ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 4 - NUMMER 4, JULI 2006

OPEN CURSOR

Terwijl wij nog volop bezig zijn met de migratie van DB2 naar versie 8, en het ontdekken van nieuwe mogelijkheden, is men bij IBM al volop bezig met versie 9, tot voor kort nog voorzichtig "Vnext" genoemd. Ondertussen is hij er dus: u kan de beta-versie van DB2 v9.1 voor z/OS nu reeds uitproberen. Met als belangrijkste blikvanger een volledig geïntegreerde ondersteuning van XML, en daarnaast (nog meer dan in versie 8) een gelijktrekken van de functionaliteiten met versie 9 voor Linux, Unix en Windows.
Met Exploring DB2 concentreren we ons in dit nummer toch nog vooral op de mogelijkheden van v8, maar houden we u in de komende nummers uiteraard op de hoogte van de verdere ontwikkelingen van ons aller DB2.

Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- Aandacht voor nieuwe partitioneringsmogelijkheden in *Tabelpartitionering in DB2 versie 8*.
- Waarom ook u met unicode in aanraking zal komen, leest u in *Tekstcodeerschema's en Unicode met DB2 v8*.
- Het derde deel rond content management, in *DB2 en content management - 3: zoekfaciliteiten*.
- *Tabelimplementatie in DB2, Oracle en SQL Server* - hoe gaan deze leveranciers om met het logische concept 'tabel'?
- *Cursusplanning augustus 2006 - december 2006*

CLOSE CURSOR

In het volgende nummer volgt nog een laatste aflevering in de serie DB2 Content Management, en verder een bijdrage over Visual Explain, en een praktische toepassing van recursieve SQL, nog zo'n nieuwe mogelijkheid van DB2 v8.

Tot dan!

Tabel-partitionering in DB2 versie 8

Peter Vanroose (ABIS)

Eén van de belangrijkste nieuwigheden voor de DBA in DB2 versie 8 is de sterk toegenomen flexibiliteit bij het manipuleren van partitioned table spaces, met als belangrijkste aspect de loskoppeling van de concepten 'datapartitionering' en 'dataclustering'.

In versie 7 kan men een tabel in een partitioned table space fysisch alleen partitioneren (op basis van de waarden in een bepaalde kolom of kolommen) door aan de tabel een 'partitioning index' te koppelen met een lijst van grenswaarden, één per partitie via 'PART n VALUES (waarde)'. De partitioning index is daardoor ook altijd de clustering index.

In versie 8 moet men niet langer een index gebruiken om de partities te specificeren: de syntax van CREATE TABLE werd uitgebreid zodat deze grenswaarden nu rechtstreeks aan de tabel kunnen gekoppeld worden.

```
CREATE TABLE tabel (...)
PARTITION BY (kolom)
 (PARTITION 1
     ENDING AT (waarde_1),
  ...
  PARTITION n
      ENDING AT (waarde_n))
```

De grenswaarden worden nu uiteraard in SYSIBM.SYSTABLEPART bijgehouden i.p.v. in SYSIBM.SYSINDEXPART.

Een gevolg hiervan is dat er nu los van het partitioneringscriterium een CLUSTER index kan gecreëerd worden, mogelijk dus op andere kolommen dan de partitioneringskolom(men).

Een voorbeeld: stel dat we een tabel 'ORDERS' hebben die alle bestellingen van de voorbije 10 jaar bijhoudt. Partitioneren op basis van de kolom 'ORDER_DATE' kan dan zinvol zijn: b.v. elk jaar of elke maand wordt in een aparte partitie geplaatst.

Voorbeeld 1 en 2

```
CREATE INDEX ci ON orders(customer_name) CLUSTER

CREATE INDEX ci ON orders(customer_name) PARTITIONED
```

Onderstel nu dat de tabel meestal ondervraagd wordt op basis van de klantennaam 'CUSTOMER_NAME'. Deze kolom is dus een goede kandidaat voor data-clustering. Creëer hiervoor een CLUSTER index - zie voorbeeld 1.

Het 'probleem' is natuurlijk dat deze index over alle partities heen loopt, en dus voortdurend heen en weer moet springen tussen partities. Terwijl een query misschien meestal als bijkomende conditie een begin- en einddatum zal opgeven, waardoor de gevraagde orders uit slechts één partitie komen.

We zouden dus eigenlijk een clustering index willen die zich beperkt tot één partitie. Dit is mogelijk: gebruik hiervoor het nieuwe keyword PARTITIONED bij de creatie van de index - zie voorbeeld 2.

Deze 'data partitioned' index loopt nog steeds over de hele tabel, maar is 'onderverdeeld' volgens de partities. Zijn index-space is dus effectief eveneens gepartitioneerd!

Let op: dit is geen 'data partitioning' index: dat zou een index moeten zijn op de kolom ORDER_DATE, maar die index hoeft er nu niet meer te zijn vermits de partitionering op tabelniveau gespecificeerd kan worden.

Een index kan uiteraard zowel 'data partitioned' als 'clustering' zijn; deze combinatie is nuttig in ons voorbeeld; maar uiteraard kan een tabel maar één clustering index hebben, terwijl er meerdere data partitioned indexen kunnen zijn.

Merk op dat PARTITIONED indexen niet UNIQUE mogen zijn: alhoewel ze per partitie uniciteit zouden kunnen garanderen, kunnen ze dat niet over de hele tabel vermits de afzonderlijke index-partities autonoom werken.

DPSIs en NPSIs

Een primaire index is elke index die 'partitioning' is, die dus als eerste kolom(men) de kolom(men) heeft waarop de partitionering gedefinieerd is - de kolom ORDER_DATE in ons voorbeeld.

Een secundaire index is een 'non-partitioning index' (NPI), die dus niet de partities één na één doorloopt. Zo'n index kan gepartitioneerd zijn of niet. In het eerste geval spreekt men van een DPSI (spreek uit: 'dipsy'), een 'data partitioned secondary index', in het tweede geval van een NPSI, een 'non-partitioned secondary index'.

DPSIs kunnen nuttig zijn om lock contention te vermijden, vooral dan in combinatie met utilities: een DPSI bestaat eigenlijk uit aparte stukken, één per partitie, zodat index locking eindigt aan een partitiegrens. Een gelijktijdig lopend utility kan dan ondertussen zonder problemen aan de andere partities werken. Bij gebruik van een NPSI is dit niet mogelijk omdat wijzigingen aan een NPSI repercussies hebben over alle partities heen; locking is dus veel globaler.

Uiteraard is het gebruik van een DPSI om informatie uit alle partities op te vragen minder performant dan het gebruik van een NPSI, terwijl bovendien het locking-voordeel vervalt. Definieer DPSIs dus alleen als queries typisch bijkomend een voldoende restrictief predikaat bevatten op de partitioning-kolom.

Roteren van partities

Als gevolg van het overstappen van index-gecontroleerde partitionering naar tabel-gecontroleerde partitionering, zijn er enkele nieuwe syntactische mogelijkheden van ALTER TABLE beschikbaar geworden die het onderhouden van partitioned tablespaces een stuk vergemakkelijken.

Zo bestaat er nu in versie 8 de mogelijkheid om een partitie toe te voegen aan een bestaande tabel, met 'ALTER TABLE ADD PARTITION ENDING AT (waarde)'. Ook kan me nu gemakkelijk partities 'rouleren', d.w.z. de 'oudste' partitie (in ons voorbeeld van ORDERS) hergebruiken voor nieuwe data van de volgende maand. Dit kan met 'ALTER TABLE ALTER PART ROTATE FIRST TO LAST'. De nieuwe 'laatste' partitie is daarna automatisch leeg en klaar voor gebruik: geen REORG nodig! De oude data in die partitie zijn natuurlijk niet langer beschikbaar (en moeten eventueel geUNLOADed worden vóór het ALTER TABLE statement). Let op: dit betekent o.a. dat de dataset met als naam '**.A001' nu niet langer de eerste maar de laatste partitie is, en '**.A002' de eerste (oudste)!

Voorbeeld 3

order_date	customer_name		
2006-05-01	Janssens	2	3
2006-05-12	Franssens	1	1
2006-05-21	Pieters	3	5
2006-06-24	Moons	2	4
2006-06-30	Franssens	1	2
clustering index, partitioning		partitioned index (DPSI)	non-partitioned index (NPSI)

Onderstel dat deze tabel met tabelgebaseerde partitionering per maand opgesplitst wordt. Onafhankelijk daarvan kan dezelfde kolom ORDER_DATE, of een andere kolom, gespecificeerd worden als clustering index.

Een DPSI op de kolom CUSTOMER_NAME bestaat effectief uit twee afzonderlijke "stukken", één met de waarden Franssens, Janssens, Pieters, en één met de waarden Franssens, Moons.

Een NPSI op diezelfde kolom bestaat uit één index met 5 waarden Franssens, Franssens, Janssens, Moons, Pieters.

Tekstcodeerschema's en Unicode met DB2 v8

Peter Vanroose (ABIS)

Sinds versie 7 ondersteunt DB2 het gebruik van Unicode voor het opslaan van tekst, maar vanaf versie 8 is Unicode de 'native encoding' geworden van DB2. Wat houdt dit precies in, en wat is Unicode nu precies? Dit proberen we in deze bijdrage kort toe te lichten.

Misverstanden

Elke presentatie over Unicode begint met de 3 misverstanden - origineel waarschijnlijk terug te brengen tot volgende publicatie: 'DB2 v8 for z/OS', IBM Redbook. Logisch hiermee te beginnen: ze vormen een mooie samenvatting betreffende de impact van Unicode op DB2 voor z/OS.

Misverstand 1: 'Bij overgang naar versie 8 moet alle data in alle tabellen naar Unicode geconverteerd worden'.

De DB2-catalog wordt weliswaar geconverteerd naar Unicode, maar alle andere data kunnen gewoon opgeslagen blijven in EBCDIC. Alleen wanneer er duidelijke voordelen zijn bij een dataconversie, b.v. om voortdurende conversies tussen EBCDIC en Unicode te vermijden voor een tabel die uitsluitend gebruikt wordt door een (niet-mainframe-)applicatie die Unicode-data verwacht, valt het te overwegen om de data als Unicode op te slaan.

Misverstand 2: 'Unicode verdubbelt altijd de vereiste stockage'.

Alle 'gewone' letters, cijfers en leestekens verbruiken nog steeds 1 byte, zoals in EBCDIC; alleen letters met accenten of trema's hebben 2 bytes nodig, en het Euroteken heeft er drie nodig.

Mistverstand 3: 'Unicode-ondersteuning in v8 zal op mij geen impact hebben'.

Toch wel: al gebeuren conversies automatisch indien nodig, b.v. bij een query op de catalog, het is op z'n minst nodig om te beseffen dat er conversies gebeuren, want dit kan gevolgen hebben voor de performantie. Er kunnen zelfs onverwachte, onaangename effecten optreden bij onzorgvuldig omspringen met CCSIDs, zoals verderop in deze tekst zal blijken.

Of zoals het zeer goed geformuleerd op een of andere website:

'*Bij DB2 voor z/OS V8 wordt Unicode iets wat je moet begrijpen. Versie 8 is fundamenteel verschillend van vorige versies doordat zowel de catalog als de parser nu Unicode spreken. Door de internationaliseringstendens in meer en meer bedrijven wordt de meerwaarde van Unicode steeds duidelijker*.'

IBM geeft bovendien zelf toe dat de stap naar Unicode min of meer afgedwongen werd door SAP, Peoplesoft en Siebel...

En dus blijft de cruciale vraag: Wat is Unicode?

Wat is Unicode? - terminologie

Een computer slaat een tekstsymbool numeriek op - een 'mapping-schema' wordt hiertoe gebruikt. Er zijn echter verschillende mappingschema's in omloop. Zo gebruikt schema EBCDIC de numerieke waarde 193 voor een teken 'A', 83 is een 'ë', 129 een 'a' en 241 het cijfer '1'. Een volledige set van dergelijke mappings wordt een code page genoemd, terwijl elke afzonderlijke toekenning een code point genoemd wordt. Inderdaad, code pages (mapping schema's) hebben een naam - de meest bekende zijn EBCDIC (mainframe based omgevingen) en ASCII (PC, UNIX based omgevingen).

DB2 gebruikt de term CCSID om naar de numerieke representatie van een codepage te verwijzen; 'code page' en 'CCSID' zijn dus nagenoeg synoniem.

Er bestaat trouwens meer dan één CCSID voor EBCDIC: verschillende landen/talen/platformen gebruiken lichtjes verschillende EBCDIC-encoderingen. Zo gebruikt code page 37 (English/USA) encodering 187 voor een ']', terwijl code page 500 (Internationaal/Europa) een 187 vertaalt als '|'; een ']' is nu 90, wat in codepage 37 als '!' getoond wordt.

Ook ASCII omvat meerdere code pages, wat al jaren voor problemen zorgt bij het uitwisselen van tekstbestanden tussen verschillende landen, platformen, of tekstverwerkers.

Tijd dus voor een 'universele' mapping tussen lettertekens en numerieke waarden. Dit is precies wat Unicode beoogt: elk letterteken, in elke mogelijke taal (inclusief Chinees, Arabisch, wiskunde, ...) wordt uniek op een positief geheel getal afgebeeld.

De 'klassieke' tekstmappings zoals EBCDIC en ASCII gebruiken 1 byte voor 1 letterteken, zodat een codepage hoogstens 256 symbolen bevat. Maar zelfs binnen één taal kan dit onvoldoende zijn; vandaar het SQL 'GRAPHIC' datatype, dat tot 65536 tekens (2 bytes) kan herbergen. Nu bleek zelfs dat niet voldoende voor Unicode. Om niet nodeloos meer dan 1 byte te moeten besteden aan veel voorkomende lettertekens, en om anderzijds toch ook niet beperkt te worden door de limiet van 65536, gebruikt Unicode variabele-lengte-encodering. Unicode laat de gebruiker toe om te kiezen uit één van drie fysische 'byte-mappings': UTF-8, UTF-16 en UTF-32. DB2 gebruikt hiervan enkel UTF-8 (CCSID 1208, voor CHAR) en UTF-16 (CCSID 1200, voor GRAPHIC). Voor alle Europese talen is UTF-8 de meest economische.

UTF-8 gebruikt één byte om de code points tot en met 127 voor te stellen, twee bytes voor de volgende code points tot en met 2047 (alle Europese talen, Arabisch, Hebreeuws), drie bytes voor code points t.e.m. 65535 (o.a. Japans, Thai, Ethiopisch, speciale tekens zoals het Euro-symbol, het promille-teken, wiskundige en grafische tekens, ...), en vier bytes voor de overige (i.h.b. Chinees).

Omdat SQL, of beter gezegd datatype CHAR, uitgaat van het idee '1 byte is 1 teken', is het gevolg van de variabele encodering van UTF-8 dat een CHAR(40) of een VARCHAR(40) mogelijk minder dan 40 lettertekens kan bevatten, in het bijzonder dus als daar andere tekens tussen zitten dan gewone letters, cijfers en leestekens. Om de garantie van '1 teken is 1 byte' te behouden, en toch UTF-8 te gebruiken, heeft DB2 de CCSID 367 ingevoerd: deze codepage bevat enkel de Unicode code points tot en met 127. Voor 'gewone' (Engelse) tekst is dit voldoende, maar accentletters, het copyright-teken of het 'niet'-teken kunnen hiermee niet voorgesteld worden.

Nog interessant om weten: deze 128 eerste code points van Unicode zijn identiek aan die van alle ASCII-encoderingen. Een ASCII-tekstbestand zal dus bij conversie naar UTF-8 fysisch niet wijzigen als er geen accentletters of andere 'ongewone' tekens in staan, en vice versa. ASCII en EBCDIC daarentegen zijn totaal verschillende encoderingen.

Zowel Unicode als EBCDIC gebruiken opeenvolgende code points voor enerzijds de 26 hoofdletters, anderzijds de 10 cijfers, anderzijds de 26 kleine letters, maar jammer genoeg is de onderlinge volgorde verschillend in Unicode en in EBCDIC: deze laatste heeft 'ë' < 'a' < 'A' < '1' (zie hierboven), terwijl voor ASCII en Unicode '1' < 'A' < 'a' < 'ë'. Als gevolg hiervan zal data die als Unicode opgeslagen is binnen DB2, anders sorteren dan data die als EBCDIC is opgeslagen. Dit heeft dus gevolgen voor i.h.b. de ORDER BY subclause, een BETWEEN-conditie, en de MIN/MAX functies, maar ook voor clustering sequences en tabelpartitionering.

Unicode in DB2 versie 8

- De hele catalog en directory worden tijdens de install van DB2 v8 geconverteerd (op enkele tabellen na) naar UTF-8. Dit gebeurt op een volledig transparante manier voor de gewone gebruiker: alle queries op catalog-tabellen blijven werken, maar moeten mogelijk impliciet hun resultaat naar EBCDIC converteren, wat een kleine performatie-implicatie kan hebben. Denk er bij een UNLOAD van de catalog ook bij na of dit best gebeurt met of zonder conversie naar EBCDIC: conversie geeft het voordeel dat het bestand met de ISPF-editor kan gelezen worden, maar er zijn twee belangrijke nadelen: de conversie-overhead, maar mogelijk ook informatie-verlies! Dit laatste wordt zo dadelijk duidelijk.

DB2 houdt van elke tablespace bij met welke CCSID de encodering gebeurd is, en zal indien nodig converteren (naar Unicode) wanneer een SQL-query tabellen gebruikt met onderling verschillende CCSID. Dit laatste was in versie 7 niet mogelijk. Waarom naar Unicode? Omdat dit de enige encodering is die alle mogelijke tekens kan voorstellen. Let dus bij manuele conversies op om geen data kwijt te spelen! Zo kan een tabel een naam hebben waarin accent-letters staan: dit is in orde voor EBCDIC; maar v8 laat ook tabelnamen toe met b.v. een Poolse L, of een Euro-teken. EBCDIC heeft hiervoor geen representa-

tie, zodat in die gevallen de catalog niet meer als EBCDIC kan geUNLOADed worden. Een reden temeer dus om op te letten met naamgeving van DB2-objecten...

Conversie gebeurt niet wanneer een tabel met CCSID 500 (EBCDIC dus) of CCSID 0 ('onbekend') vanuit een lokale mainframe-applicatie wordt ondervraagd of aangepast. Evenmin is er conversie nodig wanneer een tabel met CCSID 1208 of 367 gebruikt wordt door een Windows-applicatie via ODBC. In de meeste andere gevallen zal er dus wel moeten geconverteerd worden. Het is duidelijk dat het daarbij zeer belangrijk is dat de CCSID van de bestaande data klopt: zoals gezegd zijn er zelfs kleine verschillen tussen CCSIDs 500 en 37, dus een tabel die per vergissing CCSID 37 meekreeg maar altijd vanuit een applicatie werd gestuurd die CCSID 500 veronderstelde (zoals een CICS-applicatie in een 3270-terminalemulatie), zag er goed uit in versie 7 maar zal (door de striktere toepassing van CCSIDs in versie 8) verwisselingen veroorzaken tussen de symbolen '!', '[', ']', '|', '^', '¢' en '¬'.

- Alle SQL-queries worden door DB2 v8 geparsed in Unicode. Dit zorgt er onder andere voor dat SYSIBM.SYSVIEWS correct ingevuld wordt. Maar het impliceert ook dat de DBRM-bestanden minder leesbaar zullen worden met een ISPF-editor... In het bijzonder zouden er dus kunnen problemen ontstaan (bij foutief ingestelde CCSIDs) met SQL zoals 'WHERE col ¬= 1' of 'col1 || col2'; gebruik dus bij voorkeur 'WHERE col <> 1' en 'col1 CONCAT col2' om dergelijke problemen te vermijden. Anderzijds laat dit o.a. toe om het Euro-symbool op te nemen in een WHERE-conditie.

Meer info over Unicode? zie **www.unicode.org**

Caveats - toch even stilstaan bij ...

- Andere resultaten bij een ORDER BY, BETWEEN, MIN, MAX, afhankelijk van de CCSID van de tabel (Dit was reeds het geval in v7 wanneer ASCII-codering werd gebruikt).

- Andere resultaten bij SUBSTR (die in bytes telt) wanneer een tekst accentletters bevat én die tekst in UTF-8 opgeslagen wordt.

- Foutieve CCSIDs: corrigeer die (i.h.b. CCSID 37: in Europa is 500 couranter) vooraleer naar versie 8 te upgraden.

- Foutieve code page setting in 3270-terminalemulatie. Gekende queries kunnen plots onverwachte resultaten geven.

- Performantie-issues omwille van automatische conversies.

- Toegenomen storage (voor VARCHAR) of te kleine voorziene storage (voor CHAR(n)) wanneer een tabel geconverteerd wordt van EBCDIC naar UTF-8. Maar waarschijnlijk is er (althans voorlopig) geen nood aan dergelijke c*onversie van uw d*ata, dus hoeft dit probleem zich niet te stellen.

DB2 en content management - 3: zoekfaciliteiten

Eric Venmans (ABIS)

Inleiding

In vorige artikels over de DB2 Content Manager (DB2ContMgr) werd aandacht besteed aan enerzijds de algemene architectuur van het product en anderzijds aan het onderliggende data model met zijn bestanddelen: items, componenten, objects, attributen, relationships ...

In dit derde artikel bekijken we vooral het gebruik van het DB2 ContMgr systeem. Het gebruik vanuit applicaties, al dan niet via de ingebouwde zoektaal, wordt behandeld.

Applicatieontwikkeling

Heel wat 'content management' systemen werken in functie van internet activiteiten. Ingebouwde faciliteiten van het product kunnen dienen om content te integreren in webpagina's. De DB2 ContMgr doet meer dan dat. Het product ondersteunt een aantal API's (Application Programming Interfaces) waardoor 'content' kan gebruikt en gemanipuleerd worden vanuit allerlei applicaties, waaronder uiteraard ook de net vermelde webapplicaties.

Om voluit een integratie met andere systemen mogelijk te maken, moet wel een extra component geïnstalleerd worden. Het is de ICM connector, een onderdeel van de WebSphere Information Integrator. Deze component ondersteunt via API's de samenwerking met zowel de Library Server als met de Resource Managers. De DB2 ContMgr zelf heeft API's voor Java en C++.

De API's bieden een aantal services, waaronder:

- data & document modelling: ondersteunt het mappen van een business model met het DB2 ContMgr data model;
- search & retrieve: ondersteunt het zoeken van informatie en documenten (verder in dit artikel meer daarover);
- data import: ondersteunt het binnenhalen van externe gegevens;
- system management: ondersteunt de configuratie en het onderhoud van het systeem;
- document routing: ondersteunt workflow activiteiten (hierover ook meer in het laatste deel van dit artikel).

Het werken met de gegevens zelf doet men via DDO's 'Dynamic Data Objects'. Een item wordt gezien als een DDO, de item attributen zijn zichtbaar als DDO attributen. Voor LOBs (Large Objects) moet men XDO's - 'extended DDO's' - gebruiken. References zijn dan weer gewone attributen die verwijzen naar andere DDO's. Een DDO heeft een aantal methodes die het manipuleren van gegevens mogelijk maken:

- populate(): voor het ophalen van item attributen om ze te tonen als DDO attributen;
- add(), update(), delete(): voor het manipuleren van de items en hun attributen.

Het manipuleren van gegevens wordt iets complexer als het niet meer gaat om items, beheerd door de Library Server in de repository, maar om objecten beheerd door een Resource Manager. Er zijn twee wegen om deze extern beheerde informatie te benaderen:

- *de directe weg*: de applicatie vraagt de Library Server om het URI (Universal Resource Identifier) van het gezochte object; de applicatie, indien geautoriseerd, krijgt dit toegespeeld; voor object manipulaties gebruikt de applicatie nu de API die ondersteund wordt door de betrokken Resource Manager;
- *de indirecte weg*: de applicatie heeft mogelijk geen rechtstreekse toegang tot een Resource Manager; de DB2 ContMgr zorgt voor de communicatie met de betrokken Resource Manager.

Bij applicatieontwikkeling hoort ook automatisch de vraag naar 'transaction management': in hoeverre functioneert de DB2 ContMgr ook als transaction manager?

Zolang het gaat om een applicatie die enkel de Library Server gebruikt, is het een traditioneel transaction management, zowel impliciet als expliciet. Of toch niet helemaal: een wijziging aan gegevens moet 'gecomitted' zijn vooraleer ze zichtbaar is, ook voor de applicatie zelf die de wijziging heeft aangebracht. Dit laatste heeft te maken met de 'gelaagde' werking: de applicatie geeft opdrachten aan de DB2 ContMgr die op zijn beurt opdrachten doorgeeft aan het ondersteunende DBMS.

Als ook één of meerdere Resource Managers in een operatie betrokken zijn, wordt het mogelijk complexer. Een expliciete commit of rollback wordt traditioneel synchroon afgehandeld, maar het loopt anders bij een systeemfout. Een impliciete rollback ten gevolge van een systeemfout zorgt voor complicaties. Afhankelijk van welk systeem of combinatie van systemen (applicatie zelf, de Library Server of een Resource Manager) in de problemen komt, moet men mogelijk wachten op een 'asynchrone recovery' vooraleer alle ressources terug gesynchroniseerd en beschikbaar zijn.

Locking is nauw verbonden met transaction management. De meeste traditionele systemen voorzien in een impliciete locking, afhankelijk van het soort operaties dat gevraagd wordt. De DB2 ContMgr wijkt hier af van de traditionele database systemen. De applicatie

moet vooraleer een aanpassing te doen, expliciet een check-out vragen voor het betrokken item. Dit geeft een persistent write-lock. Een expliciete check-in is nodig om het item weer vrij te geven. Een gewone commit volstaat niet.

Zoektaal

Het zal ons niet verbazen dat we voor het ondervragen van een DB2 ContMgr een SQL-achtige taal kunnen gebruiken. Het is immers gebouwd op een RDBMS. Dat het geen gewone standaard SQL is kunnen we ook verwachten gezien de extra mogelijkheden die de DB2 ContMgr toevoegt aan het RDBMS. De vraagtaal, die het systeem beschikbaar stelt, speelt volledig op dit laatste in.

Het zoeken zelf kan op twee manieren. Deze manieren (technieken) worden in wat volgt concreet even bekeken.

- De ‘parametric search’.

Deze manier van zoeken sluit het best aan bij de klassieke benadering. In het zoeken naar informatie kan men voorwaarden (zoekargumenten) specificeren. Zulk een voorwaarde bestaat uit een attribuut van een item dat het systeem moet vergelijken met één of andere waarde. In deze voorwaarden kan men SQL-functies gebruiken en kan men uiteraard ook voorwaarden combineren via ’and’ en ’or’ (zie voorbeeld). Bij het specificeren van zoekargumenten is men niet beperkt tot de gegevens. Ook systeemattributen kan men refereren in voorwaarden.

Voorbeeld

```
parametric search:

/AUTHOR[@Name = "Taylor" AND NOT (@Country = "UK")]

text search:

/AUTHOR[contains-text-basic(@Biography, "'DB2' + 'België'") = 1]
```

De ’/’ heeft een speciale betekenis. Dit symbool refereert naar de componentenhiërarchie binnen een item. Er zijn meerdere tekens, telkens met een andere betekenis: 'direct dependent', 'parent of the current component', 'reference' ... IMS-kenners vinden hierin zelfs sporen terug van het ondervragen van hiërarchische databases.

- De ’text search’.

Het gebruik van de DB2 ContMgr kan gecombineerd worden met software die text search ondersteunt. Deze software maakt indexen aan op basis van de inhoud van tekstattributen (LOB, Objects, VARCHARs ...). De indexen wijken af van de klassieke indexen. Ze zijn speciaal ontworpen om het zoeken binnen teksten optimaal te laten gebeuren, o.a. via synoniemen, verschillende schrijfwijzen ...

Er is een aparte syntax voor text search (zie voorbeeld).

De queries worden 'embedded' in programma's (of zijn reeds embedded in services). Drie elementen maken deel uit van deze ondervraging:

- er is de query zelf (zie vorige paragrafen);
- er zijn opties die deze query kunnen vergezellen: maximum aantal resultaat items, prepare only ...;
- er is de uitvoering zelf met het opvangen van het resultaat.

Het uitvoeren kan aangepast worden in functie van het verwachte resultaat en de manier waarop men dit wil verwerken. Ofwel gaat men op een vrij klassieke manier te werk. Via het cursor-mechanisme kan de applicatie ittererend item per item beschikbaar krijgen.

Via bijgevoegde opties kan de transmissie (communicatie tussen applicatie en de DB2 ContMgr) gecontroleerd worden.

Voor resultaten die qua volume niet al te groot zijn, kan men ook vragen om het ganse resultaat onmiddellijk 'af te leveren' in de client applicatie.

Besluit

De automatisering van informatieverstrekking en -verwerking gaat steeds verder. De informatie zelf krijgt steeds meer 'exotische' vormen. Om daarmee efficiënt om te gaan, zijn krachtige hulpmiddelen nodig. De DB2 ContMgr is er één van, en zeker niet de minste. Alleen zal het gebruik ervan moeten passen in een algehele bedrijfsstrategie i.v.m. het omgaan met informatie, zowel met interne als met externe, zowel met centraalbeheerde als met gedistribueerde informatie.

In een volgend en laatste artikel rond dit onderwerp behandelen we 'document flow' faciliteiten.

Tabelimplementatie in DB2, Oracle en SQL Server

Eric Everaert (ABIS)

Inleiding

De hoofdbedoeling van een relationele database is natuurlijk data op te slaan als een tabel - relaties - om deze dan vervolgens op een relatief eenduidige manier te kunnen consulteren aan de hand van eenvoudige SQL statements. Een beschikbare tabel en toegekende rechten volstaan om met deze data aan de slag te gaan. De gebruiker schrijft SQL, gegeven de concrete vraagstelling; hij denkt na over WHERE condities. De gebruiker redeneert in rijen en kolommen.

Maar hoe zijn deze data echt fysiek opgeslagen? Een aandachtspunt waar de gebruiker, de applicatieontwikkelaar niet van wakker ligt, want - gelukkig maar - volledig transparant. Echter niet voor de DBA; verschillende fysieke opslagtechnieken hebben immers vaak belangrijke implicaties op het gebied van applicatieperformantie en database onderhoud. Het favoriete werkdomein van elke DBA.

Hier willen we in dit artikel even blijven bij stilstaan: fysieke implementatie van opslagstructuren in een kort overzicht.

De meest eenvoudige situatie

Standaard zullen DB2, Oracle en SQL Server rijen opslaan op een ongeordende, systeemspecifieke wijze, op blokken of pagina's (fysieke eenheden). Men spreekt van 'regular tables', 'heap tables', ... (zie ook het vorige artikel in deze reeks betreffende tablespaces, extent filegroup, ...). Indien geen index aanwezig is, zal elke poging data in deze tabel te benaderen aanleiding geven tot een volledige scan van de fysieke structuur achter de tabel. Deze oplossing is vaak interessant is indien omvangrijke hoeveelheden data moeten worden verwerkt; echter weinig efficiënt indien het aantal daadwerkelijk te consulteren rijen relatief beperkt is. Deze rijen worden voorts ook opgeleverd in een 'willekeurige' volgorde. Indien op basis van SQL statements een volgorde wordt opgelegd, geeft dit aanleiding tot relatief dure sorteeroperaties. Dit alles is natuurlijk 'niet meer' aan de orde als een index wordt aangemaakt op deze tabel.

Omvangrijke tabellen

Grote tabellen vormen vaak een extra uitdaging om mee om te springen. Het beheer, maar vaak ook applicatieperformantie, laten te wensen over. Een oplossing die vaak geboden wordt is het splitsen van deze tabellen in kleinere, meer beheersbare onderdelen. Deze onderdelen worden als relatief zelfstandig en onafhankelijk beschouwd: door de optimizer, die het lezen kan beperken tot betref-

fend onderdeel waarin de gezochte rijen zich bevinden; door de DBA, die beheer en organisatie vaak onafhankelijk kan verrichten van andere onderdelen; maar niet door de gebruiker/ontwikkelaar!

- *DB2 for z/OS*. Reeds lang biedt DB2 de mogelijkheid tabellen aan te maken in gepartitioneerde tablespaces (één tabel per tablespace). Data worden aan partities toegewezen op basis van een speciaal type index: een cluster index. Deze index is verplicht, en heeft enkel tot doel aan te geven welke specifieke waarden (en, bijgevolg rijen) in welke partitie moeten worden opgeslagen (fysiek afzonderlijke datasets). Merk op dat de tablespace gepartitioneerd is, niet de tabel, en dat partitionering een karakteristiek is van de tablespace. Eens toegewezen aan een partitie wordt de data in die partitie op een willekeurige wijze en volgorde opgeslagen.

DB2 for z/OS v8 ondersteunt ook partitionering op het niveau van tabellen. Maar daar weet u nu natuurlijk reeds alles van!

- *Oracle*. Oracle ondersteunt reeds enige tijd partitionering op het niveau van de tabel. Men maakt een onderscheid tussen 5 types van partitionering; de 3 belangrijkste even wat meer in detail:

  - Range partitioning: tabelpartities worden geassocieerd met impliciete kolomwaarden en rijen worden op basis van deze kolomwaarden aan partities toegekend. Een typisch voorbeeld: nummerreeksen, datumwaarden, ... In deze hebben partities dus een logische betekenis!
  - Hash partitioning: tabelpartities worden gebruikt om rijen random te verdelen overheen verschillende partities. Een partitie heeft dus geen logische betekenis! Een typisch voorbeeld: hashing op foreign key.
  - List partitioning: tabelpartities worden geassocieerd met expliciet aangegeven kolomwaarden (lijsten). Een typisch voorbeeld: partitioneren op basis van landen, regio's, ...

De tabel wordt gewoon aangemaakt in een traditionele tablespace; indexen zijn niet verplicht. Verschillende partities - segmenten in de Oracle-woordenschat - worden typisch aangemaakt in verschillende tablespaces, waarschijnlijk aangemaakt op onafhankelijke fysieke devices. Dit laatste is evenwel niet verplicht; maar ten zeerste aan te raden.

- *SQL Server*. Versie 8 van SQL Server - SQL Server 2000 - ondersteunt tabel partitionering op een wat bijzondere manier. De gehanteerde techniek wordt vaak 'view partitionering' genoemd. Inderdaad, de te partitioneren logische entiteit wordt gewoon geïmplementeerd als een reeks onafhankelijke tabellen met evenwel identieke structuur. Een view - aangemaakt als een reeks SELECT-statements gecombineerd op basis van een UNION-instructie - geeft de gebruiker vervolgens de indruk met één logische entiteit te maken te hebben.

Andere technieken

De meeste database-systemen beschikken eveneens over een reeks specifieke technieken die het de DBA moeten toelaten invloed uit te oefenen op de fysieke opslagvolgorde van rijen in een tabel. Vaak gaat het echter om technieken die, als men ze van naderbij bekijkt, toch niet zo origineel zijn!

- *DB2 - Heap-tabellen met cluster index*. Zoals reeds aangehaald bepalen de meeste relationele database-systemen op basis van één of ander algoritme waar nieuwe rijen fysiek zullen worden opgeslagen. Vaak speelt 'pagina capaciteit' (dus vrije ruimte) een rol. Indien op een DB2-tabel echter een cluster index is aangemaakt, zal DB2 trachten de verschillende rijen die logisch bij mekaar horen ook fysiek in mekaars nabijheid op te slaan. Veel hangt natuurlijk af van de ingestelde waarden die de vrije ruimte op de verschillende pagina's ter beschikking van de tabel (tablespace) regelt. Indien DB2 er echter in slaagt om deze rijvolgorde inderdaad af te dwingen, wordt de lees-efficiëntie van bepaalde volgorde-gevoelige queries aanzienlijk verbeterd.

- *Oracle - Clusters*. Traditioneel zorgt Oracle ervoor rijen afkomstig van verschillende tabellen nooit gezamenlijk fysiek op te slaan op dezelfde pagina. Inderdaad, rijen die behoren tot dezelfde tabel behoren tot een segment, dat exclusief aan één tabel toegewezen is. Bij clusters is dit niet meer het geval.

Verschillende tabellen, met een of meerdere gemeenschappelijke kolommen, worden samen in een vooraf gedefinieerde cluster aangemaakt. Gevolg: alle rijen met identieke waarden in de gemeenschappelijke kolommen van de verschillende tabellen worden fysiek samen opgeslagen. Er ontstaat als het ware een minihiërarchische database! Op de gemeenschappelijke kolommen moet een cluster index worden gedefinieerd. Deze structuur is interessant indien bepaalde tabellen vaak (altijd) samen worden gelezen, e.g. orders en orderlijnen, factuur en factuurlijnen, ... Traditionele indexen kunnen worden toegevoegd om de leesefficiëntie van andere queries te verhogen.

- *Oracle - Index Organised Tables (IOT)*. Een IOT is eigenlijk het best te vergelijken met een typische B-tree index, waarbij op de leaf-pages evenwel geen pointers naar de tabeldata is terug te vinden, maar de volledige rij (overflows zijn mogelijk)! Of nog: data en index worden geïntegreerd als één structuur. Bij het aanmaken van de tabel moet worden aangegeven welke kolom uit de tabel de 'index' kolom is, dus de kolom waarop de B-tree structuur zal worden opgezet, wat de volgorde van de rijen zal bepalen. Andere, traditionele indexen, kunnen eveneens worden aangemaakt op een IOT.

- *SQL Server - Geclusterde tabellen*. Deze structuur is vergelijkbaar met de IOT-structuur zoals die ondersteund wordt door Oracle. Ook hier kunnen gerust meer traditionele indexen aan deze structuur worden toegevoegd.

CURSUSPLANNING AUG - DEC 2006

DB2 concepten	375 EUR	op aanvraag
DB2 for OS/390, een totaaloverzicht	1700 EUR	28.08 (L), 18.09 (W), 09.10 (L), 30.10 (W), 27.11 (L)
DB2 UDB, een totaaloverzicht	1625 EUR	18.09 (W), 09.10 (L), 30.10 (W)
RDBMS concepten	325 EUR	28.08 (L), 18.09 (W), 09.10 (L), 30.10 (W), 27.11 (L)
Basiskennis SQL	325 EUR	29.08 (L), 19.09 (W), 10.10 (L), 31.10 (W), 28.11 (L)
DB2 for OS/390 basiscursus	1050 EUR	30.08 (L), 20.09 (W), 11.10 (L), 01.11 (W), 29.11 (L)
DB2 UDB basiscursus	975 EUR	20.09 (W), 11.10 (L), 08.11 (W)
SQL workshop	700 EUR	07.08 (L), 17.08 (W), 21.09 (L), 16.11 (W), 04.12 (L)
Extended SQL in DB2	400 EUR	25.09 (L), 27.11 (W)
Gebruik van DB2 procedural extensions	400 EUR	20.10 (W), 08.12 (L)
DB2 for OS/390 programmering voor gevorderden	750 EUR	23.10 (L), 18.12 (W)
DB2 for OS/390: SQL performance	1200 EUR	25.10 (L), 11.12 (W)
XML in DB2	400 EUR	14.09 (W), 11.12 (L)
DB2 for OS/390 database administratie	1600 EUR	16.10 (W), 04.12 (L)
DB2 for z/OS in een Java omgeving	400 EUR	15.12 (L), 20.12 (W)
DB2 for OS/390 operations and recovery	1500 EUR	18.08 (W)
DB2 for z/OS systems performance and tuning	1000 EUR	12.10 (W)
DB2 UDB DBA 1 - Kernvaardigheden	1700 EUR	16.10 (L), 27.11 (W)
DB2 UDB DBA 2 - Configure & Tune	1275 EUR	27.09 (W)

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245691
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Pelmolenlaan 1-K
NL-3440 AC Woerden
Tel. 0348-435570
Fax 0348-432493
training@abis.be