ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 5 - NUMMER 3, JUNI 2008

OPEN CURSOR

Bij het doornemen van de "highlights" van versie 9 van DB2 voor z/OS valt het op hoeveel aandacht IBM besteedt aan o.a. de aspecten kostenbesparing en gebruiksgemak. Ook al bij versie 8, trouwens. Wat die kostenbesparing betreft: die slaat op betere performance, o.a. wat betreft het manipuleren van data van variabele lengte (één van de redenen waarom VARCHAR maar ook ***datacompressie*** *steeds interessanter wordt). Maar is de toegenomen complexiteit van DB2 niet in tegenspraak met het geclaimde gebruiksgemak en dus de (tijds-) kostenbesparing wat betreft (data- en applicatie-) beheer? Verstandig gebruik van nieuwe mogelijkheden zoals "clone tables" en de nieuwe SWITCH-optie van REBIND (zie het hoofdartikel) kan zowel de performance als de kostenbesparing ten goede komen.*

Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- Deze maand een derde onderwerp in de reeks DB2 9, over het vermijden van "regressie" bij rebinds: *Help, hoe herstel ik mijn oud toegangspad?*
- Compressie is dan weer één van de "oude" mogelijkheden van DB2 voor z/OS, die een revival lijkt te kennen, o.a. doordat DB2 9 voor LUW nu ook compressie ondersteunt. Achtergrondinformatie vindt u in *Datacompressie in DB2: wat en hoe?*
- Ten slotte een eerste bijdrage in een reeks van drie, waarin we de toegevoegde waarde nagaan door het samengaan van IBM en Cognos, in Data Warehouses: *Cognos, an IBM Company (deel 1)*.

3

CLOSE CURSOR

In de reeks DB2 9 vertellen we u volgende keer hoe u "clone tables" kunt gebruiken. We nemen ook terug de draad op wat betreft XML: xslt, xquery, schema's... in een breder (relationeel en hiërarchisch) kader.

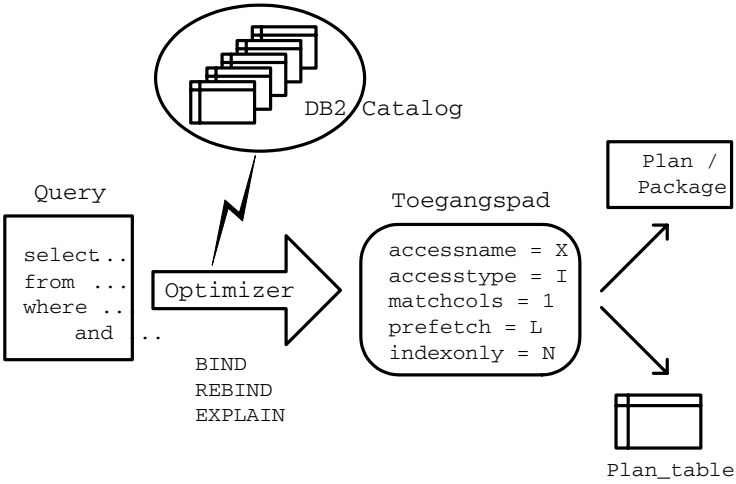
Tot dan!

Help, hoe herstel ik mijn oud toegangspad?

Steven Scheldeman (ABIS)

DB2 levert ons een zeer goed werkende optimizer die voor een gegeven SQL-query in meer dan 99% van de gevallen het meest efficiënte toegangspad (access path) kiest. Dat is natuurlijk een zeer goede score, maar helaas is het die fractie van een procent die aardig wat kopzorgen kan opleveren. Wanneer we uitsluitend gebruik maken van dynamische SQL, is dat probleem te overzien, zelfs ingecalculeerd. Bij statische SQL, bij batch verwerkingen, kan dit probleem ons aardig parten spelen.
Initieel niet zozeer: we testen onze programma's en hun embedded SQL – we passen ze aan waar nodig – tot we het gewenste resultaat krijgen. Naderhand kan het probleem zich wel stellen: iedere REBIND van een bestaand plan – om het even wat de aanleiding van de REBIND ook moge zijn – loopt het risico dat het nieuwe toegangspad minder performant is dan het oude.
Dit is natuurlijk te wijten aan het feit dat wij de optimizer niet echt onder controle hebben, gelukkig maar.

Overzichtsfiguur



Een voor de hand liggend antwoord op dit probleem luidt als volgt: na een REBIND (automatisch of niet) die leidt tot een minder performante query-implementatie, laten we teruggrijpen naar het vorige, betere toegangspad. Maar... is dat nog wel beschikbaar?

Terugkeren naar het vorige toegangspad?

Tot vóór versie 9.1 van DB2 stonden ons twee manieren ter beschikking om dit te bewerkstelligen.

De eerste manier is vrij voor de hand liggend: gebruik BIND met een nieuwe plan- of package-naam (of een nieuwe package-versie). Blijkt het nieuwe toegangspad even performant of performanter dan het oude, dan vervangen we (m.b.v. REBIND) het oude plan/package door het nieuwe. Zoniet, dan behouden we het oude plan/package.

De tweede manier maakt gebruik van OPTHINT (optimization hints). Hiervoor moeten we wel gebruik maken van een uitgebreide, post-v5 PLAN_TABLE (met o.a. de kolommen OPTHINT en HINT_USED) en moet EXPLAIN gebruikt zijn bij de originele BIND. Deze optie werd uitgebreid besproken in een vorig nummer van Exploring DB2.

Meer over OPTHINT: zie *Jaargang 1 nr 1, september 2002.*

Sinds versie 9 is er een derde manier, die gebruik maakt van een aantal nieuwe opties voor REBIND PACKAGE. Inderdaad, enkel voor packages: versie 9 is duidelijk bezig de ondersteuning voor plans af te bouwen. In de praktijk bevat een plan tegenwoordig trouwens meestal geen SQL meer. De geruchten gaan dat “Vnext” geen PLANs meer zal hebben! Zoals DB2 voor LUW, dus.

Nieuw in versie 9: “plan management”

De bedoelde nieuwe opties voor REBIND PACKAGE zijn *PLANMGMT* en *SWITCH*. Nauw daarmee verbonden is een nieuwe FREE PACKAGE optie, nl. *PLANMGMTSCOPE*.

- **PLANMGMT**: deze nieuwe optie heeft drie settings: OFF, BASIC en EXTENDED.
  - PLANMGMT(OFF) verandert niets aan het gekende gedrag van DB2. Een REBIND PACKAGE zal het oude toegangspad overschrijven; elk package heeft dus slechts één actieve kopie.
  - PLANMGMT(BASIC) bewaart het oude toegangspad als een oude kopie (PREVIOUS) en schrijft een nieuw toegangspad als actieve kopie. Hou er rekening mee dat deze twee kopieën wel exact gelijk kunnen zijn.
  - PLANMGMT(EXTENDED) bewaart twee oude kopieën, naast de actieve kopie. ORIGINAL verwijst naar het toegangspad van de originele BIND, en deze blijft altijd bewaard. “Previous” verwijst – net als in de BASIC setting – naar het toegangspad dat actief was vlak voor REBIND PACKAGE.
- **SWITCH**: deze optie van REBIND PACKAGE stelt ons in staat om een package terug te verbinden met een oudere kopie, waardoor we dus effectief de vorige REBIND teniet doen. SWITCH heeft twee settings - PREVIOUS en ORIGINAL
  - SWITCH(PREVIOUS) activeert de voorgaande kopie en dus het voorgaande toegangspad. Dit is bruikbaar bij zowel PLANMGMT(BASIC) als PLANMGMT(EXTENDED).
  - SWITCH(ORIGINAL) activeert de oorspronkelijke kopie en dus het oorspronkelijke toegangspad. Dit is alleen bruikbaar bij PLANMGMT(EXTENDED).

In beide gevallen wordt het actieve (te vervangen) toegangspad als PREVIOUS opgeslagen.

- **PLANMGMTSCOPE**: deze nieuwe optie van FREE PACKAGE geeft ons de kans om niet langer gebruikte kopieën te verwijderen. Er zijn twee settings: ALL en INACTIVE.
  - PLANMGMTSCOPE(ALL) verwijdert het hele package, met inbegrip van alle kopieën. Dit is de default.
  - PLANMGMTSCOPE(INACTIVE) verwijdert alleen de oude kopieen. Het package zelf, met zijn huidige actieve toegangspad, blijft bestaan. Nuttig dus om wat ruimte vrij te maken in de EDM-pool.

Hierbij moeten we een aantal bemerkingen maken:

- Het is meteen duidelijk dat het gebruik van PLANMGMT(BASIC) de grootte van de "skeleton package table" (tablespace DSNDB01.SPT01) kan verdubbelen, en PLANMGMT(EXTENDED) dit zelfs kan verdrievoudigen. Bijkomend moeten we ook vermelden dat er een toename van 10 tot 14% CPU overhead is bij het uitvoeren van de REBIND PACKAGE instructie. De EDM pool daarentegen wordt niet groter, vermits alleen de actieve kopie van een package rechtstreeks toegankelijk moet zijn.
- Er is ook een nieuwe systeemparameter *PLANMGMT* die de default waarden van de REBIND PACKAGE optie PLANMGMT instelt. Mogelijke waarden zijn OFF, BASIC en EXTENDED. De default-waarde bij installatie van V 9.1 is OFF.
- Houd er ook rekening mee dat bij de eerste REBIND PACKAGE, en wanneer PLANMGMT(EXTENDED) actief is, het oude toegangspad zowel onder PREVIOUS als onder ORIGINAL zal opgeslagen worden.
- Deze kopieën van een package zijn geen versies van een package. Bij "*versioning*" gaat het meestal om veranderde SQL, of om applicaties die naast elkaar moeten uitgevoerd kunnen worden. Het gebruik van PLANMGMT richt zich naar veranderde toegangspaden maar met dezelfde SQL. Bovendien kunnen verschillende versies elk dus 2 of 3 kopieën hebben, onafhankelijk van elkaar.
  Denk bij package-kopieën eerder aan het concept van CLONE TABLE: twee kopieën van dezelfde tabel, maar met verschillende inhoud, waarvan er slechts één tezelfdertijd actief kan zijn. Meer over clone-tabellen vindt u in ons volgend nummer.

Het voornaamste doel van deze nieuwe opties is op een eenvoudige manier terug te kunnen vallen op het voorgaande toegangspad na een rebind van een package. De vrees dat na een rebind ten gevolge van b.v. een DB2-upgrade of een RUNSTATS sommige packages significant minder performante toegangspaden zouden krijgen, kan hiermee voor een groot deel weggenomen worden. Indien men niet tevreden is over het toegangspad van de rebind, heeft men nu de mogelijkheid om (op korte termijn) terug te keren naar het oude. Zolang we het natuurlijk hebben over STATIC SQL.

Toegangspaden vergelijken

De vraag naar het herstellen van het “oude” toegangspad wordt uiteraard voorafgegaan door de vraag om verschillende toegangspaden met elkaar te vergelijken. Het antwoord daarop is dubbel. Er is geen 'ingebouwde' manier om twee access paths met elkaar te vergelijken. Uiteindelijk is het enkel het huidige, uitvoerbare, toegangspad dat toegankelijk is en dat dus gedocumenteerd is in de catalog tables.

Anderzijds kunnen we altijd met een manuele *compare explain* twee toegangspaden vergelijken. Visual Explain (en zijn opvolgers, het Optimization Service Center en de Optimization Expert) maakt het interpreteren van een toegangspad en de bijhorende kostprijs trouwens zeer gebruiksvriendelijk. Een kort overzicht van Visual Explain vindt u in een vorig nummer van Exploring DB2.

Meer over Visual Explain, zie: *Jaargang 4 nr 5, december 2006*.

Deze benadering vergt wel een pro-actieve instelling. Eerst en vooral dienen we elke (RE)BIND PACKAGE te draaien met de optie EXPLAIN(YES). Op die manier kunnen we later de rijen in onze PLAN_TABLE raadplegen die met de huidige en/of de vorige kopie van het toegangspad overeenkomen. In de catalog-tabel SYSIBM.SYSPACKAGE vinden we namelijk in de kolom BINDTIME de timestamp terug waarop het toegangspad aan het package gebonden werd. Deze waarde komt exact overeen met de waarde in de kolom BIND_TIME van PLAN_TABLE. Ten minste, als de kolom BIND_TIME aanwezig was op het ogenblik van Explain, d.w.z. als er niet met een tabelversie van DB2v5 gewerkt wordt...

Samen met de naam van het package en de collectie waartoe het behoort, kunnen we in de PLAN_TABLE de rijen terugvinden (query op COLLID, PROGNAME, VERSION, BIND_TIME) die met de *actieve kopie* van het package overeenkomen:

```
SELECT p.*
FROM   plan_table p INNER JOIN sysibm.syspackage g
        ON p.progname = g.name   AND p.collid = g.collid  AND
           p.version = g.version AND p.bind_time = g.bindtime
WHERE  p.progname = :package-name
ORDER BY p.queryno, p.qblockno, p.planno, p.mixopseq
```

Voor de *previous* of *original* moeten we jammer genoeg de BINDTIME met de hand bijhouden (of “even” een REBIND SWITCH uitvoeren): de catalog laat alleen de timestamp van de huidige kopie zien...

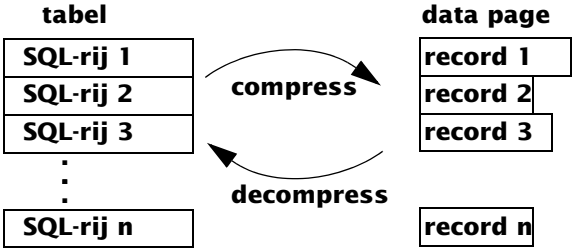
Doen we dit consequent (altijd EXPLAIN gebruiken bij (RE)BIND PACKAGE én na iedere (RE)BIND PACKAGE de timestamp noteren) dan kunnen we uit de PLAN_TABLE alle informatie halen die nodig is om het verschil tussen een nieuw en een oud toegangspad te analyseren.

Datacompressie in DB2: wat en hoe?

Peter Vanroose (ABIS)

Sinds 1994 laat DB2 for z/OS (toen nog “DB2 for MVS v3”) toe om een tablespace als *“compressed”* te configureren. De datapages en records (zowel in het VSAM-bestand als in de bufferpool en de logs) bevatten dan een gewijzigde, met name een gecomprimeerde, versie van de data-records. De datamanager moet deze data dus *decomprimeren* bij elke leesinstructie, en *comprimeren* bij elke schrijf-instructie. Dit gebeurt weliswaar via een hardware-implementatie van het compressie- en decompressie-algoritme.

Sinds versie 9 is ook in DB2 for LUW een gelijkaardige compressiemogelijkheid voorzien, zij het puur softwarematig; de uitleg in deze tekst is dus van toepassing op alle (recente) versies van DB2.



Verliesloze compressie

Uiteraard mag er bij het comprimeren geen informatie verloren gaan. De compressie- en decompressie-functies moeten elkaars effect dus exact ongedaan maken, in beide richtingen. Hoe is dit mogelijk als terzelfdertijd de gecomprimeerde data toch minder bytes bevat dan de originele data?

Het principe van verliesloos comprimeren (“lossless compression”) is relatief eenvoudig, en gebaseerd op statistische eigenschappen van de gegevens: vervang bytesequenties die dikwijls voorkomen door een kortere bytesequentie (ook wel *codewoord* genoemd), en weinig voorkomende byte-sequenties door een langer codewoord. Sommige stukjes data worden hierdoor effectief gecomprimeerd (want voorgesteld met minder bytes) terwijl onvermijdelijk sommige andere stukjes data zullen “expanderen”. Dit laatste is nodig om ondubbelzinnige decompressie te behouden, zoals uit het voorbeeld hieronder zal blijken.

De “vertaling” van een niet-gecomprimeerde byte-sequentie naar een gecomprimeerde en omgekeerd gebeurt door middel van een *codeboek* (“dictionary”), dit is een lijst van overeenkomstige paren byte-sequenties (niet-gecomprimeerd versus gecomprimeerd).

Een goede keuze van het codeboek kan ervoor zorgen dat *gemiddeld* de lengte van de gecomprimeerde tegenhanger korter is dan die van zijn ongecomprimeerde partner, ook al zullen er dus paren zijn waarbij het “codewoord” langer is dan de “*plaintext*”.

Zie het kader voor een eenvoudig voorbeeld, waar voor de eenvoud van de uitleg is gekozen voor een binair alfabet (0/1); een byte-gebaseerd (dus 256-tallig) alfabet werkt op een analoge manier, zij het uiteraard met een veel langer codeboek.

Twee voorbeelden

codeboek (1)	
origineel	*codewoord*
000	11111
001	11110
010	11101
100	11100
011	110
101	101
110	100
111	0

codeboek (2)	
origineel	*codewoord*
00	000
01	001
10	010
110	100
1110	011
11110	101
111110	110
111111	111

Voorbeeld van compressie met deze twee codeboeken:

origineel: 10111100111011011111111110

gecomprimeerd met codeboek (1): 10101111010001100 0100

gecomprimeerd met codeboek (2): 010101001100011111101

Om een bepaald codeboek te gebruiken, moet de oorspronkelijke data eerst opgedeeld worden in fragmentjes die allemaal in de linker kolom van het codeboek voorkomen. Een codeboek moet dus gegarandeerd *volledig* zijn, d.w.z. dat een willekeurige byte-sequentie moet kunnen opgedeeld worden in dergelijke stukjes. Dit is het geval voor het eerste voorbeeld vermits in de eerste kolom van (1) alle lengte-drie-combinaties voorkomen. In het algemeen hoeven deze fragmenten niet even lang te zijn, zoals blijkt uit voorbeeld (2).

Omgekeerd moet de rechterkolom gegarandeerd *prefix-vrij* zijn, d.w.z. dat de gecomprimeerde byte-sequentie op zijn beurt op een *eenduidige manier* moet kunnen opgesplitst worden in codewoorden. Dit is zo in beide voorbeelden.

Voor de twee voorbeelden: de oorspronkelijke data heeft lengte 27, de gecomprimeerde data heeft slechts lengte 21, dus een compressie van 6/27 of 22%. Is dit toevallig zo? Of beter gezegd: welke compressie-factor mogen we verwachten bij gebruik van codeboek (1)? Als de data veel meer enen bevat dan nullen, zal de gemiddelde lengte van een gecodeerd record korter zijn dan de niet-gecomprimeerde, oorspronkelijke lengte, al wordt de lengte van b.v. “000” langer na compressie! Als de data 25% nullen bevat en dus 75% enen, dan zal de “opgedeelde” originele tekst ongeveer 14% fragmentjes “011” hebben, maar slechts 1,5% fragmentjes 000. Zoals eenvoudig rekenwerk laat zien, zal een fragmentje met codeboek (1) gecomprimeerd worden van lengte 3 naar “gemiddeld” lengte 2,47, een compressie van bijna 18% dus. Op een gelijkaardige manier kan nagerekend worden dat met codeboek (2) de compressie iets meer dan 15% bedraagt.

Details van deze berekening: zie *Bijlage* op p. 13.

Data-statistieken

De allereerste stap bij het opstellen van een “nuttig” codeboek bestaat dus uit een statistische analyse van de gegevens die gecomprimeerd moeten worden. Vóór DB2 v3 kon data-compressie reeds gebruikt worden, nl. door gebruik te maken van de EDITPROC (zonder hardware-versnelling). Het meest gebruikte compressie-algoritme was toen het *Huffman-algoritme* dat op een heel eenvoudige, snelle manier een optimaal codeboek opstelt wanneer enkel de “single-byte”-statistieken van de data gekend zijn. Dit zijn dus 256 percentages met als som 100%. Het voorbeeld met codeboek (1) in het kader is de binaire Huffman-code van lengte 3 voor p(0)=25%.

De “ingebouwde” compressie in DB2 gebruikt een ander algoritme, met name dat van *Abraham Lempel en Jacob Ziv*. Voor het LZ-algoritme (oorspronkelijk uit 1978, zes jaar later gevoelig verbeterd door *Terry Welch* tot het LZW-algoritme) is de linkerkolom van het codeboek van variabele lengte, terwijl de rechterkolom van vaste lengte is, zoals in het voorbeeld met codeboek (2). Hierdoor is decompressie iets sneller dan compressie. Het belangrijkste verschil met Huffman is echter dat de gebruikte data-statistieken gedetailleerder zijn: er wordt ook rekening gehouden met statistische afhankelijkheden tussen opeenvolgende bytes. Zo is b.v. in “gewone” tekst de waarschijnlijkheid van een “u” na een “q” groter dan na een “aa”. In het algemeen blijft uiteraard gelden dat veel voorkomende tekstsequenties (b.v. veel voorkomende postcodes in een tabel met adresgegevens) een korter codewoord krijgen toebedeeld dan uitzonderlijke sequenties. De datacompressie is beduidend beter dan voor een vergelijkbaar Huffman-codeboek: typisch in de ordegrootte van 50% voor “gewone” tekst, maar 80% is niet uitzonderlijk. Anderzijds zal data met weinig redundantie uiteraard bijna niet gecomprimeerd worden.

Eén codeboek per partitie

Vermits het codeboek zelf ook moet opgeslagen worden, en de grootte van dit codeboek niet afhangt van de omvang van de data, wordt compressie pas interessant wanneer hetzelfde codeboek gebruikt kan worden voor een relatief grote hoeveelheid “gelijkaardige” data, d.w.z. met ongeveer dezelfde statistische eigenschappen.

DB2 slaat één codeboek op per (partitie van een) tablespace, in de eerste 64K van die partitie. De data per tablespace(partitie) moet dus voldoende “gelijkaardig” zijn, d.w.z. dat de totale statistieken gebruikt voor het creëren van het codeboek voldoende representatief moeten zijn voor alle data in die partitie. In het bijzonder zal de mate van compressie (de *compressie-factor*) erbij geholpen zijn als een nieuw codeboek gegenereerd wordt wanneer de statistische eigenschappen van de data sterk wijzigen. Enkel het REORG- en het LOAD-utility laten toe om een nieuw codeboek de genereren gebaseerd op de in te laden data. Dit is trouwens de default (tenzij bij LOAD RESUME YES). Gebruik de optie KEEPDICTIONARY om dit niet te doen, dus om een bestaand codeboek te behouden. Op die manier wordt behoorlijk bespaard op CPU-consumptie, vermits het opstellen van het codeboek zeer tijdsintensief is

Dit geldt zowel voor DB2 for z/OS als voor DB2 for LUW (waar compressie wordt geactiveerd op tabelniveau met de optie COMPRESS YES van CREATE/ALTER TABLE i.p.v. diezelfde optie van CREATE/ALTER TABLESPACE op z/OS).

RUNSTATS vertelt achteraf ook wat de compressiefactor was: de kolom PAGESAVE van SYSIBM.SYSTABLEPART geeft het percentage gewonnen pages dankzij compressie, en de kolom PCTROWCOMP van SYSIBM.SYSTABLES geeft aan hoeveel procent van de tabelrijen effectief gecomprimeerd werden. DB2 behoudt namelijk de niet-gecomprimeerde data in een record indien compressie tot meer bytes zou leiden.

Een win-win-situatie?

Data-compressie kan gezien worden als een ruiloperatie: minder schijfruimte voor meer CPU-verbruik. Dit is echter niet noodzakelijk zo: niet alleen gebeurt de compressie/decompressie (althans op een IBM-mainframe) op specifieke hardware (de “compressie-chip”), maar bovendien is het dikwijls zelfs zo dat de totale kost van een SQL-query een stuk kleiner is op een gecomprimeerde tablespace dan als diezelfde data niet gecomprimeerd zou zijn: er passen namelijk meer gecomprimeerde rijen in een page, zodat het aantal te verwerken pages voor de query (en de I/O en CPU die hiermee gepaard gaan) een stuk lager kan zijn!

In dat geval is compressie gewoon een must. Uiteraard is het duidelijk dat voor tabellen met zeer korte rijen (met name 16 bytes of minder) compressie niet helpt, omdat er nooit meer dan 255 rijen in een page passen; anderzijds helpt het evenmin om een rij van 4000 bytes in een 4K-page te comprimeren naar b.v. 2200 bytes. In dat laatste geval kan het natuurlijk wel helpen om de page-size op te trekken naar b.v. 16K...

Samenvattend kan men stellen dat het bijna altijd een goed idee is om de COMPRESS-optie van een TABLESPACE aan te zetten, tenzij misschien voor (1) zeer smalle tabellen, (2) zeer kleine tabellen, of (3) data die bijna geen redundantie bevat. COMPRESS zou dus eerder de regel dan de uitzondering moeten zijn. Hopelijk geeft deze tekst u er een eerste, intuïtief gevoel voor welke tabellen tot die uitzonderingen zouden kunnen behoren.

Cognos, an IBM Company (deel 1)

Kris Van Thillo (ABIS)

Er zijn een aantal karakteristieken die elke Data Warehouse (DW) en/of Business Intelligence (BI) cursus gemeenschappelijk hebben: in een inleidend hoofdstuk valt de naam Bill Inmon; en een lijst met eigenschappen van een DW wordt aangehaald. Steevast komt dan ook de vraag aan bod: "Wat is nu de belangrijkste karakteristiek van een DW? Wat maak nu echt het verschil met een traditionele, niet-DW omgeving?"

Een antwoord op deze vraag geven is niet evident; het is afhankelijk van éénieders specifieke kijk op een DW: als database specialist, data architect, gebruiker of ontwikkelaar houden we er vaak een andere mening over na. Vaak komt men echter snel tot een gemeenschappelijk standpunt: laagdrempeligheid, en alles wat noodzakelijk is om deze te realiseren – daar gaat het om!

Of waarom Cognos 'an IBM Company' werd.

Time to market!

Want gaat het daar vandaag niet om? In een steeds competitievere en sneller evoluerende mondiale markt zo kort mogelijk op de bal spelen; tendenzen herkennen; acties evalueren, succes ervan inschatten en tenslotte implementeren.

De gevolgen hiervan zijn aanzienlijk – en voor IT samen te vatten onder het kopje *laagdrempeligheid*!

- Gebruikers – analisten, rapportontwikkelaars, eindgebruikers, managers – moeten in staat worden gesteld op een zeer eenvoudige, intuïtieve en visuele wijze met data, rapporten, en rapportontwikkeling om te gaan. Self-service is key! Weg dus met lange ontwikkeltijden voor statische rapporten die zelden of nooit meer voldoen aan specifieke eisen van 'de klant' op moment van oplevering. 'Informatie-eilandjes' zijn uit den boze: toegevoegde waarde bevindt zich daar waar bruggen kunnen worden geslagen (m.b.t. inhoud, tools, definities, ...). Enkel op deze wijze kan een bedrijfsbreed 'zicht' worden ontwikkeld, dat de bron is van analyses en rapportages.
- 'Ontwikkeling' in handen geven van de gebruikers stelt IT voor ongekende uitdagingen, bijvoorbeeld:
  - Mooi uitgekristalliseerde database-modellen, het resultaat van jarenlange ervaring met specifieke databases worden in vraag gesteld, want te complex en ondoorzichtig voor de gebruiker.
  - Naast traditionele mogelijk relationele database-bronnen moeten plots ook vreemde bronnen geconsulteerd worden: files, multidimensionele structuren, spreadsheets, internet, ...
  - Performance, beveiliging, monitoring... krijgen in deze context een totaal andere, dwingendere betekenis. Bijvoorbeeld, om-

dat het dynamische en onvoorspelbare karakter van de gebruikersvraag zal toennemen: van veel gebruikers met voorspelbare acties over weinig gebruikers met onvoorspelbare acties naar veel gebruikers met onvoorspelbare acties: een hele uitdaging!

- Teneinde het combineren en integreren van data mogelijk te maken heeft de gebruiker behoefte aan een bibliotheek met definities. Een overzicht van de betekenis van de gegevens (en hun context!) die in rapporten kunnen worden opgenomen: de metadatabase, warehouse metastore, ... – vele namen doen de ronde. Ook de data-architect lijkt 'herboren'.

IBM

IBM heeft het belang van deze verschuiving steeds erkend. De focus lag hierbij voornamelijk op data: DB2 van de nodige features voorzien om als een volwaardig en efficiënt DW te kunnen fungeren. Of naar (DB2-externe) tools toe invulling geven aan het Extractie, Transformatie, Load (ETL) proces. Misschien wat kort door de bocht samen te vatten als 'technisch enablement' van de dataïnfrastructuur: de databron staat centraal. Met slechts beperkte aandacht voor de insteek laagdrempeligheid zoals hierboven werd uiteengezet.

Cognos

De *IBM Cognos 8 BI* omgeving richt zijn aandacht duidelijk op het ter beschikking stellen van een globale DW-architectuur en -infrastructuur die het alle gebruikers moet mogelijk maken op een laagdrempelige wijze met data en informatie om te gaan.

- openheid staat centraal: naar databronnen toe, naar integratie met bestaande al dan niet DW-applicaties toe, naar tools van derden toe, naar ondersteunende systemen toe voor beveiliging, monitoring, ... ;
- implementatie op basis van een open SOA-architectuur: webservices, XML, ... zorgen voor uitbreidbaarheid en integratiemogelijkheden. De te hanteren API is bekend (WSDL beschikbaar) en SDKs voor .Net en Java EE zijn beschikbaar. Men kan gebruik maken van bestaande web- en applicatieservers;
- gebruiksvriendelijkheid: rapporten, scorecards, dashboards zijn beschikbaar, of kunnen door gebruikers worden ontwikkeld; informatie kan via een veelheid aan media ter beschikking worden gesteld (Windows Mobile, Blackberry, Web, Portals, Office, PDF, ...);
- beheersbaarheid: een management-infrastructuur is beschikbaar (logging, auditing, event management, ...)

In twee vervolgartikels pogen we bovenstaande topics wat meer in detail uiteen te zetten. Volgende onderwerpen komen dan aan bod:

- de *IBM Cognos 8 BI architectuur*: met name staan we stil bij de opzet en de organisatie van dit platform (presentatie/web-laag, applicatielaag, en datalaag)
- de *IBM Cognos 8 BI data-tier* in meer detail besproken, waarbij de link met een veelheid aan databronnen aan de orde is..

CURSUSPLANNING JULI-OKTOBER 2008

DB2-concepten		op aanvraag
DB2 for z/OS, een totaaloverzicht	1900 EUR	25-29/08 (L), 13-17/10 (W), 20-24/10 (L)
DB2 UDB for LUW, totaaloverzicht	1825 EUR	25-26/08 en 15-17/09 (L), 13-17/10 (W)
RDBMS-concepten	350 EUR	25/08 (L), 13/10 (W), 20/10 (L)
Basiskennis SQL	350 EUR	25/08 (L), 13/10 (W), 20/10 (L)
DB2 for z/OS basiscursus	1200 EUR	27-29/08 (L), 15-17/10 (W), 22-24/10 (L)
DB2 UDB for LUW basiscursus	1125 EUR	15-17/09 (L), 15-17/10 (W)
SQL-QMF voor eindgebruikers	1200 EUR	op aanvraag
SQL workshop	750 EUR	08-09/09 (L), 23-24/10 (W)
SQL voor gevorderden	425 EUR	27/10 (W)
DB2 procedural extentions	425 EUR	op aanvraag
DB2 for z/OS programmeren voor gevorderden	800 EUR	op aanvraag
DB2 for z/OS: SQL performance	1275 EUR	29/09-01/10 (L)
XML in DB2	425 EUR	17/10 (L)
DB2 for z/OS database administratie	1700 EUR	20-23/10 (W)
DB2 for z/OS installation & migration	1050 EUR	22-23/09 (London)
DB2 for z/OS operations & recovery	1500 EUR	24-26/09 (London)
DB2 for z/OS system performance and tuning	1050 EUR	27-28/10 (London)
DB2 LUW DBA 1: Kernvaardigheden	1700 EUR	22-25/09 (L)
DB2 v8 upgrade		op aanvraag
DB2 v9 upgrade		op aanvraag

Plaats: L = Leuven, W = Woerden;
voor details en andere cursussen, zie `http://www.abis.be/`

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245639
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

Berekening van de gemiddelde lengte van een codewoord

Wat is de compressie-factor van het codeboek van voorbeeld (1)?

Om dit te bepalen, moeten we enerzijds weten hoe dikwijls een bepaald stukje "plaintext" (element uit de linker-kolom) voorkomt in een "gemiddelde" tekst, en anderzijds hoe lang het bijhorende codewoord (d.w.z., gecomprimeerde versie ervan) is.

De sequentie `000` komt $(0,25)^3$=0,015625 keer voor, vermits een apart teken met 0,25 kans een 0 is. De gecomprimeerde lengte 5 (lengte van het bijhorende codewoord) zal dus voor ongeveer 1,5% bijdragen in de gemiddelde lengte van een stukje plaintext van lengte 3. Die gemiddelde lengte is dus

$5*((0,25)^3+3*0,75*(0,25)^2) + 3*(3*0,25*(0,75)^2) + 1*(0,75)^3 = 2,46875$

vermits de fragmenten `001`, `010` en `100` alle drie een relatieve frequentie van $0,75*(0,25)^2$ hebben, elk met een codewoord van lengte 5. Analoog voor de drie fragmenten `101`, `011` en `110` en voor `111`.

De compressie is bijgevolg 2,46875/3 = 0,8229167, of een reductie met 17,71%.

Voor de compressie-factor van voorbeeld (2) met dezelfde tekst-statistieken (dus 25% nullen en 75% enen) gaan we gelijkaardig te werk: wat zijn de relatieve frequenties van de 8 "plaintext" fragmenten: `00`, `01`, `10`, `110`, `1110`, `11110`, `111110` en `111111`?

De relatieve frequentie van b.v. `01` is de kans dat er "ergens" in de plaintext een 0 staat, gevolgd door een 1. Deze kans is uiteraard `0,75*0,25`. De andere relatieve frequenties worden analoog bepaald. Verifieer dat de som van deze 8 getallen wel degelijk 1 is!

Zoals voor codeboek (1) kan voor codeboek (2) de gemiddelde lengte bepaald worden, deze keer echter van een "plaintext fragment":

$2*((0,25)^2+2*0,75*0,25) + 3*0,25*(0,75)^2 + 4*0,25*(0,75)^3 + 5*0,25*(0,75)^4 + 6*(0,25*(0,75)^5+(0,75)^6) = 3,538085937625.$

De "plaintext"-lengte is nu uiteraard langer dan 3, vermits de "codetext"-lengte van elk fragment altijd 3 is. De compressie-factor is hier dus 3/3,538085937625=0,84791, een winst van ongeveer 15,21%.

Optimaliteit

De twee voorbeelden (1) en (2) zijn beide optimale codes met codeboeklengte 8, voor een binaire tekst waarbij de frequentie van enen gelijk is aan 75%, en waarbij respectievelijk vast-naar-variabele en variabele-naar-vaste codering wordt gebruikt. Er is een eenvoudig algoritme voor het opstellen van het codeboek voor zowel (1) als (2): dit zijn respectievelijk het *Huffman*-algoritme en het *Tunstall*-algoritme.

Grotere codelengtes dan 3 leveren beter compressie. De limietwaarde voor de compressiefactor is de *entropie* van de basistekst; voor het voorbeeld: h(0,25) = 0,81128, dus compressiewinst 18,89%.