Histogrammen & Bind variabelen

Performance en tuning

Zoals met veel performance-tuning zaken, valt op dat er onder DBA'ers nogal wat mythen bestaan aangaande de combinatie van histogrammen en binds. Eén van de mythen is dat het te allen tijde goed is histogrammen aan te maken en dat het belangrijk is bind variabelen te gebruiken. Maar hoe zit het met de combinatie van deze twee?

In het hierna volgende zal ik met behulp van een voorbeeld aangeven dat histogrammen de performance negatief kunnen beïnvloeden indien ze worden gebruikt in combinatie met bind variabelen.

Aanmaken schema en tabel

```
=>sqlplus '/ as sysdba'

SQL*Plus: Release 9.2.0.6.0 - Production on Wed Apr 27 13:01:25 2005

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP and Oracle Data
Mining options
JServer Release 9.2.0.6.0 - Production

SQL> create user rk identified by rk default tablespace users temporary
tablespace temp;

User created.

SQL> grant create session, select_catalog_role, resource, alter session
to rk;


Grant succeeded.

SQL> conn rk/rk
Connected.
```

Het is voor deze test de bedoeling dat we een niet uniform verdeelde tabel verkrijgen. Ik gebruik daarvoor dba_objects met daarin objecten die *niet* uniform verdeeld zijn over de verschillende schema's. Onderstaand wordt daarom een tabel aangemaakt die grotendeels lijkt op dba_objects. Aan de hand van de object-owner wordt het user_id in de test-tabel opgenomen, deze is dan niet uniform verdeeld en zorgt voor een uitstekend voorbeeld:

```
SQL> create table mytest as
  2     (select dos.OBJECT_ID
  3             , dos.OBJECT_NAME
  4             , dosOBJECT_TYPE
  5             , dos.OWNER
  6             , dus.USER_ID
  7      from   dba_objects dos
  8             , dba_users  dus
  9      where  dos.owner = dus.username);

Table created.

SQL> insert into mytest (select * from mytest)
/  2

58910 rows created.
...
SQL> insert into mytest (select * from mytest)
  2  /

942560 rows created.

SQL> commit;

Commit complete.
```

Toon verdeling aan

Om aan te tonen dat de verdeling niet uniform is, de volgende query:

```
SQL>select owner, user_id, count(OBJECT_ID) amount from mytest group by
owner, user_id order by amount;

OWNER                            USER_ID     AMOUNT
------------------------------ ---------- ----------
RK                                   1704         32
SNAP                                   25         32
TQUSER                                 75         32
```

```
DS                                1533          64
ALYZE                               63          96
WEB_U                             1543         160
DIS_MGR                             86         160
OUTLN                               11         224
BN_USER                           1664         224
ITUSR                              125         416
ITBATCH                            375         416
BOM                               1506         480
BPO_OWNER                          101         576
SHUSER                            1688         608
CCM                                 80         736
APPVW                             1516         800
OHADM                               70        1152
TC_MGR                              74        1248
REPUSR                            1544        1280
TRGUSR                            1655        1280
TOCMGR                              89        1440
REPUSER_I                         1665        1440
TMDBMGR                            124        1728
SYSTEM                               5       12960
STUSR                              117       25376
MSTROST_OWNER                     1622       35008
MSTROWN                           1620       42112
PRREAL                            1453       50144
SCSMGR                            1673       67936
STATSUSR                           115       69792
SYS                                  0      131008
TOA                               1476      195456
PS_P                              1532      566528
BIES                               133      593952
```

Voor de test kiezen we de twee uitersten: user 'RK' met weinig objecten en user 'BIES' met veel.

Aanmaken indexen

```
SQL> create index rk1 on mytest (user_id) ;

Index created.

SQL> create index rk2 on mytest (object_id);

Index created.
```

Bepalen tabel statistieken met histogrammen

Het bepalen van statistieken doen we in dit geval met de optie SKEWONLY: Oracle stelt de kolommen vast om de histogrammen te verzamelen, gebaseerd op de data-distributie van de kolommen.

```
SQL> begin
  2 dbms_stats.gather_table_stats ( ownname =>'RK'
  2                     ,tabname => 'MYTEST'
  4                  ,estimate_percent => dbms_stats.auto_sample_size
  5                     ,block_sample     => true
  6                     ,method_opt       =>'for all columns size ske-
wonly'
  7                     ,degree           => 1
  8                     ,granularity      =>'default'
  9                     ,cascade  => true);
 10  end;
 11  /

PL/SQL procedure successfully completed.
```

Controle

Vervolgens moet gecontroleerd worden of de histogram-statistieken beschikbaar zijn. Een rij correspondeert met één bucket in het histogram.

```
SQL> SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
  2    FROM DBA_HISTOGRAMS
  3    WHERE TABLE_NAME ='MYTEST'
  4    AND COLUMN_NAME='USER_ID'
  5    AND OWNER = 'RK'
  6    ORDER BY ENDPOINT_NUMBER;

ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
            252              0
            454              5
            461             11
            487             83
            528             89
            546            101
            936            115
           2613            133
           3076           1453
           3233           1476
           5354           1532
           5420           1542
           5422           1543
           5441           1544
           5715           1620
           5843           1622
           5887           1627
           5894           1664
           6199           1673
           6218           1688
           6220           1704

21 rows selected.
```

Je kunt bovenstaand zien dat Oracle 21 buckets heeft gemaakt en dat de laagste user_id in de eerste bucket '0' is (SYS) en dat de hoogste user_id in de laatste bucket 1704 is (RK):

```
SQL>select username, user_id from dba_users where user_id  in (0,1704);

USERNAME                       USER_ID
------------------------------ ----------
RK                                 1704
SYS                                   0
```

Advanced SQL-tracing aanzetten

Een manier om de binds, de waits en het toegangspad te zien is door SQL-tracing aan te zetten voor het uitvoeren van de test:

```
SQL> alter session set events '10046 trace name context forever, level
12';

Session altered.
```

De test (met bind variabelen en histogrammen)

Update de tabel op die plaatsen alwaar de user-id veelvuldig voorkomt:

```
SQL> variable v_tmp number;
SQL> begin
  2   :v_tmp := 133;
  3  update mytest set object_id = object_id + 1 where user_id = :v_
tmp;
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.
```

En update de tabel op die plaatsen alwaar de user-id niet veelvuldig voorkomt:

```
SQL> begin
  2  :v_tmp := 1704;
  3  update mytest set object_id = object_id + 1 where user_id = :v_
tmp;
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.
```

Hoe zien de traces eruit

In onderstaande traces kan je zien dat voor de eerste van de bovenstaande updates gekozen wordt voor een Full Table Scan (FTS) met als gevolg een grote hoeveelheid waits, een FTS is hier gezien de hoeveelheid te verwachten 'hits' trouwens geen gekke keus van de optimizer.

```
PARSING IN CURSOR #1 len=94 dep=0 uid=1704 oct=47 lid=1704
tim=1088471939562439 hv=1027600660 ad='5d6db050'
begin
 :v_tmp := 133;
update mytest set object_id = object_id + 1 where user_id = :v_tmp;
end;
END OF STMT
PARSE #1:c=0,e=11986,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=108847193956
2429
BINDS #1:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=0
size=24 offset=0
   bfp=1103960a8 bln=22 avl=00 flg=05
=====================
PARSING IN CURSOR #12 len=64 dep=1 uid=1704 oct=6 lid=1704
tim=1088471939592828 hv=1169526826 ad='5d33a2c8'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = :B1
END OF STMT
PARSE #12:c=0,e=854,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,tim=1088471939592
823
BINDS #12:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=13 oacfl2=1
size=24 offset=0
   bfp=1103960a8 bln=22 avl=03 flg=09
   value=133
WAIT #12: nam='db file scattered read' ela= 1894 p1=4 p2=20 p3=5
...
WAIT #12: nam='db file scattered read' ela= 1703 p1=4 p2=25 p3=8
STAT #12 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #12 id=2 cnt=593952 pid=1 pos=1 obj=853868 op='TABLE ACCESS FULL
MYTEST '
```

Pas in de tweede query valt op dat nogmaals gekozen wordt voor een FTS, terwijl deze beter via de index benaderd had kunnen worden. Hier wordt zonder dat we het willen automatisch het vorige toegangspad gebruikt omdat door de binds voor de optimizer het statement hetzelfde lijkt en opnieuw gebruikt wordt om zodoende een 'parse' te voorkomen.

```
begin
:v_tmp := 1704;
update mytest set object_id = object_id + 1 where user_id = :v_tmp;
end;
END OF STMT
PARSE #1:c=0,e=35856,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=108847250668
0962
BINDS #1:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=0
size=24 offset=0
   bfp=1103960a8 bln=22 avl=03 flg=05
   value=133
=====================
PARSING IN CURSOR #12 len=64 dep=1 uid=1704 oct=6 lid=1704
tim=1088472506750518 hv=1169526826 ad='5d33a2c8'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = :B1
END OF STMT
PARSE #12:c=0,e=177,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1088472506750
513
BINDS #12:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=13 oacfl2=1
size=24 offset=0
   bfp=1103960a8 bln=22 avl=03 flg=09
   value=1704
WAIT #12: nam='db file scattered read' ela= 1351 p1=4 p2=20 p3=5
...
```

```
WAIT #12: nam='db file scattered read' ela= 1658 p1=4 p2=11969 p3=8
STAT #12 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #12 id=2 cnt=32 pid=1 pos=1 obj=853868 op='TABLE ACCESS FULL
MYTEST'
```

Geen histogrammen

Wat gebeurt er zonder histogrammen ? Eerst verwijderen we de statistieken:

```
SQL> conn rk/rk
Connected.
SQL> begin
  2     dbms_stats.delete_column_stats ( ownname =>'RK'
  3                 ,tabname => 'MYTEST'
  4                 ,colname => 'USER_ID');
  5  end;
  6  /

PL/SQL procedure successfully completed.
```

Dan tonen we aan dat er geen histogrammen meer bestaan:

```
SQL> SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
  2    FROM DBA_HISTOGRAMS
  3    WHERE TABLE_NAME ='MYTEST'
  4    AND COLUMN_NAME='USER_ID'
  5    AND OWNER = 'RK'
  6    ORDER BY ENDPOINT_NUMBER;

no rows selected

SQL> alter session set events '10046 trace name context forever, level
12';

Session altered.
```

De update wordt opnieuw gedaan, nu zonder histogrammen.

```
SQL> variable v_tmp number;
SQL> begin
  2  :v_tmp := 133;
  3  update mytest set object_id = object_id + 1 where user_id = :v_
tmp;
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.

SQL> begin
  2  :v_tmp := 1704;
  3  update mytest set object_id = object_id + 1 where user_id = :v_
tmp;
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.

SQL> exit
```

Je ziet in de onderstaande trace dat zonder histogrammen zelfs de eerste FTS (die mij in dit geval aannemelijk zou lijken) al wordt vermeden en er nu al gebruik gemaakt wordt van een index. Kennelijk wordt de optimizer nu niet 'afgeleid'.

```
PARSING IN CURSOR #1 len=93 dep=0 uid=1704 oct=47 lid=1704
tim=1088472739561519 hv=1036415784 ad='534734c8'
begin
:v_tmp := 133;
update mytest set object_id = object_id + 1 where user_id = :v_tmp;
end;
END OF STMT
PARSE #1:c=0,e=1780,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=1088472739561
511
BINDS #1:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=0
size=24 offset=0
   bfp=1103960c8 bln=22 avl=00 flg=05
====================
PARSING IN CURSOR #13 len=64 dep=1 uid=1704 oct=6 lid=1704
tim=1088472739571690 hv=1169526826 ad='5d33a2c8'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = :B1
END OF STMT
PARSE #13:c=0,e=873,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,tim=1088472739571
683
WAIT #13: nam='db file sequential read' ela= 9682 p1=4 p2=12999 p3=1
...
STAT #13 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #13 id=2 cnt=593952 pid=1 pos=1 obj=853870 op='INDEX RANGE SCAN
RK1 '
```

Ook bij de tweede update, wanneer het aantal te verwachten 'hits' nog minder voorkomt wordt nu gekozen voor een `'INDEX RANGE SCAN'`:

```
PARSING IN CURSOR #1 len=94 dep=0 uid=1704 oct=47 lid=1704
tim=1088472891210909 hv=4222785832 ad='5618f520'
begin
:v_tmp := 1704;
update mytest set object_id = object_id + 1 where user_id = :v_tmp;
end;
END OF STMT
PARSE #1:c=0,e=112415,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=10884728912
10899
BINDS #1:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=0
size=24 offset=0

   bfp=1103860c8 bln=22 avl=03 flg=05
   value=133
```

```
====================
PARSING IN CURSOR #13 len=64 dep=1 uid=1704 oct=6 lid=1704
tim=1088472891257473 hv=1169526826 ad='5d33a2c8'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = :B1
END OF STMT
PARSE #13:c=0,e=182,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1088472891257
468
BINDS #13:
 bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=13 oacfl2=1
size=24 offset=0
   bfp=1103860c8 bln=22 avl=03 flg=09
   value=1704
WAIT #13: nam='db file sequential read' ela= 8192 p1=4 p2=15974 p3=1
WAIT #13: nam='db file sequential read' ela= 5724 p1=4 p2=15973 p3=1
EXEC #13:c=10000,e=20487,p=2,cr=3,cu=39,mis=0,r=32,dep=1,og=4,tim=108847
2891278057
WAIT #1: nam='SQL*Net message to client' ela= 6 p1=1650815232 p2=1 p3=0
EXEC #1:c=30000,e=67190,p=2,cr=3,cu=39,mis=0,r=1,dep=0,og=4,tim=10884728
91278282
WAIT #1: nam='SQL*Net message from client' ela= 6391842 p1=1650815232
p2=1 p3=0
STAT #13 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #13 id=2 cnt=32 pid=1 pos=1 obj=853870 op='INDEX RANGE SCAN RK1 '
```

Test opnieuw

Nu voeren we dezelfde test uit, maar dan zonder bind variabelen en mét histogrammen. Eerst bepalen we de statistieken.

```
SQL> begin
  2          dbms_stats.delete_column_stats ( ownname =>'RK'
  3                       ,tabname => 'MYTEST'
  4                       ,colname => 'USER_ID');
  5  end;
  6  /

PL/SQL procedure successfully completed.

SQL>
SQL> begin
  2      dbms_stats.gather_table_stats ( ownname =>'RK'
  3                            ,tabname => 'MYTEST'
  4                            ,estimate_percent => dbms_stats.auto_sam-
ple_size
  5                            ,block_sample      => true
  6                            ,method_opt        =>'for all columns size
skewonly'
  7                            ,degree             => 1
  8                            ,granularity       =>'default'
  9                            ,cascade   => true);
 10      end;
SQL> /

PL/SQL procedure successfully completed.
```

Zet tracing aan:

```
SQL> alter session set events '10046 trace name context forever, level
12';

Session altered.
```

Voer de update uit, zonder bind variabelen:

```
SQL> begin
  2  update mytest set object_id = object_id + 1 where user_id = 133;
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.

SQL>  begin
  2  update mytest set object_id = object_id + 1 where user_id = 1704;
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.
```

De eerste update zonder binds kiest voor een FTS:

```
====================
PARSING IN CURSOR #19 len=63 dep=1 uid=1704 oct=6 lid=1704
tim=1088548891215037 hv=87291687 ad='5bc2a028'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = 133
END OF STMT
PARSE #19:c=0,e=3769,p=0,cr=5,cu=0,mis=1,r=0,dep=1,og=4,tim=108854889121
5032
BINDS #19:
WAIT #19: nam='db file scattered read' ela= 17894 p1=4 p2=20 p3=5
WAIT #19: nam='db file scattered read' ela= 5269 p1=4 p2=25 p3=8
...
WAIT #19: nam='db file scattered read' ela= 5269 p1=4 p2=25 p3=8
STAT #19 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #19 id=2 cnt=593952 pid=1 pos=1 obj=853868 op='TABLE ACCESS FULL
MYTEST '
====================
```

De tweede update zonder binds kiest voor een index range scan:

```
====================
PARSING IN CURSOR #19 len=64 dep=1 uid=1704 oct=6 lid=1704
tim=1088549894832344 hv=2063028559 ad='538a9e68'
UPDATE MYTEST SET OBJECT_ID = OBJECT_ID + 1 WHERE USER_ID = 1704
END OF STMT
PARSE #19:c=0,e=6484,p=0,cr=19,cu=0,mis=1,r=0,dep=1,og=4,tim=10885498948
32339
BINDS #19:
WAIT #19: nam='db file sequential read' ela= 864 p1=4 p2=11996 p3=1
...
WAIT #19: nam='db file sequential read' ela= 765 p1=4 p2=11859 p3=1
STAT #19 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #19 id=2 cnt=32 pid=1 pos=1 obj=853870 op='INDEX RANGE SCAN RK1 '
```

Samenvatting

Dat toegangspaden nogal eens wisselen mag geen nieuws zijn, maar dat men meent er goed aan te doen met binds en histogrammen en toch de doorlooptijd van een statement negatief beïnvloed is opvallend. We kunnen concluderen dat een combinatie van histogrammen en binds in dit scenario tot een foute beslissing van de optimizer leidt voor het tweede (gelijke statement). Het gebeurt in zo'n geval dat de optimizer een bepaald pad voor een query kiest dat de eerste keer verstandig is, maar de tweede keer wel eens een verkeerde beslissing kan zijn omdat de verdeling anders is. Het blijkt dat met bind variabelen en histogrammen de optimizer de tweede keer hetzelfde pad neemt, om de parse te voorkomen, terwijl het op dat moment beter was geweest voor een ander toegangspad te kiezen. Dit wordt ook wel bind variable peeking genoemd. Een belangrijk advies luidt dan ook: houd je aan de regels die gesteld worden in de Oracle Performance & Tuning Guide:

Histograms are not useful for columns with the following characteristics:
- *All predicates on the column use bind variables.*
- *The column data is uniformly distributed.*
- *The column is unique and is used only with equality predicates.*

NB. In verband met de leesbaarheid en privacy is sommige output aangepast.

René Kundersma (rkundersma@qualogy.com) is Oracle Certified Master en werkzaam als database consultant bij Qualogy Consultancy B.V.

NIEUWS

Artikelen met praktische informatie, geschreven door en bestemd voor Oracle-professionals vindt u in het Online Archief van Array Publications. Vaktijdschriften als Database Magazine, Software Release en Java Magazine hebben hun artikelenarchief online gezet. Met een heldere zoekstructuur vindt u snel wat u zoekt op www.optimize.nl.

Oracle koopt Siebel

Voor een slordige 6 miljard dollar neemt Oracle de aandelen van CRM-specialist Siebel over. "Oracle is hiermee in één klap 's werelds nummer 1 op het gebied van CRM," zegt Oracle-CEO Larry Ellison. "Met de 4000 klanten en 3,4 miljoen gebruikers van Siebel's software erbij versterken we onze koppositie als applicatievendor in Noord-Amerika en komen we dicht in de buurt van de nummer 1 van de wereld." The Board of Directors van Siebel Systems stemde voor de overname en ook groot-aandeelhouder Tom Siebel heeft gezegd dat te zullen doen. De overige aandeelhouders brengen het bod van 10,66 dollar per aandeel in een speciale vergadering ter stemming. Er is geen toestemming vereist van Oracle-aandeelhouders; de transactie is onderworpen aan diverse wettelijke regelingen, waardoor de overname pas begin 2006 zou kunnen worden geëffectueerd.

Oracle introduceert Oracle TimesTen In-memory database 6.0

Eerste productaankondiging sinds overname: uitbreiding populaire database voor real-time datamanagement. Oracle presenteert Oracle TimesTen In-Memory Database Release 6.0. Dit is de eerste product-upgrade na de acquisitie van TimesTen in juni van dit jaar. Deze versie biedt enorme verbeteringen qua opslagcapaciteit en beschikbaarheid, een groter databasegeheugen en meer dataopslag, sterkere integratie met Oracle-producten en bredere ondersteuning van standaarden als SQL en Java. Met deze nieuwe versie is Oracle de trotse eigenaar van een end-to-end datamanagement oplossing die zowel real-time, embedded en front-office-systemen omvat, als grootschalige enterprise databases en data warehouses.

Oracle presenteert Oracle Application Server 10g release 3

Op Oracle Open World, afgelopen september in San Francisco, heeft Oracle de Oracle Application Server 10g release 3 gepresenteerd. Het betreft een update van het geïntegreerde middleware-platform voor Service Oriented Architecture (SOA). Oracle Application Server 10g Release 3 helpt bedrijven met de ontwikkeling en inzet van Service Oriented applicaties binnen een grid computing-architectuur. Daarnaast ondersteunt het platform de integratie van services binnen bedrijfsprocessen en beveiligt en beheert het services, applicaties en data binnen een heterogene omgeving.