Nieuwe features Oracle 11g (3)

Partitioning en Result Caching

Naast nieuwe handigheidjes voor beheer, zoals er in de vorige aflevering een paar aan bod kwamen, zijn er ook verbeteringen aangebracht op het gebied van performance. In deze aflevering wil ik er twee behandelen. Eerst worden de uitbreidingen op het gebied van partitioning behandeld. In het tweede deel ga ik in op de nieuwe result-cache.

Partitioning is in Oracle al heel lang beschikbaar, sinds versie 8. Dave Ensor beschrijft in Oracle Insights, Tales of the Oak Table (Apress, 2004) dat deze optie toen werd uitgebracht onder grote druk van klanten. Voordat Oracle partitioning uitbracht als optie (tegen extra kosten) bij de Enterprise Edition werd er soms al een eigen vorm van partioning gehanteerd door inventieve gebruikers. Daarbij werden gegevens van een jaar gesplitst over bijvoorbeeld 12 tabellen, waarbij iedere tabel de gegevens van één maand bevatte. Een view voegde de informatie van deze 12 tabellen vervolgens met union operaties weer samen. Op deze manier konden queries over één maand beperkt worden tot de ene tabel, terwijl nog steeds aggegraties over alle gegevens van het hele jaar mogelijk waren. Deze werkwijze wordt nog steeds toegepast, vooral in situaties waar slechts Standard Edition beschikbaar is, of de gebruikersorganisatie niet bereid is te betalen voor de Partitioning-optie op Enterprise Edition. De werkwijze is ook wel bekend onder de naam Poor Mans Partitioning (PMP).

Oracle Partitioning is op het eerste gezicht niet veel anders dan de hierboven beschreven werkwijze. De gebruiker definieert een tabel, en geeft aan dat de gegevens in deze tabel over een reeks partities moeten worden verdeeld. Intern zijn deze partities als afzonderlijke tabellen opgeslagen. Het is ook mogelijk deze tabellen in aparte tablespaces onder te brengen, zodat I/O operaties over verschillende schijven kunnen worden verdeeld. Het stuk interne adminstratie dat Oracle biedt met deze optie, en de extra mogelijkheden voor het op geavanceerde wijze manipuleren van partities, zoals afzonderlijk vullen, verwijderen etc. maken het een redelijk complex stuk software. Het kost wat, maar dan heb je ook wat. Heel veel zaken die met PMP handmatig moeten worden geregeld worden door de Oracle Partitioning Option automatisch geregeld. Een actie die nog steeds handmatig moest worden uitgevoerd was het toevoegen van partities, bijvoorbeeld bij partitionering op basis van kalendermaand.

Er is bijvoorbeeld een tabel aangemaakt voor de opslag van bijvoorbeeld de email headers van de laatste drie maanden van het afgelopen jaar. De (sterk gesimplificeerde) tabel ziet is met het volgende statement gecreëerd:

```
SQL> CREATE TABLE  emh
   2 (mail_id       NUMBER        NOT NULL
   3 ,sent_d        DATE          NOT NULL
   4 ,sender        VARCHAR2(80) NOT NULL
   5 ,recipient     VARCHAR2(80) NOT NULL
   6 )
   7 PARTITION BY RANGE (sent_d)
   8 (PARTTION part_0710 VALUES LESS THAN
  10            (TO_DATE('0111200700000', 'ddmmyyyyhh24miss'))
  11 ,PARTITION part_0711 VALUES LESS THAN
  12            (TO_DATE('0112200700000', 'ddmmyyyyhh24miss'))
  13 ,PARTITION part_0712 VALUES LESS THAN
  14             (TO_DATE('0101200800000', 'ddmmyyyyhh24miss'))
  15 )
  16 /
```

Een mailtje dat wordt verstuurd door Sinterklaas aan de Kerstman, om te melden dat deze het werk mag overnemen, wordt keurig verwerkt:

```
SQL> INSERT INTO emh (mail_id, sent_d, sender, recipient)
   2 VALUES           (1, TO_DATE('05122007 203000', 'ddmmyyyy hh24miss')
   3                   ,'hulpsint@pakjesboot12.nl' ,'kerstman@noordpool.fi');

1 row created.
```

Op het moment dat nu de Kerstman in 2008 aan Sinterklaas een nieuwjaarswens stuurt en het e-mailprogramma probeert het mailtje op te slaan gebeurt echter het volgende:

```
SQL> INSERT INTO emh (mail_id, sent_d, sender, recipient)
   2 VALUES           (2, TO_DATE('01012008 070500', 'ddmmyyyy hh24miss')
   3                   ,'kerstman@noordpool.fi', 'hulpsint@pakjesboot12.nl');

INSERT INTO emh (mail_id, sent_d, sender, recipient)
            *
ERROR at line 1:
ORA-14400: inserted partition key does not map to any partition
```

De DBA heeft kennelijk vergeten tijdig de partitie aan te maken voor de komende maand. Veel databases waarin gebruik gemaakt wordt van partitioning op basis van tijd worden periodiek door de DBA voorzien van de partities voor de komende periode. Bij andere systemen is dit geautomatiseerd met scripts, maar ook dan is het een storingsgevoelig punt. Het blijkt altijd weer dat er om één of andere reden een rij wordt toegevoegd in de toekomst, in een periode waarvoor de partities nog niet zijn aangemaakt. Of de scriptjes voor het aanmaken van de partities waren gescheduled met cron, en het systeem was net in onderhoud op het moment dat de scripts werden gedraaid. Murphy vindt altijd een mogelijkheid om goede bedoelingen in de wielen te rijden.

Met ingang van Oracle 11g kan ook dit toevoegen van partities worden geautomatiseerd. Daarvoor moet de tabel wel op een iets andere manier worden aangemaakt:

```
SQL> CREATE TABLE  emh
   2 (mail_id      NUMBER       NOT NULL
   3 ,sent_d       DATE         NOT NULL
   4 ,sender       VARCHAR2(80) NOT NULL
   5 ,recipient    VARCHAR2(80) NOT NULL
   6 )
   7 PARTITION BY RANGE (sent_d)
   8 INTERVAL (NUMTOYMINTERVAL(1, 'MONTH'))
   9 (PARTITION part_0710 VALUES LESS THAN
  10            (TO_DATE('0111200700000', 'ddmmyyyyhh24miss'))
  11 )
  12 /
```

Hierbij wordt gebruik gemaakt van de functie NUMTOYMINTERVAL om de datum in sent_d te vertalen naar maanden. Het is verplicht om initieel tenminste 1 partitie aan te maken. In het voorbeeld is de partitie voor de maand oktober 2007 aangemaakt. Als de structuur van de tabel in de data dictionary wordt geraadpleegd blijkt dat ook:

```
SQL> select partition_name, high_value
   2 from   user_tab_partitions
   3 where  table_name = 'EMH';

PARTITION_ HIGH_VALUE
---------- -------------------------------------------------
PART_0710  TO_DATE(' 2007-11-01 00:00:00', 'SYYYY-MM-DD
           HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
```

Nu worden dezelfde twee mailtjes als in het eerdere voorbeeld ge-insert:

```
SQL> INSERT INTO emh (mail_id, sent_d, sender, recipient)
   2 VALUES           (1, TO_DATE('01012008 070500', 'ddmmyyyy hh24miss')
3 ,'kerstman@noordpool.fi', 'hulpsint@pakjesboot12.nl');

1 row created.

SQL> INSERT INTO emh (mail_id, sent_d, sender, recipient)
   2 VALUES           (2, TO_DATE('05122007 203000', 'ddmmyyyy hh24miss')
   3                   ,'hulpsint@pakjesboot12.nl' ,'kerstman@noordpool.fi');

1 row created.
```

Als nu opnieuw gekeken wordt welke partities de tabel emh heeft blijkt dat de database er zelf twee heeft toegevoegd:

```
SQL> select partition_name, high_value
   2 from   user_tab_partitions
   3 where  table_name = 'EMH';

PARTITION_ HIGH_VALUE
---------- -------------------------------------------------
PART_0710  TO_DATE(' 2007-11-01 00:00:00', 'SYYYY-MM-DD
           HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')

SYS_P41    TO_DATE(' 2008-02-01 00:00:00', 'SYYYY-MM-DD
           HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')

SYS_P42    TO_DATE(' 2008-01-01 00:00:00', 'SYYYY-MM-DD
           HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
```

Noot van de auteur: De oplettende lezer zal misschien zien dat de door het systeem gegenereerde sequence-naam identiek is aan die in het voorbeeld van Arup Nanda, die op OTN over dit onderwerp heeft gepubliceerd. Deze overeenkomst viel mij ook op. Ik heb het uitgezocht, en het blijkt dat een 'vers' gecreëerde 11g database begint met nummer 41, vanuit de sequence PARTITION_NAME$. Het testen van deze functionaliteit op basis van een verse 11g database geeft dus een grote kans op identieke resultaten.

Het automatisch maken van nieuwe partities neemt de DBA uiteraard een steeds terugkerende taak uit handen, en het voorkomt fouten als er een nieuwe rij wordt toegevoegd die is bestemd voor een nog niet aangemaakte partitie. Het lost echter niet alles op. Waar wordt die nieuw toegevoegde partitie opgeslagen? Het is mogelijk om dit aan te geven bij het creëren van de tabel:

```
SQL> CREATE TABLE  emh
   2 (mail_id      NUMBER       NOT NULL
   3 ,sent_d       DATE         NOT NULL
   4 ,sender       VARCHAR2(80) NOT NULL
   5 ,recipient    VARCHAR2(80) NOT NULL
```

```
 6 )
 7 PARTITION BY RANGE (sent_d)
 8 STORE IN (EMH1, EMH2, EMH3, EMH4)
 9 INTERVAL (NUMTOYMINTERVAL(1, 'MONTH'))
10 (PARTITION part_0710 VALUES LESS THAN
12             (TO_DATE('0111200700000', 'ddmmyyyyhh24miss'))
13             TABLESPACE EMH1
14 )
15 /
```

Daarbij zijn EMH1..EMH4 de namen van de tablespaces waarin de partities zullen worden opgeslagen. Oracle doet dit erg netjes. Bij het aanmaken van een partitie wordt niet, zoals ik vreesde, de eerstvolgende partitie uit het rijtje genomen. Er wordt uitgerekend in welke tablespace de partitie terecht had moeten komen, als ze op volgorde zouden zijn aangemaakt. In het voorbeeld worden in tablespace EMH1 de partities voor oktober, februari en juni aangemaakt. Dat gebeurt ongeacht de volgorde waarin inserts, bestemd voor een bepaalde partitie, plaatsvinden. Voor veel situaties zal dit voldoende zijn. Ik kom in de praktijk echter ook situaties tegen waarin iedere partitie een eigen tablespace krijgt. In dat geval zal het toevoegen van partities handmatig moeten blijven gebeuren.

Interval partitioning in plaats van range partitioning zal de DBA van een vervelende steeds terugkomende taak kunnen verlossen, en systemen robuuster maken doordat rijen die 'te vroeg' worden toegevoegd automatisch de juiste partitie laten aanmaken. Het blijft wel zaak de vinger aan de pols te houden met betrekking tot de beschikbare schijfruimte en in de gaten te houden welke partities er allemaal zijn. Een rij die foutief wordt aangemaakt, zal zelfs als de betreffende transactie met een rollback ongedaan wordt gemaakt, toch de bijbehorende partitie hebben laten aanmaken. Deze wordt met de rollback niet meer verwijderd.

Caching

Caching is al sinds jaar en dag de methode om trage schijven zoveel mogelijk te omzeilen. Door gegevens die frequent worden geraadpleegd in het interne geheugen op te slaan wordt het ophalen van die gegevens aanzienlijk versneld. Die snelheid wordt door ontwikkelaars nog wel eens overschat. Omdat geheugen 'oneindig' snel is maakt het niet uit hoeveel gegevens hoe vaak worden gelezen. Een aardig voorbeeld daarvan kwam ik ruim een jaar geleden tegen. Ik werd gevraagd met spoed naar een magazijnsysteem te komen kijken. Het bedrijf was vrijwel tot stilstand gekomen, omdat het pasgeleden in gebruik genomen systeem niet meer performde. Daardoor werden er geen goederen meer afgeleverd, en hoopten de voorraden zich op. Toen ik de 'crisisruimte' binnenkwam was het Quest Spotlight scherm via de beamer op een groot scherm geprojecteerd. Midden in het scherm stond een bijna fluorescerend groene balk me aan te grijnzen. Erboven stond het getal 100.0%. Het was de Buffer Cache Hit Ratio (BCHR). 100.0% van alle gegevens werden uit de cache gelezen, dus de database MOEST wel snel zijn. Om een lang verhaal kort te maken: er konden zo'n 8 vrachtwagens per uur geladen worden met de beschikbare heftruck- en dockcapaciteit. De applicatie rekende echter 80.000 keer per uur het gewicht van een vrachtwagen uit. Dat was nu eenmaal het generieke concept van de applicatie. Nu kan een cache nog zo snel zijn, een leesoperatie van Oracle heeft iets meer om het lijf dan simpelweg transporteren van een paar bytes van het geheugen naar de processor en omgekeerd. Het beschermen van al deze lees-operaties tegen concurrerende processen (collega-gebruikers) vergt de inzet van o.a. latches en locks. Alle 'overhead' zorgt ervoor dat het ophalen van een rij uit de cache maar zo'n 15-30 keer zo snel is als het ophalen van die rij van schijf.

Hoe snel geheugen, processoren, schijven en netwerken ook worden, er is niets op tegen om software te ontwerpen en schrijven met een gezonde hoeveelheid gedrevenheid tot het schrijven van optimale code. Te vaak wordt geschreven met het adagium 'als het traag is gooien we er wel wat extra ijzer tegen aan'. KIWI. Kill It With Iron. Ik heb zelfs meegemaakt dat applicatie-architecten de ontwikkelaars expliciet verboden met DBA's te praten. Dat zou alleen maar het risico opleveren dat ze code zouden schrijven die 'databaseafhankelijk' zou zijn. Tonnen investeren in database-software, maar dan geen gebruik willen maken van die dure functionaliteit. Een gemiste kans, maar wel steeds vaker de realiteit.

Om deze reden wordt RDBMS-software dan ook meer en meer voorzien van features om tekortkomingen in applicatiesoftware op te vangen. Mijn eerste reactie is om de Result Cache, die met Oracle 11g zijn intrede doet, onder deze noemer te scharen.

De result cache houdt niet de gegevens die eerder van schijf gelezen zijn vast, maar het complete resultaat van uitgevoerde queries. Dat geldt voor de hele instance. Als twee sessies kort na elkaar dezelfde query uitvoeren, èn de onderliggende gegevens zijn in de tussenliggende tijd niet gewijzigd, dan wordt de tweede keer eenvoudig het hele klaarstaande resultaat van de query gebruikt. Een eerste voorwaarde voor zinvol gebruik van deze functionaliteit is dat de server over een forse hoeveelheid geheugen beschikt. Resultaten van queries zijn niet zelden groot tot zeer groot. De result cache is onderdeel van de shared pool. Oracle zal maximaal 75% van de shared pool gebruiken voor de result cache. Het gedeelte dat van deze 75% mag worden gebruikt voor het resultaat van een enkele query kan met de parameter RESULT_CACHE_MAX_RESULT worden ingesteld. Deze parameter specificeert een percentage. De toegestande waarde ligt tussen 1 en 100, de standaard waarde is 5%.
De result cache kan op twee manieren worden geactiveerd: Door

de hint `/*+ result_cache */` op te nemen in de SQL statements, of door de result cache voor de hele instance aan te zetten met behulp van een parameterinstelling. In het laatste geval kan voor bepaalde statements de result cache weer worden uitgeschakeld met de hint `/*+ no_result_cache */`.

Om één en ander te testen heb ik een demo-scriptje gemaakt, waarin eerst een tabel werd gemaakt op basis van dba_objects (een kopie van de eerste 999 rijen) en vervolgens een query op deze tabel werd uitgevoerd. De tabel had ik gemaakt omdat de documentatie aangeeft dat de result cache niet werkt op objecten uit de data dictionary. Ik werkte echter onder de gebruiker SYS. De definitie van 'objecten uit de data dictionary' moest kennelijk worden uitgebreid tot 'objecten eigendom van SYS'. Daarom ging ik verder onder SYSTEM, maar ook daarbij werd de result cache niet aangesproken. Uiteindelijk heb ik een gebruiker SCOTT/TIGER aangemaakt, en daarmee werkte het wel:

```
select object_type, avg(object_id)
from   test_objects
group  by object_type;


Elapsed: 00:00:00.02

Execution Plan
----------------------------------------------------------
Plan hash value: 3353918500

--------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |     7 |    77 |   274   (1)| 00:00:04 |
|   1 |  HASH GROUP BY      |              |     7 |    77 |   274   (1)| 00:00:04 |
|   2 |   TABLE ACCESS FULL| TEST_OBJECTS |  1000 | 11000 |   273   (0)| 00:00:04 |
--------------------------------------------------------------------------------


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       1005  consistent gets
          0  physical reads
          0  redo size
        692  bytes sent via SQL*Net to client
        420  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          7  rows processed
```

Daarna werd het statement nog eens uitgevoerd, vanuit een andere sessie:

```
7 rows selected.

Elapsed: 00:00:00.01

Execution Plan
----------------------------------------------------------
Plan hash value: 3353918500

--------------------------------------------------------------------------------
| Id  | Operation           | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                           |     7 |    77 |   274   (1)| 00:00:04 |
|   1 |  RESULT CACHE       | 5syn2hwmwftvk9abkqk1bp3fh4 |       |       |            |          |
|   2 |   HASH GROUP BY     |                           |     7 |    77 |   274   (1)| 00:00:04 |
|   3 |    TABLE ACCESS FULL| TEST_OBJECTS              |  1000 | 11000 |   273   (0)| 00:00:04 |
--------------------------------------------------------------------------------


Result Cache Information (identified by operation id):
------------------------------------------------------

   1 - column-count=2; dependencies=(SCOTT.TEST_OBJECTS); parameters=(nls); name="select object_type,
avg(object_id)
from test_objects
group by object_type"


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
          0  redo size
        692  bytes sent via SQL*Net to client
        420  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          7  rows processed
```

Duidelijk is uit de statistieken te zien dat er geen block gets meer worden gedaan. Het executieplan laat in regel 1 zien dat de result cache wordt gebruikt.

Veel tijdwinst is er in de test niet behaald, maar dat heeft te maken met de kleine omgeving en kleine hoeveelheid data waarmee is getest. In grote systemen zal de result cache in gevallen waar vaak dezelfde data (die bovendien niet te vaak mag worden gewijzigd) wordt opgehaald een interessante verbetering kunnen laten zien. Toch blijft het een vorm van het paard achter de wagen spannen: Als de data al is gelezen, waarom wordt dat dan steeds weer opnieuw gedaan? Zoals Cary Millsap pleegt te zeggen: Het snelste SQL is nog altijd het SQL dat niet wordt uitgevoerd, omdat het overbodig is. Als dat uitgangspunt bij de ontwikkeling van de software een beetje in acht wordt genomen, is de result cache misschien helemaal niet nodig.

Carel-Jan Engel werkt als onafhankelijk Oracle-consultant. Hij is lid van het Oak Table Network.
E-mail: cjpengel.dbalert@xs4all.nl.

Colofon

Vereniging
OBC, opgericht 29 juni 1998

Adresgegevens
Oracle Business Club,
p/a OC Centor BV
Postbus 124
3430 AC Nieuwegein

Bestuur
Voorzitter:
Ruud Rijsmus

Eerste secretaris:
Wessel van Alphen

Tweede secretaris:
Margaret Stavast

Penningmeester:
Christel Petroci

Website
www.odbclub.org

Contact
secretariaat@odbclub.org

Sponsor
Oracle

Bedrijfspresentaties

Netwerken tussen bedrijven die zich met bezighouden met dienstverlening of producten rondom Oracle is de essentie van de OBC. Om leden beter in staat te stellen kennis te nemen van de sterke punten van collega bedrijven, worden bedrijfspresentaties gegeven. Vorig jaar is daarmee begonnen en ook in 2008 zal die mogelijkheid worden geboden.

De businessmeetings die de OBC gedurende het jaar organiseert staan voor een belangrijke deel in het teken van netwerken. Het doel van netwerken is om te bekijken of bedrijven elkaar kunnen helpen. Dat kan bijvoorbeeld door het inhuren van elkaars medewerkers of het gezamenlijk uitvoeren van projecten. Kennisuitwisseling over wat er in de markt gaande is, is een ander belangrijk aspect. Eveneens belangrijk is het om nieuws te vernemen van Oracle zelf als leverancier. Ook in het geval er presentaties worden gegeven, worden de business meetings zo georganiseerd dat er voldoende ruimte is om te netwerken.
Bij netwerken is het wel van belang om met de juiste mensen in contact te komen. Oftewel met mensen van bedrijven waar mogelijk een samenwerking mee kan plaatsvinden of interessante informatie mee gedeeld kan worden. In 2007 is door diverse leden aan het bestuur verzocht of het niet mogelijk is zichzelf en hun bedrijf kort te presenteren. Naast bijvoorbeeld een korte omschrijving op de website van OBC stelt dit leden in staat gerichter contact met elkaar te zoeken. Het bestuur van OBC vond dit een goed idee en heeft hiervoor gelegenheid geboden. Quobell was de eerste organisatie die zich presenteerde.

Ook in 2008 wil het bestuur genoemde gelegenheid bieden. Op de eerste business meeting van 2008, die gepland staat op 4 maart, is er die gelegenheid. Ten tijde van het schrijven van dit artikel is een eerste aanmelding binnen, namelijk van Truston.
Bedrijfspresentaties moeten kort maar krachtig zijn, ze mogen niet langer dan 15 minuten duren. Er mogen er ook niet meer dan drie plaatsvinden tijdens een business meeting. Per slot van rekening moet er altijd tijd overblijven om te netwerken. Zijn er geen bedrijfspresentaties dan kan het zijn dat het bestuur een spreker uitnodigt. Er moet dan wel een thema behandeld worden dat van belang is voor de business. In voorkomende gevallen kan dat ook een spreker van Oracle zijn. Maar ook dan geldt dat de spreektijd beperkt is. De vooraankondiging van de aanstaande business meeting op 4 maart is al via e-mail verstuurd. Als de agenda definitief is geworden, worden de gebruikelijke uitnodigingen verstuurd. Bent u nog geen lid en heeft u interesse? Neem dan contact met ons op. U kunt altijd een keer vrijblijvend kennismaken en introducés zijn welkom.

Namens het bestuur,
Ruud Rijsmus, voorzitter