ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 2 - NUMMER 4, DECEMBER 2003

OPEN CURSOR

U hebt het laatste nummer van dit jaar in uw handen.
Een nummer waar performance enerzijds, en DB2 versie 8 anderzijds, een belangrijke positie innemen.
Voor dit jaar zijn we uitgepraat. Even een paar weekjes relatieve rust!
Relatief, want het moment is aangebroken om onze projecten van het najaar af te ronden; om resultaten en conclusies op papier te zetten. En om nieuwe projecten voor het voorjaar toe te wijzen.
Dit alles om ons kennisaanbod up-to-date te houden en om nieuwe artikels voor Exploring DB2 te kunnen voorbereiden.
Het volgende nummer mag u verwachten rond 15 februari 2004.
Het hele ABIS team wenst u prettige eindejaarsfeesten, en een uitdagend 2004!

Het ABIS DB2-team.

IN DIT NUMMER:

- Het laatste deel in onze reeks DB2 applicatieperformance - *DB2 performance - case 4*.
- In *Dossier 8* staan we stil bij *multiple row selects*.
- Reeds gekend van DB2 voor LUW - nu ook op z/OS in versie 8, *Materialized Query Tables* (MQT).
- *Cursusplanning januari 2004 - maart 2004*.

CLOSE CURSOR

In het volgende nummer - .Net configuratie en optimisatie in DB2. En extra aandacht voor utilities.

Tot dan!

DB2 performance - case 4

Eric Venmans (ABIS)

Inleiding

Indien de performance van een SQL-query te wensen overlaat, wordt in vele omgevingen te snel naar het wapen van de 'index' gegrepen. Maar zijn er geen alternatieven? Volgende 'case' is daar een illustratie van. Zoals in de vorige artikels in deze reeks, is het voorbeeld opzettelijk eenvoudig gehouden om de kern van het probleem duidelijk in de verf te zetten.

CASE 4: probeer eerst SQL-varianten!

Een online transactie moet een gebruiker snel een overzicht kunnen geven van alle producten:

- met een leveringstermijn groter dan een opgegeven waarde (meestal een hoge waarde)
- waarvan minstens één voorraad is uitgeput.

Hiertoe moet de SQL in die transactie zich richten tot minstens twee tabellen; deze worden hieronder beschreven.

Voorbeeld 1: de tabellen

```
Create TABLE Producten
(ProductID    Char(6)     Not Null,
 .....
 PRLeverterm  Integer     Not Null with default,
 .....
 Primary Key (ProductID))

Create UNIQUE INDEX PKPrdkIX
on Producten(ProductID) ...

Create index PRLtrmIX
on Producten(PRLeverterm) ...

Create TABLE Voorraden
(VR_ProductID     Char(6)   Not Null,
 VR_MagazijnID    Char(4)   Not Null,
 .....
 VRAantalstuks    Integer   Not Null with default,
 VRStatus         Char(1),
 VRLaatsteLev     Date,
 .....
 Primary Key (VR_ProductID, VR_MagazijnID),
 Foreign key FK1 (VR_ProductID)references Producten on delete cascade,
 .....

Create UNIQUE INDEX PKVrrdIX
on Voorraden(VR_ProductID, VR_MagazijnID) ...
```

```
SELECT ProductID, .....
FROM Producten
WHERE PRLeverterm > :X
AND VR_ProductID IN (
  SELECT VR_ProductID
  FROM Voorraden
  WHERE VRAantalstuks = 0)
```

De SQL die boven genoemde online transactie implementeert, zou dan als hiernaast aangegeven kunnen geformuleerd worden.

Het toegangspad dat de DB2 optimizer voor deze query gebruikt is echter ver van ideaal. We kijken daarvoor in detail naar voorbeeld 2.

Voorbeeld 2: initieel accespad

QB	Mthd	Table Name	Access	Match Cols	Index Name	Index Only	Sort N	Sort C	Prefetch
1	0	PRODUCTEN	N	1	PKPRDKIX	N	NNNN	NNNN	
2	0	VOORRADEN	R	0		N	NNNN	NNNN	S
2	3			0		N	NNNN	YNYN	

De subquery (QBlock = 2) wordt eerst uitgevoerd. Hierbij worden alle voorraadrijen sequentieel gelezen (Access = R). Als voor een gelezen rij VRAantalstuks gelijk is aan 0, wordt de identificatie (VR_ProductID) van het bijhorende product in een tussenresultaat bewaard. Nadien worden deze identificaties geordend (Method = 3) om dubbels te elimineren en om nadien het zoeken in het tussenresultaat te optimaliseren.

Na het uitvoeren van de subquery, wordt de 'outer select' (QBlock = 1) uitgevoerd. Elke waarde uit het tussenresultaat van de subquery wordt gebruikt om via de index op de productidentificatie (PFPRIDIX) de bijhorende productrij op te halen. Na controle van de leveringstermijn wordt het betreffende product al dan niet in het eindresultaat getoond.

Optimalisatiepoging

Kernprobleem bij dit toegangspad is het sequentieel lezen van alle voorraden.

Een eenvoudige oplossing dringt zich op - we creëren een index op een voor dit SQL-statement relevante kolom, bijvoorbeeld op VRAantalstuks.

```
Create index VRAstkIX
on Voorraden(VRAantalstuks)
...
```

Het toegangspad dat de DB2 optimizer nu gebruikt, is heel wat beter (zeker op vlak van het lezen van gegevens). Dit blijkt duidelijk uit voorbeeld 3 op de volgende bladzijde. Het mechanisme blijft hetzelfde, alleen wordt de subquery een stuk lichter. Het sequentieel lezen van alle voorraden is vervangen door het opzoeken van de 'uitgeputte' voorraden via de nieuwe index. Zoveel te selectiever de voorwaarde 'VRAantalstuks = 0' zoveel te groter de besparing door via de nieuwe index te zoeken.

Voorbeeld 3: accespad eerste indexvariant

QB	Mthd	Table Name	Access	Match Cols	Index Name	Index Only	Sort N	Sort C	Prefetch
1	0	PRODUCTEN	N	1	PKPRDKIX	N	NNNN	NNNN	
2	0	VOORRADEN	I	1	VRASTKIX	N	NNNN	NNNN	L
2	3			0		N	NNNN	YNYN	

Als men dan toch een nieuwe index wil toevoegen, kan deze nog beter gedefinieerd worden op een aantal kolommen, met name VRAantalstuks en VR_ProductID:

```
Create index VRAstPIX
on Voorraden(VRAantalstuks,
VR_ProductID) ...
```

Het toegangspad dat DB2 nu gebruikt, staat in voorbeeld 4. Het uitvoeren van de subquery wordt een index-only proces. De aangegeven sort (Method = 3) is bovendien overbodig (alle index entries met eenzelfde VRAantalstuks zitten geordend op product identificatie). Dit laatste wordt echter niet 'gezien' door de optimizer, of wordt toch niet getoond via de explain.

Voorbeeld 4: accespad tweede indexvariant

QB	Mthd	Table Name	Access	Match Cols	Index Name	Index Only	Sort N	Sort C	Prefetch
1	0	PRODUCTEN	N	1	PKPRDKIX	N	NNNN	NNNN	
2	0	VOORRADEN	I	1	VRASTKIX	Y	NNNN	NNNN	
2	3			0		N	NNNN	YNYN	

Alternatieve oplossing

De oplossing waarbij een nieuwe index gecreëerd wordt heeft één groot nadeel, namelijk het onderhoud van die index. Er zullen allicht een hele reeks mutaties gebeuren van VRAantalstuks, zeker als bij elke verkoop de voorraad van het betrokken product online wordt aangepast. Ook een reorganisatie van de betrokken tabel zal meer energie vragen.

Het creëren van de index en de extra inspanningen die het systeem nadien moet leveren om de inhoud up-to-date te houden, kunnen echter vermeden worden. De query kan immers herschreven worden als een join!

```
SELECT DISTINCT ProductID,
.....
FROM Producten, Voorraden
WHERE ProductID =
VR_ProdukID
AND VRAantalstuks = 0
AND PRLeveringstermijn > :X
```

Het toegangspad dat de DB2-optimizer voor deze query gebruikt is beter dan de oorspronkelijke oplossing, ook zonder extra index - zie voorbeeld 5. We zijn automatisch het sequentieel lezen van alle voorraden kwijt, omdat DB2 begint met eerst een product te zoeken met een leveringstermijn groter dan de opgegeven waarde, om dan de voorraden ervan op te zoeken via een bestaande index (deze op de primary key).

Een ideale oplossing is dit echter nog steeds niet.

Voorbeeld 5: accespad JOIN query

QB	Mthd	Table Name	Access	Match Cols	Index Name	Index Only	Sort N	Sort C	Prefetch
1	0	PRODUCTEN	I	1	PKPRDKIX	N	NNNN	NNNN	
1	1	VOORRADEN	I	1	VRASTKIX	N	NNNN	NNNN	L
1	3			0		N	NNNN	YNNN	

Elk geselecteerd product wordt gecombineerd met al zijn voorraden die uitgeput zijn. Daarna pas worden dubbels geëlimineerd (zie Method = 3). We willen een product immers slechts één keer in ons resultaat zien (vandaar DISTINCT).

Ook deze laatste 'overhead' kunnen we elimineren door onze query te herschrijven als een 'gecorreleerde subquery' met het sleutelwoord 'EXISTS'.

```
SELECT ProductID, .....
FROM Producten P
WHERE PRLeverterm > :X
AND EXISTS (
SELECT 'exists'
FROM Voorraden
WHERE VRAantalstuks = 0
AND VR_ProdukID =
P.ProductID)
```

Het toegangspad is nu nog eenvoudiger - zie voorbeeld 6. DB2 herschrijft de query wel als een join (Method 1 'nested loop join'), maar houdt onmiddellijk rekening met het feit dat de tweede tabel (voorraden) enkel benaderd wordt voor een bestaanscontrole. Zodra DB2 voor een geselecteerd product een eerste uitgeputte voorraad heeft gevonden, is de 'EXISTS'-voorwaarde voldaan en wordt niet verder gezocht naar andere voorraden voor het betrokken product. Er wordt per product maximaal één combinatie gemaakt met een uitgeputte voorraad. De 'SORT' om dubbels te elimineren vervalt.

Voorbeeld 6: accespad EXISTS query

QB	Mthd	Table Name	Access	Match Cols	Index Name	Index Only	Sort N	Sort C	Prefetch
1	0	PRODUCTEN	I	1	PKPRDKIX	N	NNNN	NNNN	
1	1	VOORRADEN	I	1	PKVRRDIX	N	NNNN	NNNN	L

Ook de kostenberekening van DB2 (aangegeven in de DSN_STATEMNT_TABLE) bevestigt dat de laatste oplossing de goedkoopste is.

Besluit

Het beschreven voorbeeld toont aan dat het optimaliseren van queries niet enkel een taak is voor database administrators. Er zijn ook een aantal queries waarbij het creëren van een index maar een gedeeltelijke oplossing is. Het herschrijven van de query is soms beter en vooral goedkoper, tenminste als het in een vroeg stadium gebeurt (d.w.z. tijdens het schrijven van de toepassing). Indien de problemen pas vastgesteld worden als de toepassing al gemigreerd is naar een productieomgeving, is ook het herschrijven van een query niet meer zo evident en 'goedkoop'.

DOSSIER 8

Multiple Row Select

In een vorige editie van Dossier 8 hadden we het reeds over 'multiple row inserts'.

In dit nummer, willen we het over 'multiple row selects' hebben. Hierbij staat het concept 'rowset' centraal.

Een rowset bestaat uit die groep rijen die door middel van één enkele fetch, uit de resultaatstabel van een query worden opgehaald. Deze groep rijen wordt door DB2 als één logisch geheel behandeld, bijvoorbeeld vanuit locking standpunt. Typisch wordt het aantal te lezen rijen opgegeven bij het uitvoeren van de fetch-operatie zelf.

EXEC SQL FETCH FIRST ROWSET STARTING AT ABSOLUTE 5 FROM C1 FOR 10 ROWS INTO :r1, :r2;

Host-variabelen 'r1' en 'r2' zijn gedefinieerd als rijen, in een host taal. Het fetch-statement werd gewijzigd om de omvang van de rowset te kunnen aangeven. Indien niet opgegeven, wordt de rowset-grootte bijvoorbeeld impliciet bepaald door de omvang van de vorige rowset van dezelfde cursor. Merk op dat niet elke fetch een zelfde aantal rijen moet opleveren, en dat niet elke rowset aanéénsluitend moet gefetcht worden. Voorts kunnen individuele rij-fetches gecombineerd worden met rowset-fetches op dezelfde cursor. Belangrijk is te beseffen dat een single-row-fetch altijd wordt geïnterpreteerd relatief t.o.v. de eerste rij in de vorige rowset. DB2 genereert een 'SQLCODE +100' indien een rowset-fetch het eind van een resultaattabel heeft bereikt. In combinatie met instructies die specifiek zijn voor scrollable-cursor, biedt deze uitbreiding aan elke applicatieontwikkelaar een groot scala aan mogelijkheden.

Opdat rowset-fetching mogelijk zou zijn, moet bij de declaratie van de betreffende cursor ROWSET POSITIONING geactiveerd zijn; dit is default niet het geval. Bij het openen en/of alloceren van een cursor kan de GET DIAGNOSTICS instructie gebruikt worden om na te gaan of ROWSET POSITIONING werd geactiveerd.

Locks worden gehouden op alle rijen in de rowset; het locking-gedrag zelf wordt zoals steeds bepaald door een aantal traditionele factoren: isolatieniveau (CS, RR, RS, UR), en de bind 'current data' instelling. De inhoud van een rowset reflecteert steeds een toestand op één bepaald moment in de tijd. Het scroll-effect doorheen een cursor die rowsets ophaalt, kan dus aanleiding geven tot het ontstaan van 'holes', 'phantoms', etc.

Tenslotte nog even vermelden dat ook de 'cursor controled update' en 'cursor controlled delete' rowsets aankunnen. Denk eraan dat de rijen die u wil wijzigen in de rowset misschien reeds door een andere applicatie werden benaderd. De kans dat dit gebeurt is groter naarmate de rowset groter is.

Tom Avermaete (ABIS)

Over MQT - Materialized Query Tables

Kris Van Thillo (ABIS)

Het idee achter MQT's (Materialized Query Tables) op DB2 for z/OS and OS/390 versie 8 is vrij eenvoudig: materialiseer in een soort van tijdelijke tabel het resultaat van een complexe query; en zorg ervoor dat deze tijdelijke structuur automatisch en transparant gebruikt kan worden op het moment dat de DB2-optimizer het gebruik van deze structuur als efficiënter beoordeelt dan het gebruik van de oorspronkelijke tabellen waarop deze complexe query is gebouwd. Tenslotte moet dan nog even stil gestaan worden bij technieken die men wil gebruiken om de MQT en de eigenlijk tabellen 'in sync' te houden.

MQT's, vroeger Automatic Summary Tables genaamd, bestaan reeds geruime tijd op het DB2 for LUW (Linux, UNIX, Windows) platform.

Dit vergt een extra woordje uitleg.

Een voorbeeld

Een MQT is strikt genomen een tabel die afgeleide data bevat. Afgeleid, in de zin dat het typisch om geaggregeerde data gaat (vandaar ook hun oude naam) waarvan de brondata reeds in één of een aantal tabellen aanwezig is - zie ook voorbeeld 1. Alhoewel de definitie van een MQT sterk lijkt op deze van een view, is er toch één belangrijk verschil: MQT's zijn gematerialiseerd, d.w.z. dat ze fysiek ruimte in beslag nemen. Views daarentegen zijn enkel logische constructies. MQT's hebben ook iets weg van een index, omdat normaal gezien het gebruik van MQT's transparant is voor de gebruiker. Inderdaad, DB2 bepaalt zelf wanneer een MQT wordt gebruikt, en DB2 is zelf verantwoordelijk voor het onderhouden van de inhoud van de MQT.

Voorbeeld 1: Aanmaken van een MQT

```
create table MQT_courserev
as
(select c.cid, c.cstitle, sum(s.sincomes) as cincomes
 from courses c, sessions s
 where s_cid = cid
 group by c.cid, c.cstitle)
data initially deferred
refresh deferred
maintained by system
enable query optimization
```

Het idee MQT op zich is niet nieuw. Vaak hebben een aantal vooruitziende DBA's - voornamelijk in DSS of datawarehouse-omgeving - een aanverwant concept geïmplementeerd. Aggregaten worden in afzonderlijke tabellen opgeslagen en bepaalde applicaties worden verplicht op de data in deze tabellen beroep te doen. Met evidente

nadelen: hoe onderhouden we de data in deze tabellen (toegegeven, niet zo complex in een datawarehouse-stage)? Maar vooral: het gebruik van deze tabellen is niet transparant voor de finale gebruiker.

Deze nadelen behoren spoedig tot het verleden - dit dankzij een aantal interessante karakteristieken van MQT's die in wat volgt uiteengezet worden.

De MQT 'summary' query

De MQT-query moet voldoen aan de syntaxeisen van een fullselect. Alle SELECT-constructies zijn dus eigenlijk toegelaten. Merk op dat, conform het 'summary' idee, deze constructies in veel gevallen een eenvoudige JOIN in combinatie met een GROUP BY zullen bevatten.

Initiële opbouw

Ook het moment van initiële opbouw - onmiddellijk na creatie van de MQT - moet worden opgegeven. De optie 'data initially deferred' geeft aan dat de MQT pas zal worden 'opgevuld' met data nadat een refresh werd uitgevoerd. Deze optie is op dit moment verplicht - het onmiddellijk opbouwen - 'build immediate' - wordt (nog?) niet ondersteund.

Synchronisatie van de inhoud

Een van de belangrijkste uitdagingen geassocieerd met het gebruik van MQT's, is de problematiek van de synchronisatie met de eigenlijke basistabellen. DB2 for z/OS and OS/390 voorziet 2 mogelijkheden:

- de optie 'maintained by user' geeft weer dat DB2 voor het onderhoud van deze MQT niet verantwoordelijk is. Dit heeft bijvoorbeeld als gevolg, dat aan de hand van UPDATE, DELETE en INSERT (of LOAD) statements, de MQT kan worden bijgewerkt. Het 'REFRESH TABLE'-statement is ook beschikbaar. Voor dit type MQT's wordt bij een refresh de cataloginformatie niet bijgewerkt.

- de optie 'maintained by system' (default) geeft weer dat DB2 zelf met de datasynchronisatie wordt belast. UPDATE, DELETE en INSERT (of LOAD) statements op de MQT zijn niet langer toegestaan. Het 'REFRESH TABLE'-statement moet hiervoor worden aangewend.

Indien van 'REFRESH TABLE' gebruikt wordt gemaakt, wendt DB2 intern volgende procedure aan om tot een volledige synchronisatie van de MQT te komen:

- de MQT wordt leeggemaakt a.d.h.v. een (mass)delete; voor performance-redenen gebruikt men dus meestal een segmented tablespace;
- de MQT-fullselect wordt opnieuw uitgevoerd, en het resultaat wordt in de MQT-tabel opgeslagen;
- de DB2-cataloog wordt aangepast: het tijdstip van refresh alsook de statistische informatie die de MQT beschrijft, worden gewijzigd.

Merk op dat dit een relatief omslachtige procedure is. DB2 kent geen techniek van incrementele refresh (zoals bijvoorbeeld Oracle), waarbij de MQT up-to-date wordt gebracht a.d.h.v bijvoorbeeld een logstructuur. Dit implementeren kan enkel indien gebruik wordt gemaakt van 'user maintained' synchronisatietechnieken. Denk hierbij bijvoorbeeld aan triggers, aangemaakt op de basistabellen, die bij elke wijziging op de basistabel de beschrijving van deze wijziging registreren in een logtabel. Op het moment van de refresh wordt de logtabel gebruikt - bijvoorbeeld aan de hand van een stored procedure - om de MQT aan te passen, waarna de logtabel wordt leeggemaakt. Gegeven het feit dat de MQT meestal op een geaggregeerd niveau zal worden opgebouwd, is deze refresh manueel opzetten niet echt een eenvoudige taak. In versie 7 van DB2 for OS/390 was het het enige alternatief!

Ook een meer synchrone benadering, waarbij bijvoorbeeld de MQT op het moment van de commit van datawijziging op de basistabellen automatisch zou aanleiding geven tot het updaten van MQT's, wordt op dit moment niet ondersteund.

Query rewrite

Hier is het uiteindelijk allemaal om te doen!

'Query rewrite' - de mogelijkheid die DB2 biedt om een query op een basistabel te vertalen in een (aangepaste) query op een MQT. Met als doel een betere antwoordtijd, want deze MQT zou in omvang aanzienlijk kleiner moeten zijn dan de basistabellen.

Figuur 1 geeft een overzicht van de belangrijkste eigenschappen van 'Query rewrite'.

Conceptueel gaat de DB2 optimizer het volgende na:

- Bestaat er structureel en inhoudelijk voldoende overeenkomst tussen de user-query en de query op basis waarvan de MQT is gedefinieerd? Is bijvoorbeeld de user-query identiek aan de definitie van de MQT?

- Is het mogelijk eventueel ontbrekende informatie af te leiden, te 'berekenen'? Het standaardvoorbeeld: de user vraagt een gemiddelde; de MQT heeft enkel een 'som' en een 'totaal' ter beschikking. Of via een 'joinback' structuur naar de basistabel haalt men een ontbrekende kolom op.

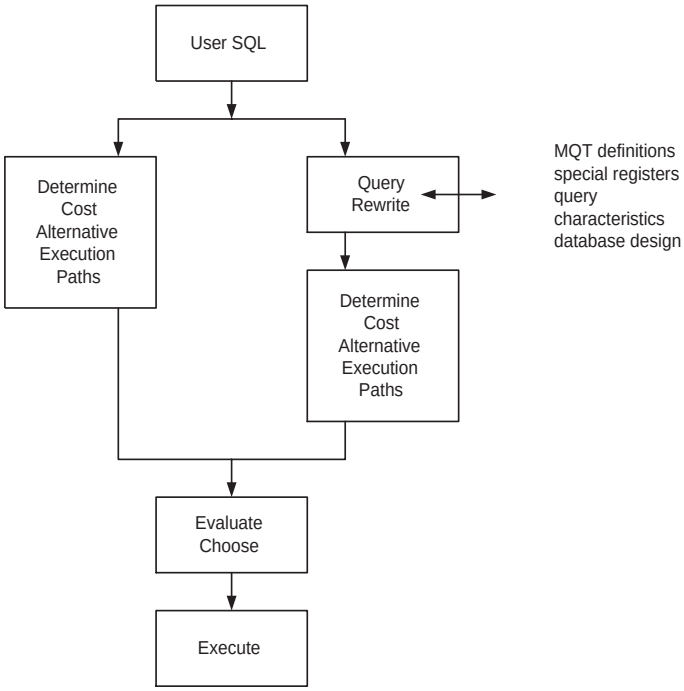
- Wordt de integriteit van het eindresultaat bewaard?

Indien een MQT inderdaad kan worden aangewend, herschrijft DB2 de user-query op basis van de MQT-definitie. De kost hiervan wordt bepaald en vergeleken met de initiële kost van de user-query. Het goedkoopste alternatief wordt zoals altijd weerhouden en uitgevoerd. Of een MQT al dan niet wordt gebruikt kan natuurlijk het makkelijkst worden bekeken aan de hand van de PLAN_TABLE: bij een TABLE_TYPE-indicatie 'M' geeft TNAME de naam van de MQT weer.

Het gedrag van de Query rewrite feature van DB2 wordt door een aantal elementen gestuurd.

De gespecificeerde opties bij het aanmaken van een MQT spelen een rol: enkel MQT's aangemaakt met eigenschap 'enable query optimization' komen voor 'Query rewrite' in aanmerking.

Figuur 1: Query rewrite



Twee nieuwe 'special registers' spelen eveneens een rol by 'Query rewrite', en wel als volgt:

- CURRENT REFRESH AGE, aan de hand waarvan wordt meegegeven wat de maximaal toegestane verlopen tijd is tussen het huidige moment CURRENT TIMESTAMP en het moment van de laatste MQT-refresh. Specificeer '0' en MQT's worden niet gebruikt; specificeer ANY en alle MQT's worden in aanmerking genomen. Andere waarden zijn niet toegelaten.

De bedoeling van dit register is niet onmiddellijk duidelijk. We moeten ons met name afvragen wat belangrijk is: de relatie tussen het huidige moment CURRENT TIMESTAMP en het moment van de last refresh, dan wel de relatie tussen dit laatste moment en het moment van de laatste UPDATE van de overeenkomende basistabellen...

▪ CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION, die aangeeft welke MQT's in aanmerking komen voor de rewrite: ALL, NONE, SYSTEM (i.e. enkel 'system maintained' MQT's), dan wel USER (i.e. enkel 'user maintained' MQT's).

Ook de structuur van de user-query speelt een belangrijke rol. De query kan op zich uit verschillende query-blokken bestaan - maar de individuele query-blokken moeten deterministisch van aard zijn. UPDATE- en INSERT-statements, alsook user-queries op MQT's worden niet herschreven.

De aanwezigheid van referentiële integriteit. Aan de hand van traditionele RI-regels bepaalt DB2 of bepaalde user-queries kunnen worden vervangen door queries op MQT's. Dit zal voornamelijk gebeuren, als de user-query moet worden herwerkt in een join van verschillende MQT's dan wel in een join tussen een MQT en een gewone tabel. Om de overhead (zeker in datawarehouses) van gewone, traditionele RI-constraints te vermijden, heeft DB2 versie 8 een nieuwe vorm van RI beschikbaar: 'not enforced'-integriteit. Deze integriteit wordt volledig genegeerd door DB2 tijdens traditionele processing; enkel tijdens het 'Query rewrite' proces wordt met deze RI rekening gehouden.

Toch een aantal belangrijke opmerkingen.

- MQT's worden enkel geëvalueerd in de context van 'dynamically prepared' SQL - op zich niet echt onverwacht, gezien het belang dat moet worden gehecht aan de correctheid van de evaluatie van de beschikbaarheid van een MQT als alternatief voor een user-query;
- Het isolatieniveau van de MQT moet minstens gelijk zijn aan het isolatieniveau van de user-query;
- MQT's zijn gematerialiseerd en nemen dus ruimte in. Hun onderhoud vergt bijkomende ressources;
- Het behoort tot de taak van de DBA te beslissen welke MQT's aan te maken; en het is de taak van de DBA na te gaan in hoeverre bepaalde MQT's gebruikt worden of niet. Dit alles leunt nauw aan bij traditioneel INDEX beheer;
- Op dit moment beschikt DB2 niet over een 'MQT Advisor', die bijvoorbeeld op basis van workloads, MQT's suggereert.

Besluit

Datawarehouse omgevingen worden vaak opgebouwd volgens een 'ster'- of 'sneeuwvlok'-structuur, bestaande uit een grote, centrale fact-tabel, in combinatie met een aantal kleinere dimensietabellen. Vaak verwachten gebruikers data uit deze fact-tabel geaggregeerd terug te vinden, op basis van de waarden in de verschillende dimensietabellen. Deze aggregaten vormen uitstekende kandidaten om als MQT te worden geïmplementeerd.

CURSUSPLANNING JAN 2004 - MRT 2004

DB2 concepten	375 EUR	08/03(L)
DB2 for OS/390, een totaaloverzicht	1625 EUR	12-16/01 (L), 26-30/01(W), 01-05/03 (L), 29/03-02/04 (W)
DB2 UDB, een totaaloverzicht	1625 EUR	01-05/03 (L)
RDBMS concepten	325 EUR	12/01 (L), 26/01 (W), 01/03 (L), 29/03 (W)
Basiskennis SQL	325 EUR	13/01(L), 27/01(W), 02/03 (L), 30/03 (W)
DB2 for OS/390 basiscursus	975 EUR	14-16/01 (L), 28-30/01(W), 03-05/03 (L), 31/03-02/04 (W)
DB2 UDB basiscursus	975 EUR	03-05/03 (L)
SQL workshop	700 EUR	09-10/02 (W), 11-12/03 (L)
DB2 for OS/390 programmering voor gevorderden	1050 EUR	08-09/03 (L)
Gebruik van DB2 procedural extensions	350 EUR	10/03 (L)
DB2 for OS/390: SQL performance	1200 EUR	24-26/03 (L)
DB2 UDB applicatieperformance	400 EUR	08/06 (W)
Database applicatieprogrammering met Java	800 EUR	29-30/04 (L)
Fysiek ontwerp van relationele databases.	700 EUR	03-04/05 (L)
DB2 for OS/390 database administratie	1600 EUR	15-18/03 (W)
DB2 for OS/390 operations and recovery	1650 EUR	11-13/02 (UK)
DB2 UDB systeembeheer en performance	400 EUR	30/04 (L)
DB2 UDB en zijn extenders: XML en text search	200 EUR	12/03 (L)
DB2 UDB integratie met MQSeries	200 EUR	12/03 (L)

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