd
  where col1 in (select col1 from tabelsub)
  and col2 in (select col2 from tabelsub);
```

Query 3:
```
select * from tabelhoofd h
  where exists (select * from tabelsub s
          where h.col1 = s.col1 and h.col2 = s.col2);
```

Wanneer je de query 1 uitvoert worden de koppels van de hoofdtabel in hun geheel vergeleken met de koppels van de subtabel. (a,3) zit ertussen en (a,4) niet; we krijgen dus als resultaat (a,3). Bij query 2 daarentegen worden apart de waarden van de hoofdtabel met die van de subtabel vergeleken: a komt voor in de eerste kolom van de subtabel, 3 komt voor in de tweede kolom en 4 ook, wat als resultaat geeft (a,3), (a,4). Query 3 geeft dan weer wel hetzelfde resultaat als query 1: wanneer men een gecorreleerde subquery uitvoert, gaat men rij per rij te werk. Wanneer men de rij met de waarde (a,4) neemt en hiervoor de subquery gaat uitvoeren, dan levert dit geen resultaat op. Query 3 heeft dus inderdaad als resultaat (a,3).

Zowel in, not in, =some, =any, <>all zijn bruikbaar in de where conditie. Voor een 'basic predicate' en 'quantified predicate' row expression zijn >, >=, <, <= echter niet mogelijk.Ook =all, <>some, <>any zijn niet mogelijk.

Katrien Platteborze (ABIS)

CURSUSPLANNING MRT - APR - MEI 2003

DB2 for OS/390, een totaaloverzicht	1625 EUR	07-11/04 (W), 22-28/05 (W), 02-06/06 (L)
DB2 UDB, een totaaloverzicht	1625 EUR	24-28/03 (L), 22-23/5 & 02-04/06 (W)
RDBMS concepten	325 EUR	24/03 (L), 07/04 (W), 22/05 (W), 02/06 (L)
Basiskennis SQL	325 EUR	25/03 (L), 08/04 (W) 23/05 (W), 03/06 (L)
DB2 for OS/390 basiscursus	975 EUR	26-28/03 (L), 09-11/04 (W), 26-28/05 (W), 04-06/06 (L)
DB2 UDB basiscursus	975 EUR	26-28/03 (L), 02-04/06 (W)
DB2 UDB concepten	375 EUR	05/06 (L)
SQL workshop	700 EUR	22-23/04 (W), 12-13/06 (W), 16-17/06 (L)
DB2 for OS/390 programmering voor gevorderden	1050 EUR	02-04/04 (W), 19-21/05 (L)
DB2 for OS/390: SQL performance	1200 EUR	11-13/06 (L)
Fysiek ontwerp van relationele databases	700 EUR	28-29/04 (L)
DB2 for OS/390 database administratie	1600 EUR	12-15/05 (L)
DB2 UDB database administratie	1600 EUR	05-08/05 (L)
DB2 UDB systeembeheer en performance	400 EUR	21/03 (W), 09/05 (L)
DB2 for OS/390 V7 upgrade voor ontwikkelaars	375 EUR	06/06 (L)
DB2 UDB en zijn extenders: XML en text search	200 EUR	28/03 (W), 16/05 (L)
DB2 UDB integratie met MQSeries	200 EUR	28/03 (W), 16/05 (L)

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

Create van een tabel met LOB column

```
-- CREATIE VAN TABLE SPACE VOOR BASISTABEL
-----------------------------------------
  CREATE TABLESPACE TSLOB
  IN TBDB0XXX;

-- CREATIE VAN 2 LOB TABLE SPACES VOOR HULPTABELLEN
-----------------------------------------------
  CREATE LOB TABLESPACE TSAUX1
  IN TBDBXXX
  LOCKSIZE LOB;

  CREATE LOB TABLESPACE TSAUX2
  IN TBDBXXX
  LOCKSIZE LOB;


-- CREATIE VAN BASISTABEL MET 2 LOB KOLOMMEN EN ROWID
-----------------------------------------------------
  CREATE TABLE TBTLOB
  (
  PNO SMALLINT NOT NULL,
  LOBCOL1              CLOB(1M),
  LOBCOL2BLOB(1G)
  TBTLOBROWID          ROWID GENERATED ALWAYS NOT NULL,
  PRIMARY KEY (PNO)
  )
  IN TBDBXXX.TSLOB;


-- CREATIE VAN EEN UNIQUE INDEX OP DE PRIMARY KEY
-------------------------------------------------
  CREATE UNIQUE INDEX IND_PNOLOB1
  ON TBTLOB(PNO)
  ;

----CREATIE VAN 2 HULPTABELLEN IN DE LOB TABLE SPACES
-----------------------------------------------------------
  CREATE AUXILIARY TABLE AUXTABLE1
  IN TBDBXXX.TSAUX1
  STORES TBTLOB COLUMN LOBCOL1;

  CREATE AUXILIARY TABLE AUXTABLE2
  IN TBDBXXX.TSAUX2
  STORES TBTLOB COLUMN LOBCOL2;

--  CREATIE VAN UNIQUE INDEX OP DE HULPTABELLEN
-----------------------------------------------
  CREATE UNIQUE INDEX IND_LOB1
  ON AUXTABLE1;

  CREATE UNIQUE INDEX IND_LOB2
  ON AUXTABLE2;
```