ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 8 - NUMMER 2, APRIL 2013

OPEN CURSOR

Deze maal hebben we voor u een volledig "DB2 10 for z/OS" nummer ineen gestoken. Nu u goed en wel op dreef bent met deze laatste versie van DB2, is het wellicht tijd om enkele van de nieuwe mogelijkheden uit de proberen.
Wij deden dat alvast voor u, en exploreerden een drietal opvallende nieuwigheden.
Wat dacht u van een efficiënter access-path dan matching index access? Hashing! En wat is de impact op het systeem van zogenaamde "online" DDL-wijzigingen? En om af te ronden nog een kleine smaakmaker rond host-variabelen...

Veel leesgenot met dit nummer!

Het ABIS DB2-team.

IN DIT NUMMER:

- Met "*Hash access voor user data in DB2 10*" laten we u kennismaken met een (voor DB2) nieuwe, snelle manier van single-row-fetch. Wellicht is minstens één van uw tabellen de ideale kandidaat om in te richten met hash access!
- Een tweede bijdrage, "*Pending DDL*", geeft een korte beschrijving van de nieuwe manier waarop DB2 enkele van de nieuwe "online schema changes" implementeert, en wat de implicaties daarvan zijn voor uw DB2-omgeving.
- Dossier 10 heeft het deze keer over "*indicatorvariabelen*", met daarin een opvallende nieuwe mogelijkheid om informatie uit te wisselen met DB2 10 for z/OS.

2

CLOSE CURSOR

Hopelijk genoeg materiaal om zelf aan de slag te gaan met versie 10! En misschien al uit te kijken naar wat een volgende versie nog meer in petto heeft...

Eindelijk: Hash access voor user data in DB2 10!

Kris Van Thillo (ABIS)

De meeste relationele database systemen beschikken over een aantal toegangspaden voor het efficiënt ophalen van user data. Vaak zijn deze paden in te delen in twee grote groepen of 'families':

- een toegangspad gestoeld op scanning van data - men spreekt van tablescans, of tablespace scans in een DB2 omgeving: 'alle' data wordt in fysieke volgorde gelezen;
- een toegangspad gestoeld op index access - op basis van een 'logische key'-waarde wordt een indexstructuur doorzocht; na het vinden van een overeenkomend rij-adres in die index wordt de pagina waarop die rij zich bevindt, opgehaald waarna de relevante rij verder kan worden verwerkt.

Het spreekt voor zich dat DB2 over een hele reeks van alternatieve toegangspaden beschikt. Alternatieven - lees: variaties op - die er moeten voor zorgen dat bovenstaande toegangspaden sneller en efficiënter toegang tot gebruikersdata mogelijk maken (o.a. afhankelijk van beschikbaarheid en type index, het aantal tabellen in de query, de structuur van de select lijst, ...)

Een derde groep toegangspaden bestaat reeds een hele tijd, aanvankelijk echter enkel beschikbaar voor systeemdata (o.a. toegang tot de DB2 directory structuren): hashed data access. In DB2 10 is dit toegangspad eveneens beschikbaar voor gebruikersdata! In wat volgt worden een aantal eigenschappen van hash tabellen uiteengezet. Voor details verwijzen we graag naar ABIS cursussen en naar DB2 manuals.

Inleiding -- uitgangspunt

Het idee is vrij eenvoudig: door het leggen van een rechtstreeks band tussen data (een logische-key-waarde) en de fysieke locatie van die data (RID) in een tablespace zal het aantal I/Os vereist voor het ophalen van die data aanzienlijk dalen.

Dit kan eenvoudig worden bewerkstelligd door het toepassen van een hashing functie (hashing algoritme) op die logische key: de key-waarde wordt aldus vertaald in een RID. De daling waarvan sprake slaat zowel op logische als fysieke I/Os; strikt genomen volstaat nu immers één logische I/O per data rij.

Opzet en organisatie

Om van dit nieuwe accesspad gebruik te kunnen maken, moet een tabel op een specifieke manier worden aangemaakt: concreet dient te worden aangegeven dat de fysieke tabelorganisatie hashing (dus toegang op basis van hashing) moet ondersteunen.

Onderstaand voorbeeld geeft een idee van hoe dit in DB2 10 dient te gebeuren.

Merk op:

- dat wordt aangegeven welke logische keys de input zullen worden voor het hashing algoritme; hashing van deze logische keys moet dus 'mappen' met rij RIDs; de keys moeten uniek zijn (1);
- dat een vaste hash data space wordt aangevraagd en aangemaakt (2) - standaard bedraagt deze 4 GB. De rijen toegevoegd aan de tabel worden hier opgeslagen;
- dat hash organized tabellen dienen te worden aangemaakt in DB2 partitioned universal tablespaces (UTS) (3).

Voorbeeld

```
create table ...
...
partition by range | growth                  (3)
 partition 1 ...
 partition 2 ...
...
organize by hash unique (col_1, col_2)       (1)
         hash space 8G                       (2)
```

Bij het aanmaken van de hash tabel wordt automatisch eveneens een 'hash overflow space' voorbereid (3% tot 5% van de opgegeven hash space wordt hiertoe aan de vaste hash data space toegevoegd). Daarnaast maakt DB2 impliciet een hash overflow index space (al dan niet onmiddellijk, afhankelijk van een aantal ZPARM parameters).

De universal tablespaces (UTS) zijn partitioned 'by growth' dan wel 'by range':

- indien gedefinieerd als 'partitioned by growth' (PBG), dan wordt één hash data space en één overflow space in die ene UTS aangemaakt;
- indien gedefinieerd als 'partitioned by range' (PBR), wordt één hash data space en één overflow space *per range* (dus per partitie) in die ene range partitioned UTS aangemaakt. De partitie-key moet een subset zijn van de hash-key; DB2 kan aldus eerst bepalen welke partitie te weerhouden voor een bepaalde rij, om daarna op basis van hashing de rij in de weerhouden partitie te kunnen opslaan.

De hash overflow index wordt door DB2 zelf beheerd.

Het is evident dat de keuze voor rij allocatie op basis van hashing het gebruik van cluster-indexen onmogelijk maakt.

Hash algoritme

Het hash algoritme bepaalt de voor een logische key te hanteren RID op basis van:

- *de logische key-waarde (de 'key data') op moment van insert/update*: toepassing van een hash functie op deze key genereert een hash waarde (double word integer);
- *het aantal beschikbare pagina's in de vaste hash data space (header/systeem pages worden niet in rekening gebracht)*: de rij wordt opgeslagen op pagina-berekende hash-waarde modulo aantal pagina's in de vaste hash data space. Dit is de row home page;
- *het aantal mogelijke keys per data page (afhankelijk van de page size en de gemiddelde omvang van een rij met standaard max van 255)*: de rijlocatie op de row home page wordt bepaald als de berekende hash-waarde modulo aantal keys per pagina. Deze locatie is eigenlijk een pointer naar een hash anchor map entry positie, die op haar beurt verwijst naar de standaard RID map op diezelfde pagina om de benodigde rij te kunnen terugvinden.

Elke key entry positie in de hash anchor map beslaat twee bytes; het eerste verwijst naar de row entry in de standaard RID map; de tweede byte geeft aan of er nog andere 'identieke' keys zijn die 'elders' werden opgeslagen. Elders, op dezelfde pagina indien mogelijk, of elders, in het daadwerkelijke overflow gebied.

Overflow

Het is steeds mogelijk dat het aantal rijen op een bepaalde row home page een maximum bereikt: omdat het aantal rijen per page wordt benaderd (als steeds, 255 rijen); of omdat de pagina haar maximale opslagcapaciteit heeft bereikt (rijen groeien; of niet alle rijen hebben gemiddeld dezelfde lengte). Rijen die alsnog moeten worden toegevoegd komen in een hash overflow gebied terecht.

Toegang tot deze rijen is mogelijk via een overflow index.

Het ophalen van rijen in een overflow gebied is 'nooit' efficiënt, en dient te worden vermeden. Immers, DB2 dient eerst de gegevens in de row home page op te halen, om dan, op basis van de informatie teruggevonden in de hash anchor map entry, via de overflow index, de benodigde rij in het overflowgebied op te halen.

Het is dus noodzakelijk om de omvang van de vaste hash data space goed in te schatten! Vermits de logische hash keys in de tabel uniek zijn (en data 'skew' niet kan optreden), kan enkel een 'mismatch' tussen het aantal rijen in de tabel en de beschikbare, vooraf gealloceerde hash data space voor overflow zorgen.

Het belang van een goede keuze van deze hash data space size kan niet worden overschat, en bepaalt in grote mate de effectiviteit van het hashing algoritme, en de datatoegang op basis daarvan!

Toegangspaden -- explain

Het spreekt voor zich dat het gebruik van hash-georganiseerde tabellen nieuwe toegangspaden ter beschikking stelt van DB2. Als DB2 deze paden gebruikt, kom dit tot uiting in de `access_type` kolom van de `plan_table` na het vragen van een explain, bijvoorbeeld:

- `H`: full matching hash access -- b.v. where hash_col = value
- `HN`: in list hash access -- b.v. where hash_col in (val, val)

Ook voor een aantal ondersteunende taken kan DB2 beschikbare hash toegangspaden gebruiken, bijvoorbeeld om na te gaan of voor FKs overeenkomende PKs kunnen worden teruggevonden in een hash based parent tabel.

In tegenstelling tot wat vaak wordt aangenomen sluit het werken met hash tabellen het kiezen van andere, meer traditionele toegangspaden natuurlijk niet uit. Enkel paden die gestoeld zijn op de aanwezigheid van clustered indexes - of die enkel dan efficiënt zijn - zijn dus niet beschikbaar.

Utilities

De meeste utilities zijn aangepast om specifiek met hashing overweg te kunnen.

Het REORG utility is in deze ongetwijfeld het belangrijkst. Al dan niet op basis van RTS (Real Time Statistics) biedt het de mogelijk de vaste hash data space te heralloceren om overflow tegen te gaan. Gelijktijdig worden overflow rijen aangepakt, en wordt waar mogelijk overflow beperkt. Het CHECK DATA utility kijkt de overflow structuren (hash anchor map en hash index) na; het CHECK INDEX utility beperkt zich tot de hash index zelf. REBUILD index is eveneens ondersteund, alsook BACKUP en RECOVER (met beperkingen wat betreft point-in-time recovery).

Data laden met het LOAD utility is evenwel een uitdaging: de generatie van hash keys enerzijds, en het inserten op basis van deze keys anderzijds, verloopt traag. Het is niet evident in deze context data ‘presorted’ aan te bieden omdat ‘presorted’ in deze dus niet wijst naar het vooraf sorteren van logische keys in de input data set, maar naar een sortering op basis van een ‘nog te genereren’ hash key!

Gebruik

Wanneer is het gebruik van hash tabellen echt aangewezen? Een kort lijstje met aandachtspunten:

- de kandidaat tabel is relatief statisch, omvangrijk, en de benodigde ruimte kan vooraf worden gepreallocеerd;
- de data in de tabel wordt naar verwachting meestal opgehaald via een ‘H’ access pad -- dus full matching;
- de rijen in de tabel hebben globaal dezelfde structuur, lengte, en worden zelden geüpdated (met impact op rijlengte);

Zelfs indien aan deze voorwaarden is voldaan moet grondig worden geëvalueerd of:

- overflow tot een minimum kan worden beperkt - rijen ophalen uit het overflow gebied heeft een dramatische impact op applicatieperformance;
- hashing geen aanleiding geeft tot 'hot spots' - hashing kan immers aanleiding geven tot dataconcentratie op een beperkt aantal datapagina's!

Tot slot ...

In dit artikel hebben we slechts de basis van tabel hashing aangekaart; de geïnteresseerde lezer verwijzen we graag naar ABIS cursussen om rond dit topic meer te weten te komen.

Zo is het mogelijk, bijvoorbeeld, om de organisatie van bestaande tabellen te 'reorganiseren' naar hash tabellen; of de hashing van een tabel aan te passen dan wel te verwijderen. Vermits dit allemaal operaties zijn met fysieke impact - data organisatie - zijn deze operaties relatief complex, met tijdelijk geen, dan wel verminderde, datatoegang tot gevolg.

DB2 10 for z/OS biedt nog geen mogelijkheid hashing overheen tabellen op te zetten. Denk hierbij aan PK en FK tables, die op basis van een identieke, gemeenschappelijke key PK en FK (tabel)data 'hashen' op één datapagina. Deze datapagina bevat dan alle rijen van alle PK en FK entiteiten met een gemeenschappelijke key/kolom-attribuutwaarde. Hiervoor moet je nog steeds bij Oracle zijn.

Pending DDL *Peter Vanroose (ABIS)*

Online changes versus pending changes

Sinds DB2 v8 is het begrip "*online schema changes*" steeds belangrijker geworden. Met name: objecteigenschappen aanpassen zonder dat daarbij de (tabel)data tijdelijk onbeschikbaar is voor de applicaties. Mogelijke wijzigingen in deze stijl sinds v8 zijn o.a.: kolommen breder maken, kolom-default wijzigen, tabellen en kolommen van naam veranderen, kolommen aan een index toevoegen, partitiegrenzen aanpassen, partities toevoegen, ...

Elk van deze wijzigingen (en ook de "oudere" wijzigingen zoals kolommen toevoegen, en PRIQTY, bufferpool, COMPRESS, LOCKSIZE, PCTFREE, ... van tablespaces wijzigen) is enerzijds ogenblikkelijk (want de catalog weerspiegelt dadelijk de nieuwe realiteit, en nieuwe data is mogelijk die aan de nieuwe realiteit beantwoordt: nieuwe kolom, bredere kolom, ...) maar anderzijds ook *met uitgesteld effect*, want oude records worden pas bij een volgende REORG in lijn gebracht met de nieuwe realiteit.

DB2 10 brengt radicaal verandering in dit hele concept van structuurwijzigingen van een tablespace, althans (voorlopig enkel) voor de *nieuwe* versie-10 "online schema changes" op tablespace-niveau. Wijzigingen zijn niet meer ogenblikkelijk maar worden bijgehouden in een nieuwe catalog-tabel **`SYSIBM.SYSPENDINGDDL`**; de eerstvolgende REORG raadpleegt die tabel en past de DDL-wijziging pas dan effectief toe, tijdens het heralloceren van de tablespace-dataset(s).

Vergeleken met de "oude" situatie, b.v. het breder maken van een kolom, is dit beter voor de performance: de tablespace moet geen twee "types" records ondersteunen; de tablespace blijft 100% in de oude toestand totdat REORG de switch naar de nieuwe toestand realiseert.

Overzicht van de wijzigingen als "pending DDL"

Het gaat hierbij enkel om wijzigingen aan de fysieke structuur van een tablespace of een indexspace, nooit om "zichtbare" wijzigingen op het niveau van de tabel. Bovendien moet de tablespace een **UTS** (universal tablespace) zijn *of worden*: nieuwe features worden sinds versie 9 inderdaad niet meer ondersteund voor de andere types table space (segmented, partitioned, en simple), ook pending DDL niet.

En inderdaad: "worden". Want de meest opvallende nieuwe mogelijkheid, die dan ook via dit nieuwe mechanisme geïmplementeerd wordt, is het converteren van segmented, partitioned of simple tablespaces naar UTS: hetzij naar subtype PBG ("partition by growth"), hetzij naar subtype PBR ("partition by range").

IBM heeft het terzelfdertijd ook mogelijk gemaakt om andere "harde" fysieke interne eigenschappen van de page-structuur van tablespaces te wijzigen op deze manier: nl. de page size (ook voor indexen), de dataset size, en de "member cluster"-structuur.

Alle wijzigingen gebeuren via voor de hand liggende ALTER TABLESPACE syntax (en ALTER INDEX voor BUFFERPOOL):

- SEGSIZE maakt een PBG van een simple TS, en PBR van partitioned.
- MAXPARTITIONS maakt een PBG van simple of segmented.
- BUFFERPOOL laat toe om de page size te wijzigen.
- DSSIZE wijzigt de dataset size
- MEMBER CLUSTER YES of NO wijzigt de interne pageset-structuur.
  (conceptueel identiek aan optie APPEND YES/NO op tabelniveau.)

Advisory reorg pending

Tablespaces met "hangende" wijzigingen van de genoemde aard worden in de nieuwe toestand **AREOR** geplaatst; niet te verwarren met enerzijds de restrictieve REORP (reorg vereist om door te gaan) of de toestand AREO* (advisory reorg pending, oude stijl) die optreedt bij de genoemde "halfslachtige" ogenblikkelijke DDL zoals kolom breder maken. AREOR is 100% "advisory"; de "hangende" DDL kan zelfs eenvoudig ongedaan gemaakt worden, gewoon door een ALTER TABLESPACE DROP PENDING CHANGES. Dit komt gewoonweg neer op het weghalen van alle relevante rijen uit SYSIBM.SYSPENDINGDDL.

De "pending reorg" die de wijzigingen gaat materialiseren mag geen SHRLEVEL NONE meekrijgen, noch een FASTSWITCH NO. Daarnaast hoeft een STATISTICS TABLE ALL INDEX ALL niet opgegeven te worden, want dit is de default wanneer er pending changes zijn.

Bemerk dat de genoemde wijzigingen ook mogelijk zijn voor LOB tablespaces en XML tablespaces. Deze laatste zijn altijd al UTS's (met een niet wijzigbare pagesize van 16K), terwijl LOB tablespaces UTS zijn wanneer de basistabel in een UTS zit. LOB tablespaces komen dus dikwijls "vanzelf" in SYSIBM.SYSPENDINGDDL terecht ...

Het is mogelijk om t.z.t. meerdere "pending changes" klaar te zetten voor dezelfde tablespace, voor in dezelfde REORG dus. Het is echter niet mogelijk om de nieuwe stijl pending changes te "mixen" met oude stijl online schema changes (dus met een AREO*-toestand)!

De eigenlijke DDL ("ALTER" statement) zal steeds een SQLCODE +610 teruggeven, om u eraan te herinneren dat er nog niets gebeurt op dat ogenblik. Anderzijds zal de REORG die de wijziging(en) materialiseert niet alleen de betreffende rijen uit SYSIBM.SYSPENDINGDDL verwijderen, maar ook een extra rij in SYSIBM.SYSCOPY plaatsen (met ICTYPE='A') om zo de ALTER TABLESPACE-historiek bij te houden. Dit is nodig voor het correct interpreteren van (oude) backups.

Conclusie

Het ziet ernaar uit dat deze "pending DDL" een eerste stap is in de richting van een DB2-systeem waar structurele aanpassingen op geen enkel ogenblik belemmerend werkt op het gebruik van de data. Mogelijk wordt dit in de toekomst ook de manier waarop record-structuur-wijzigingen zullen gebeuren (op tabel-niveau dus) ...

DOSSIER 10

Nieuwe mogelijkheden van indicator-variabelen

"NULL-indicatoren" zijn hulpvariabelen in embedded SQL (in b.v. COBOL, PL/1 of C) om aan te geven wanneer een DB2-kolomwaarde NULL is, hetzij vanuit DB2 naar de applicatie, hetzij in de andere richting.

In DB2 10 for z/OS zijn enkele bijkomende waarden geïntroduceerd voor indicator-variabelen, naast de gekende waarden **0** en **-1** die hun betekenis behouden.

DB2 geeft een indicator-waarde **-2** terug wanneer de conversie van de SQL-waarde naar het datatype van de hostvariabele onmogelijk is; dit gebeurt o.a. bij deling door nul, of bij numerieke overflow, b.v. als `123.45` in een variabele terecht komt van slechts twee decimalen voor de komma, of ook bij impliciete conversie van b.v. een accentletter naar een codepage die de betreffende letter niet kent. In al deze gevallen wordt de waarde van de hostvariabele zelf niet ingevuld, enkel die van de indicator, zoals in geval van een NULL dus. Bij afwezigheid van een indicator-variabele wordt in de genoemde gevallen een negatieve SQLCODE teruggegeven (-802 resp. -304), terwijl dat bij aanwezigheid van de indicator (die dus op -2 gezet wordt) de overeenkomstige positieve SQLCODE oplevert (dus +802 resp. +304).

Een strict positieve indicator-waarde duidt op afrondingsfouten als gevolg van een te kleine hostvariabele, maar met een zinvol te gebruiken resultaat; dit was al zo in DB2 9. Zo zal b.v. `'abc'` de afgekapte waarde `'ab'` geven in een host-variabele van lengte 2, maar daarnaast ook de indicatorwaarde +3 om aan te geven dat de oorspronkelijke lengte "3" was. Een afgekapte TIME-waarde zal enkel hh:mm teruggeven, en de seconden as positieve waarde in de indicator-variabele plaatsen.

Een teruggegeven waarde **-3** in DB2 10 betekent dat geen data beschikbaar is voor een bepaalde rij, nl. in het geval van een multi-row FETCH bij een "delete hole", dus bij gebruik van een sensitive scrollable cursor. De SQLCODE is in dit geval +222.

Omgekeerd worden indicatoren ook gebruikt om (bij INSERT of UPDATE) een NULL-waarde aan DB2 door te geven. Met een indicator-waarde -1 dus. Nieuw in versie 10 zijn de waarden -5 en -7.

Met de indicatorwaarde **-5** in een INSERT vraagt het programma aan DB2 om de default-waarde te gebruiken, dus precies zoals bij gebruik van het keyword DEFAULT.

De indicatorwaarde **-7** genaamd "UNASSIGNED" bij een INSERT of UPDATE gedraagt zich alsof de overeenkomstige kolom niet aanwezig was geweest in het statement. Bij UPDATE blijft dus de oude waarde bewaard; bij INSERT wordt de defaultwaarde gebruikt. Update-triggers voor de betreffende kolom worden niet opgeroepen.

Belangrijke opmerking: deze nieuwe interpretatie van -5 en -7 bij INSERT en UPDATE is pas actief bij gebruik van de BIND-optie "EXTENDEDINDICATOR". *By default* blijft de oude interpretatie van kracht, nl. dat een negatieve waarde van een indicator duidt op een NULL, noch min noch meer. Maar gebruik in die gevallen liefst -1.

Andere negatieve waarden dan de genoemde (i.h.b. dus -2, -3, -4 en -6) blijven hun oorspronkelijke betekenis behouden, met name als indicatie voor een NULL.

Peter Vanroose (ABIS)

CURSUSPLANNING APR. – JUN. 2013

DB2 for z/OS, een totaaloverzicht	2175 EUR	06.05(W)
DB2 UDB for LUW, totaaloverzicht	2175 EUR	06.05(W)
RDBMS-concepten	405 EUR	06.05(W), 03.06(L)
Basiskennis SQL	405 EUR	07.05(W), 04.06(L)
DB2 for z/OS basiscursus	1365 EUR	13.05(W)
DB2 UDB for LUW basiscursus	1365 EUR	13.05(W)
SQL-QMF voor eindgebruikers	1365 EUR	op aanvraag
SQL workshop	860 EUR	22.04(L), 30.05(W), 20.06(L)
SQL voor gevorderden	480 EUR	30.04(L), 14.06(W)
Software-ontwikkeling met SQL PL	960 EUR	19.06(W)
DB2 triggers, stored procedures, UDFs	480 EUR	21.06(W)
DB2 for z/OS programmeren voor gevorderden		op aanvraag
DB2 for z/OS: SQL performance	1440 EUR	25.06(W)
XML in DB2		op aanvraag
DB2 for z/OS database administratie	2020 EUR	22.04(L), 10.06(W)
DB2 for z/OS installation and migration	850 GBP	10.06(UK)
DB2 for z/OS data recovery	825 GBP	14.05(UK)
DB2 for z/OS systems performance and tuning	850 GBP	22.05(UK)
DB2 LUW DBA – Kernvaardigheden	1920 EUR	22.04(L), 04.06(W)
Database appicatieprogrammering met JDBC	480 EUR	24.05(W), 24.06(L)
DB2 10 for z/OS: new features		op aanvraag
DB2 10 for LUW: new features		op aanvraag
SQL voor BI	480 EUR	16.05(L)

Plaats: L = Leuven, W = Woerden, UK = High Wycombe (bij Londen);
alle cursussen ook op aanvraag;
voor details en andere cursussen, zie `http://www.abis.be/html/nlTraining.html`
pour détails et autres cours, voir `http://www.abis.be/html/frTraining.html`

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245639
http://www.abis.be/
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Zaagmolenlaan 4
NL-3440 AC Woerden
Tel. 0348-413663
Fax +32-16-245639
http://www.abis.be/
training@abis.be