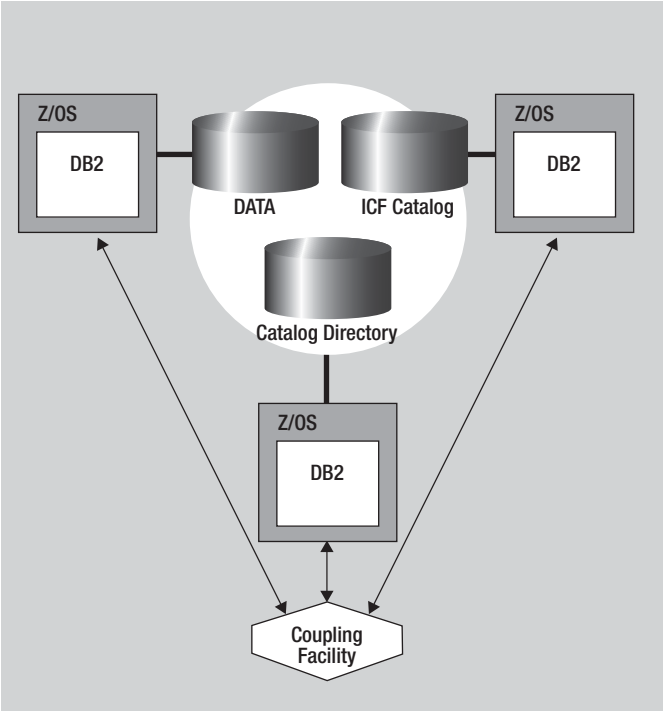
Gegevens verdwijnen niet meer en zullen hooguit migreren

DB2 en 256 Terabytes aan data

Klaas Brant

Heel veel operationele data online hebben is tegenwoordig een normale zaak. Dacht men vroeger nog de data mooi te kunnen scheiden in operationele data en historische data (datawarehouses), tegenwoordig is dat onderscheid compleet aan het vervagen.

Operationele systemen krijgen de beschikking over gigantische hoeveelheden historische data. Deze operational datastores zoals ze zo zaak worden genoemd, zijn erg in en brengen een nieuwe uitdaging. Kijkt u maar eens naar uw online telebankieren. Hoe ver kunt u uw afschriften nu bekijken? Was dit vroeger nog slechts een paar maanden of zelfs weken, nu is dat vaak 12 of 18 maanden en (binnenkort) nog meer. Het komt zelfs vaker voor dat men niet meer kan aangeven waneer data uit de database verwijderd zullen worden: wellicht nooit. De database die alleen maar groeit; dat is een griezelige gedachte!



Afbeelding 1: Data Sharing.

Shared Nothing en Shared Everything

Veel Banken en Verzekeringsmaatschappijen gebruiken DB2 voor de grote operationele systemen. Laten we DB2 eens tegen het licht houden en kijken wat de problemen zijn en waar de grenzen liggen. Om te beginnen moeten we ons afvragen om welke DB2-versie het eigenlijk gaat: DB2 LUW (Linux/Unix/Windows) of DB2 z/OS (mainframe)? De twee DB2-varianten gebruiken namelijk andere technieken om mega-hoeveelheden data te verwerken. LUW gebruikt een shared nothing-architectuur die een aantal servers (nodes) samenvoegt tot één grote MPP server. Larry Allison van Oracle deed ooit de uitspraak: "shared nothing, runs nothing". Maar Larry Allison zit er naast, want deze oplossing is perfect schaalbaar en geeft perfecte resultaten (zie TPC site www.tpc.org voor meer details).

DB2 z/OS gebruikt een shared everything-techniek. Zowel disk als geheugen (coupling facility) van de servers worden gedeeld en op deze manier kan een giga-server gemaakt worden. Toch is dit alles van totaal onderschikt belang, veel operational stores worden niet gebruikt voor analyse maar gewoon om data langer online te houden. Als u via uw online telebankieren nog een beetje verder terug gaat in de tijd, dan gaat niet om een typische datawarehouse query maar om een simpele transactie. Dus moeten we gaan kijken wat de problemen zijn bij online transactieverkeer (liefst erg veel) en gigantische hoeveelheden data. Dit betreft vrijwel altijd een mainframe-omgeving, dus ik zal mij beperken tot de z/OS-omgeving.

Data Sharing

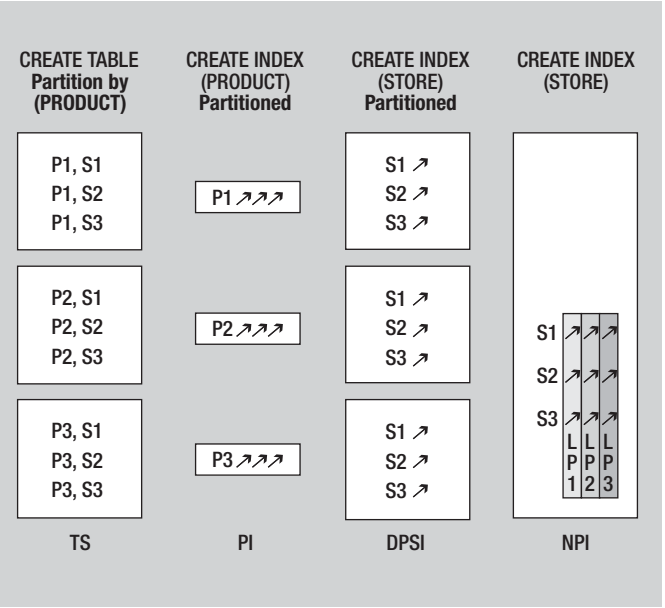
De omgeving waarbij meerdere systemen samen een cluster vormen heet in DB2-termen een data sharing-omgeving. Meerdere DB2's (members) vormen samen een grotere DB2 (Group), terwijl ze allemaal individueel in staat zijn om de

transacties te processen. In deze omgeving is er dus geen sprake van een master/slave-relatie tussen de members. Ieder member is een master die eventueel een ander member kan vragen om samen aan een query te werken (parallel SQL processing over meerdere members). Dit heeft alleen zin als de query een bepaalde omvang heeft zoals bij datawarehousing.
Voor transactie-processing zal in 99.9 procent van alle gevallen één member de query processen. Als men gebruik maakt van data sharing dan is dit om een zéér hoge beschikbaarheid te kunnen krijgen, omdat de members (die op verschillende machines draaien) elkaars backup zijn. Data sharing en zéér grote hoeveelheden data gaan dus niet per definitie hand in hand. Het is heel goed mogelijk om in een enkelvoudige DB2-omgeving enorme hoeveelheden data op te slaan. Data sharing kan de oplossing zijn voor de tabel met 10 rijen die men voor 100 procent beschikbaar wil hebben.

DB2 for z/OS slaat zijn gegevens op in tablespaces. Tablespaces bestaan in een aantal varianten. De twee meest populaire zijn *segmented* en *partitioned* tablespaces. In een segmented tablespace kunnen een of meerdere tabellen bestaan die dan ieder hun eigen opslagruimte krijgen in de vorm van segmenten. Er zijn enkele performance-aspecten die bepalen of men één of meerdere tabellen wil opslaan in een segmented tablespace. Veel bedrijven hebben een richtlijn dat men slechts één table per tablespace mag hebben. Dit is een strategie die niet vol te houden is als men net als bij SAP meer dan vele tienduizenden tabellen heeft. Fysiek op disk slaat DB2 een segmented tablespace op in één of meerdere datasets. Echter alle datasets samen kunnen samen niet groter worden dan 64 Gigabyte. Wil men meer dan 64 Gigabyte opslaan dan moet men verplicht gaan partitioneren. Het maximum aantal partities is 254 in versie 7 en 4096 in versie 8. Iedere partitie kan 64 Gigabyte groot worden en een partitioned tablespace kan slechts één tabel bevatten. In versie 8 kan een tabel dus 256 Terabyte groot worden.

> Data sharing en zéér grote hoeveelheden data gaan dus niet per definitie hand in hand

Wel moet men bij partitioning goed de restricties zoals die in de manuals staan in de gaten houden. Indien de tablespace reeds in versie 6 of 7 is aangemaakt en niet onder controle staat van SMS (DSSIZE keyword), dan zijn veel nieuwe features van versie 8 niet te gebruiken. Het aanpassen vereist dan een drop/create; dat is met grote hoeveelheden data een moeizame taak. En hoe zit dat met Binairy Large OBjects (BLOB)? BLOB's zijn geen normale data, maar betreffen de mogelijkheid om in een enkele rij een



Afbeelding 2: Version 8 partitioned tablespace with possible indexes.

enorme hoeveelheid data (foto's, audio-bestanden, XML-documenten etcetera) op te slaan. Ook DB2 kent deze mogelijkheden, maar omdat het om speciale data gaat blijft dit verder buiten beschouwing.

Partitionering

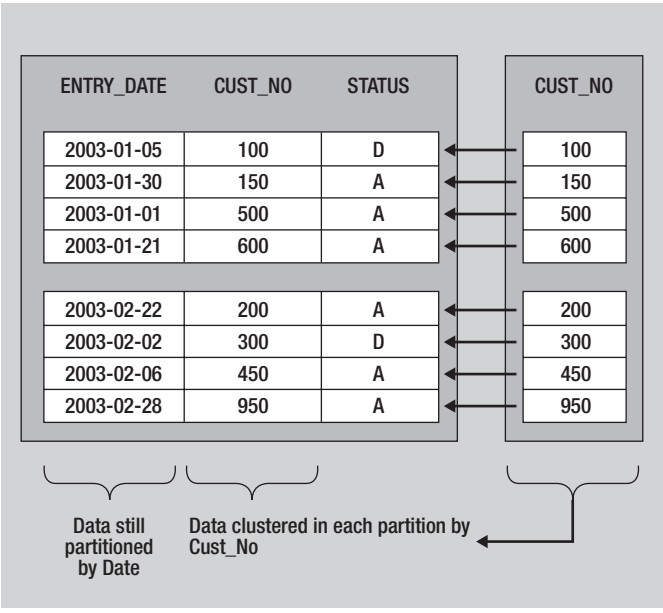
Er zijn twee redenen om te partitioneren: de hoeveelheid data komt boven 64 Gigabyte óf men heeft data die logisch in stukken te delen zijn en men wil slim gebruik maken van de individuele stukken. Slim gebruik heeft in dit geval diverse betekenissen: allereerst de SQL optimizer die aan de hand van predicates in de query weet dat het geen zin heeft om te gaan zoeken in bepaalde partities. Hiermee kan een query slechts een deel van de data doorzoeken, wat de responstijd van query zeer ten goede komt. Een goed voorbeeld is als men de data partitioneert op jaarbasis: alle gegevens van één jaar zitten netjes bij elkaar in één partitie. Als de query nu ook een jaarselectie in zich heeft zoals bijvoorbeeld: SELECT * FROM DATA WHERE JAAR=2000, dan zal DB2 alléén naar de partitie van 2000 gaan en de andere partities links laten liggen, omdat daar geen data te vinden zijn die aan de selectiecriteria voldoen. Nog een vorm van slim gebruik is dan men de partities onafhankelijk van elkaar kan behandelen met utility's. Zo kan een bepaalde partitie geladen worden via een LOAD utility of gereorganiseerd worden met een REORG utility, zonder dat de SQL query's op de andere partities hier last van hebben. Dit fenomeen wordt partition independence genoemd. Hierover volgt later meer, want er zijn ook beperkingen aan.

Naast slim gebruik van query's en utility's kan men ook slim gebruik maken van DDL-opties. Met DDL kan worden aangeven dat de grenzen van de partities veranderd worden. Zo kan men partities bijmaken (bijvoorbeeld een nieuwe partitie met gegevens

van een nieuw jaar) of partities hergebruikt worden (bijvoorbeeld de gegevens van de partitie met data uit 1995 worden overschreven met gegevens uit 2005). Het laatste staat ook wel bekend als *rolling partitions*. Ook heeft DB2 speciale opties om een query sneller te verwerken door de query in meerdere parallelle processen op te delen (query parallellisme). Van nature werkt dit beter op partitioned data omdat er natuurlijke grenzen aanwezig zijn.

Partition Independence

Reeds vanaf versie 4 komt het aspect partition independence bij iedere nieuwe release weer terug als onderwerp. Door het CLAIM/DRAIN-principe, dat geïntroduceerd is in DB2 versie 4, werd partition independence mogelijk gemaakt. Iedere partitie heeft zijn eigen dataset(s) en ook de partitioning index is in stukken gehakt, zodat het mogelijk werd om individuele partitions via het DRAIN-principe los te maken van SQL processing.
Maar zodra men op gepartitioneerde data extra indexen (secondary indexen) gaat aanbrengen, dan bestrijken deze indexen de gehele data. Vandaar dat zo'n index de naam Non-Partioning Index (NPI) kreeg. Deze NPI's zijn de dwarsliggers bij partition independence. Zodra een partitie opnieuw geladen wordt met data of gereorganiseerd wordt, dan moet deze NPI opnieuw gebouwd of bijgewerkt worden, met als gevolg dat deze onbeschikbaar is voor SQL. Dit is dus geen partition independence.
In versie 8 is het principe van partitioning geheel op de helling gegaan en zijn er veel nieuwe zaken met partitions mogelijk. Om te beginnen kunnen data gepartitioneerd worden zonder indexen. De keyranges worden bij de aanmaak van de tabel reeds opgegeven via het PARTITIONED BY keyword. Het gebruik van indexen is optioneel geworden. Maar men moet bedenken dat DB2 voor het uniek maken van één of meerdere velden een index nodig heeft. Een goed ontwerp heeft een primary key die uniek hoort te zijn. Iedere index kan vanaf versie 8 gepartitioneerd zijn. Is de index betrokken op dezelfde velden als waarop gepartitioneerd is, dan noemen we de index (zoals vroeger) een partitioning index (PI). Maar ook andere indexen, die vroeger verplicht non-partitioning waren, kunnen nu gepartitioneerd worden door het PARTITIONED keyword, dit worden dan Data Partitioned Secondary Indexen. Dit geeft als afkorting DPSI en deze worden in de volksmond 'dipsies' genoemd. Natuurlijk bestaan de NPI's ook nog steeds. Door alleen gebruik te maken van PI en/of DPSI's is data independence een feit geworden.



Afbeelding 3: DPSI als clustering index.

> In versie 8 is het principe van partitioning geheel op de helling gegaan

DPSI voor- en nadelen

Naast partition independence heeft de DPSI nog een voordeel; namelijk de mogelijkheid om aangewezen te worden als de clustering index. Was in versie 7 en lager de PI per definitie de clustering index, nu is het mogelijk om een DPSI of zelfs een NPI aan te wijzen als clustering index. Voor een NPI is dat niet zo zinnig maar voor een DPSI kan dit voordelen hebben. In het verleden wilde men graag de data opsplitsen, maar waren de kolommen waarop dat gebeurde zeer ongewenst voor de clustering. Het klassieke voorbeeld is het opsplitsen van data in time series (partition by year) met daarbij de wens om binnen de partitie te clusteren op klantnummer. Dit was onmogelijk in versie 7 maar mogelijk in versie 8 met een DPSI. Echter, de DPSI heeft ook (grote) nadelen. Zo is een DPSI per definitie een non-unique index. Aangezien dezelfde waarde van de DPSI key in iedere partitie kan voorkomen zou DB2 bij iedere partitie moeten checken (index probe) om te kijken of een waarde uniek is. Omdat versie 8 nu 4096 partities kan maken zou dit 4096 checks kunnen beteken. Omdat dit de performance niet ten goede komt is dat dus niet mogelijk. Dit betekent dus dat als we niet willen partitioneren op de primary key, we toch nog onze toevlucht moeten nemen tot NPI's voor de primary key.

Maar dat is niet het enige nadeel. Als we via een NPI meerdere records gaan ophalen van dezelfde key (non-unique NPI), dan benaderen we slecht één key en vinden hierbij de RID's naar de diverse partities waar de records te vinden zijn. Gaat dit echter via een DPSI dan zullen we dus voor iedere partitie een index probe moeten doen om te kijken of de key in deze partitie voorkomt. DB2 is wel in staat om selectief partities over te slaan, als uit andere predicates van de query blijkt dat deze partities geen data

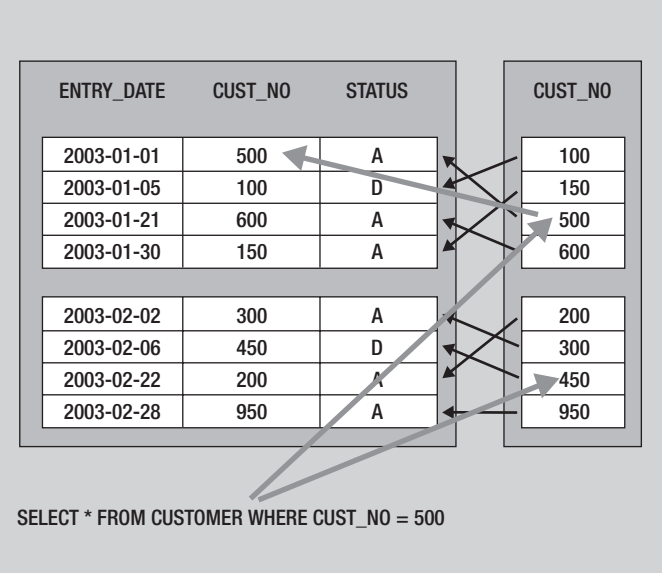
kunnen leveren. DPSI zijn dus een fantastische optie voor partitie-independence, maar kunnen op SQL-gebied dus 'des duivels' zijn. Met name in ad hoc query-omgevingen moet men erg uitkijken. Maakt u zich zorgen om de aantallen IO's op de NPI's (queueing), kijk dan eens naar de piecesize-parameter die de NPI kunstmatig in stukken breekt. Deze stukken hebben niets met partities te maken (en lossen dus ook het partition independence-probleem niet op), maar verdelen het aantal IO's dus wel wat meer over de (geforceerde) datasets. Als het enige probleem de partition independence tijdens de REORG is (BUILD2 Phase), dan wordt er in het lab druk aan gewerkt om ook hiervoor een oplossing te vinden in de vorm van mirroring in plaats van het bijwerken tijdens de BUILD2. Wellicht zit dit in de volgende versie van DB2.

Het delete-probleem

Een moderne variant van het spreekwoord "Ik wens u veel personeel" zou kunnen zijn "Ik wens u veel data"; want ik wens niemand 256 Terabyte aan data toe. Niet alleen is het beheer van een dergelijke hoeveelheid een probleem, maar ook de groei. En alsof dat nog niet voldoende is dan komt toch vroeg of laat ooit een keer het probleem van 'hoe kom ik er van af'. Het is zelfs een kritische vraag geworden bij een design review doe. Het is te gemakkelijk om de 'kop-in-het-zand-strategie' te volgen en te zeggen "dat zien we later dan wel". Wat voor optie heeft DB2?

> Het is te gemakkelijk om de 'kop-in-het-zand-strategie' te volgen

Allereerst is er de mogelijkheid om de data van een partitie te replacen of het principe van rolling-partities toe te passen.
Dit is eenvoudig, maar wel een soort strategie van 'one size fits all'. Maar al te vaak wordt er aan dataclassificatie gedaan, waarbij sommige data na een paar maanden of een paar jaar weg zijn en andere data veel langer bewaard moeten blijven. Is dat het geval dan is alleen maar te adviseren om reeds tijdens het ontwerp kritisch te zijn en te kijken hoe de classificatie meegenomen kan worden bij de partitioning-strategie.
Wil men selectief van de data af, dan kan men natuurlijk een DELETE statement loslaten op de data, maar dat zou betekenen dat bij veel data deleten er ook veel gelogd zou worden. Daarom is het goed te weten dat de REORG utility een mogelijkheid heeft om tijdens de REORG data via selectiecriteria tijdens de reorg te laten verdwijnen. Omdat u de REORG niet wilt loggen, zullen deze rows uit de database verdwijnen zonder plaats in te nemen op de recovery log. Dit is een perfecte oplossing die nog niet voldoende door DB2-gebruikers toegepast wordt. Natuurlijk moet



Afbeelding 4: DPSI multiple index probe in SQL.

u zelf de RI (al dan niet DB2 enforced) goed in de gaten houden, want een fout is zo gemaakt en maar weinig mensen weten procedures om PK-FK relaties te checken.

Nooit meer deleten

Wat moet men doen als men data wel wil bewaren maar de kans dat deze data benaderd worden erg klein is? Dan zijn er tegenwoordig ook producten op de markt die in staat zijn om DB2-data te archiveren. Het lijkt dus dat de data nog aanwezig zijn, maar in werkelijkheid zijn ze gemigreerd naar een background medium zoals een cartridge, dat is goedkoper. Mochten de data alsnog nodig zijn dan kunnen ze redelijk snel weer teruggehaald worden van deze background media.
Dit soort producten wordt gemaakt door Storagetek en Princeston Softech. Het lijkt een mooie optie maar deze strategie is niet altijd toepasbaar; denk bijvoorbeeld maar eens aan een tablespace scan die dan gaat proberen om alle gemigreerde data weer terug te halen. Dit zou een regelrechte ramp zijn.
Toch zijn er toepassingen voor dergelijke producten en ook de leveranciers die deze technologie ontwikkelen maken hun producten steeds beter, verfijnder en sneller. Maar het blijft een raar idee dat data nooit meer verdwijnen en hooguit zullen migreren.

Klaas Brant (kbrant@kbce.nl) is DB2-specialist en directeur van KBCE b.v. Meer informatie over DB2 is te vinden op www.kbce.nl en www.db2-times.com