LOGIN
software

ORACLE CERTIFIED PARTNER

software

Alen Prodan

Optimizacija 11g baze uz pomoć novih alata i mogućnosti

Agenda

- Novosti u generiranju SQL trace dijagnostičkih informacija
- Novosti u CBO okruženju

  SQL Plan Management
- Novosti u upravljanju CBO statistikama

  Poboljšana AUTO sampling opcija, inkrementalne globalne statistike, pending statistike, Extended, MultiColumn i Expression statistike
- SQL Result Cache i PL/SQL function result cache
- Adaptive Cursor Sharing

  Inteligentno dijeljenje cursora
- Poboljšanja u performansama SQL operacija

  Native hash full outer join, join filter pruning
- Pitanja i odgovori

Novosti u generiranju SQL trace dijagnostičkih informacija

ADR- nova lokacija SQL trace dijagnostičkih datoteka

- Automatic Diagnostic Repository (ADR) – nova lokacija SQL trace i ostalih dijagnostičkih datoteka
- ADR je datotečni repozitorij za dijagnostičke podatke baze podataka kao što su trace datoteke, dump datoteke, alert log, itd.
- Uz pomoć sljedećeg upita pronalazimo lokaciju trace datoteke za tekuću sesiju:

```
SQL> select name, value
  2  from v$diag_info
  3  where name = 'Default Trace File';

NAME                 VALUE
-------------------- ------------------------------------------------------------
Default Trace File   c:\oracle\diag\rdbms\ora111\ora111\trace\ora111_ora_1160.trc
```

Novosti u generiranju SQL trace dijagnostičkih informacija

SQL Trace event u Oracle 11g

- Proširena sintaksa kojom uključujemo generiranje SQL trace podataka za neku sesiju ili proces
- Nove opcije omogućuju generiranje SQL trace dijagnostičkih podataka za samo jednu ili nekolicinu SQL naredbi

```
                session
alter  ———————————— set events ‘sql_trace [sql:<sql_id>|<sql_id>] ..specifikacija..’
                system
```

Novosti u generiranju SQL trace dijagnostičkih informacija

SQL Trace event u Oracle 11g

- Primjer uključivanja SQL trace opcije za jednu ili više određenih SQL naredbi:

```
SQL> select sql_id, sql_text
  2  from v$sql
  3 where sql_text like 'select * from dual'
/
SQL_ID        SQL_TEXT
------------- --------------------------------------------------
a5ks9fhw2v9s1 select * from dual

SQL> alter session set events 'sql_trace [sql:a5ks9fhw2v9s1]‘;
Session altered.
```

Novosti u generiranju SQL trace dijagnostičkih informacija

SQL Trace event u Oracle 11g

- Druga vrlo korisna opcija za sql_trace odnosi se na generiranje SQL tracea uz pomoć podataka o procesu, koristeći sljedeću sintaksu:

```
                session
alter   ———————————————   set events ‘sql_trace {process : pid = <pid>,pname = <pname>,
                system     orapid = <orapid>} ..specifikacija..’
```

Novosti u generiranju SQL trace dijagnostičkih informacija

SQL Trace event u Oracle 11g

- Moguće je dakle uključiti trace za pojedini proces koristeći podatke o process ID, imenu procesa ili orapid vrijednosti:

```
SQL> select s.sid, s.serial#, p.spid
  2  from v$session s, v$process p
  3  where s.paddr = p.addr
  4  and s.sid in (select distinct sid from v$mystat)
  5  /
       SID    SERIAL# SPID
---------- ---------- ------------------------
       459      21718 16395

SQL> alter session set events 'sql_trace {process:16395}';
Session altered.

SQL> alter session set events 'sql_trace {process:16395} off';
Session altered.
```

Novosti u generiranju SQL trace dijagnostičkih informacija

SQL Trace event u Oracle 11g

- Ostatak specifikacije izgleda kao što slijedi:

```
wait=true|false, bind=true|false, plan_stat=never|first_execution|all_executions, level=12
```

- Primjer cjelovite sintakse za uključivanje SQL_TRACE eventa:

```
alter session set events ‘sql_trace {process:16395} wait=true, bind=true,
plan_stat=all_executions,level 12’;

Session altered.
```

Novosti u CBO okruženju

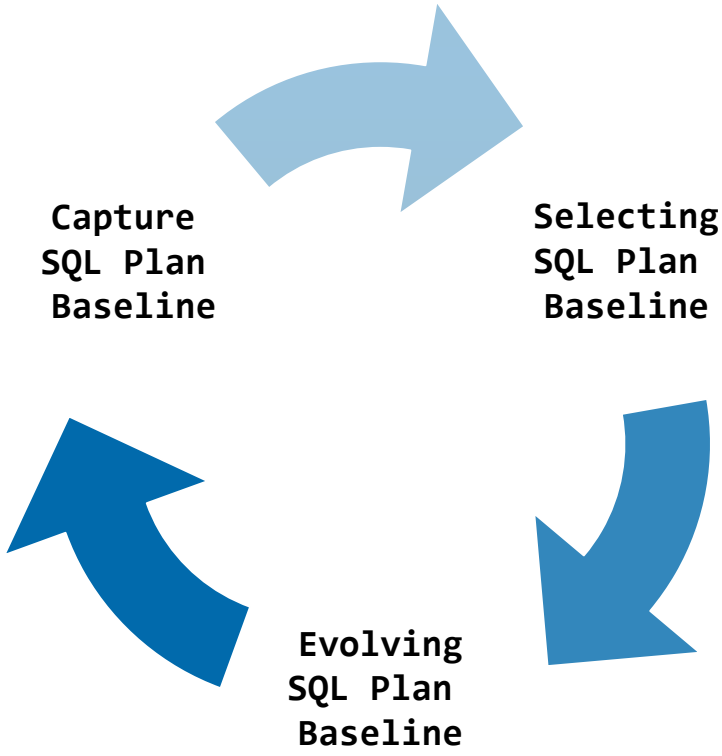
SQL Plan Management

- SQL Plan Management (SPM) novo je svojstvo 11g baze podataka
- Omogućuje upravljanje planovima izvršavanja SQL naredbi
- SPM-om je moguće prevenirati nagle padove performansi SQL naredbi koje su rezultat nenadanih promjena u njihovim planovima izvršavanja
- Takve promjene mogu nastupiti uslijed promjene verzije optimizera, CBO statistika, sistemskih postavki i sl.

Novosti u CBO okruženju

SQL Plan Management

- Upravljanje osnovicama SQL planova (SQL Plan baselines) sastoji se od 3 faze:



Novosti u CBO okruženju

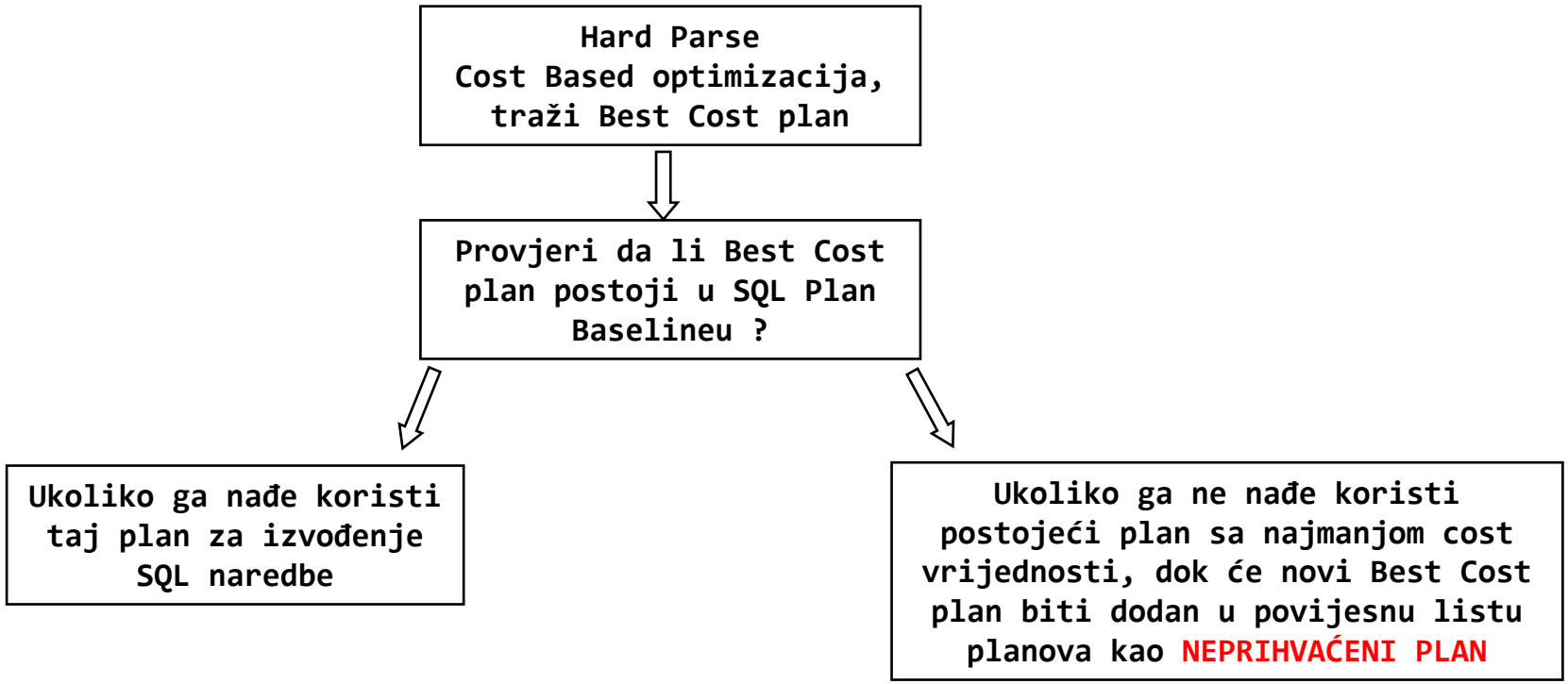
SQL Plan Management (Capturing SQL Plan Baselines)

- U toj fazi Oracle baza podataka bilježi informacije o izvođenju SQL naredbi kako bi detektirala promjene planova
- Baza podataka čuva povijest planova izvršavanja za svaku pojedinu SQL naredbu i to samo za one naredbe čije se izvršavanje ponavlja više puta
- Za automatsku pohranu planova izvršavanja potrebno je podesiti vrijednosti **`OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE`** inicijalizacijskog parametra na razini instance ili sesije

Novosti u CBO okruženju

SQL Plan Management (Selecting SQL Plan Baselines)

- U fazi izbora plana, Oracle vrši detekciju promjene planova na temelju pohranjenje povijesti planova izvršavanja, te izabire planove kako bi se izbjegli potencijalni problemi sa performansama



Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Kada optimizer pronađe novi plan za neku SQL naredbu, plan se dodaje u povijesnu listu planova kao neprihvaćeni
- Nakon što se za novododani neprihvaćeni plan utvrdi da ne uzrokuje probleme sa performansama, njegov se status mijenja u prihvaćen
- Ta se provjera vrši usporedbom njegovih performansi sa planom koji se već nalazi unutar SQL Plan Baselinea, garantirajući pritom bolje performanse
- Evolviranje planova moguće je izvršiti učitavanjem planova iz cursor cachea ili SQL Tuning seta (ručno), ili pak pozivom na **`DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE`** funkciju

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Prvi korak: priprema tablice koja sadrži 100.000 redaka od kojih 10.000 sadrži vrijednost 1 u X koloni:

```
SQL> create table t
  2  as
  3  select
  4  trunc((mod(rownum, 10)+1)/10) x, rpad('x', 100, 'x') vc
  5  from dual
  6  connect by level <= 100000
  7  order by 1
  8  /

Table created.

SQL> create index t_idx on t(x);

Index created.
```

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Nad tablicom su prikupljene statistike, ali bez histograma, pa je selektivnost za kolonu X izračunata kao 1/NDV, odnosno ½=0.5. Zbog krive pretpostavke o selektivnosti, optimizer se odlučuje za FTS:

```
SQL> set autot traceonly
SQL> select * from t where x = 1;

10000 rows selected.

Execution Plan
----------------------------------------------------------
Plan hash value: 1601196873
--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 50000 |  5029K|   424   (1)| 00:00:06 |
|*  1 |  TABLE ACCESS FULL| T    | 50000 |  5029K|   424   (1)| 00:00:06 |
--------------------------------------------------------------------------------
Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_441126ef94ecae5c" used for this statement
```

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Plan sa full table scanom, tako postaje prvi plan za taj upit koji je prihvaćen u SQL Plan Baseline:

```
SQL> select sql_handle, sql_text, accepted
     from dba_sql_plan_baselines;

SQL_HANDLE                     SQL_TEXT                       ACC
------------------------------ ------------------------------ ---
SYS_SQL_a338a0b4441126ef       select * from t where x = 1    YES
```

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Sada ponavljamo izvršavanje istog upita, ali ovoga puta sa prikupljenim histogramom nad X kolonom:

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1601196873
--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 10000 |  1005K|   424   (1)| 00:00:06 |
|*  1 |  TABLE ACCESS FULL| T    | 10000 |  1005K|   424   (1)| 00:00:06 |
--------------------------------------------------------------------------------
Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_441126ef94ecae5c" used for this statement
```

- Ovoga puta optimizer je dobro procjenio broj redaka koje upit vraća (10000), međutim plan je ostao smrznut zbog SQL Plan Managementa

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Iz **DBA_SQL_PLAN_MANAGEMENT** viewa vidljivo da je optimizer našao novi best-cost plan, koji je pohranjen kao neprihvaćen u listu planova:

```
SQL_HANDLE                     SQL_TEXT                       ACC
------------------------------ ------------------------------ ---
SYS_SQL_a338a0b4441126ef       select * from t where x = 1    YES
SYS_SQL_a338a0b4441126ef       select * from t where x = 1    NO
```

- Da bi novi plan postao prihvaćen, potrebno je provjeriti kakve su njegove performanse pozivom na **DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE** funkciju

```
SQL> select dbms_spm.evolve_sql_plan_baseline('SYS_SQL_a338a0b4441126ef') from dual;
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE('SYS_SQL_A338A0B4441126EF')
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
                         Evolve SQL Plan Baseline Report
--------------------------------------------------------------------------------
Inputs:
-------
  SQL_HANDLE = SYS_SQL_a338a0b4441126ef
  PLAN_NAME  =
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = YES
Plan: SYS_SQL_PLAN_441126efae82cf72
-----------------------------------
  Plan was verified: Time used ,078 seconds.
  Passed performance criterion: Compound improvement ratio >= 8,76
  Plan was changed to an accepted plan.
                     Baseline Plan      Test Plan     Improv. Ratio
                     -------------      ---------     -------------
  Execution Status:        COMPLETE       COMPLETE
  Rows Processed:             10000          10000
  Elapsed Time(ms):              19              0
  CPU Time(ms):                  31              0
  Buffer Gets:                 1521            174              8,74
  Disk Reads:                     0              0
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE('SYS_SQL_A338A0B4441126EF')
--------------------------------------------------------------------------------
  Direct Writes:                  0              0
  Fetches:                        0              0
  Executions:                     1              1
--------------------------------------------------------------------------------
                                  Report Summary
--------------------------------------------------------------------------------
Number of SQL plan baselines verified: 1.
Number of SQL plan baselines evolved: 1.
```

Novosti u CBO okruženju

SQL Plan Management (Evolving SQL Plan Baselines)

- Iz generiranog izvještaja vidimo da je novi plan prihvaćen, a to možemo provjeriti ponovnim izvršavanjem testnog SQL upita:

```
SQL> select sql_handle, sql_text, accepted, buffer_gets from dba_sql_plan_baselines;

SQL_HANDLE                     SQL_TEXT                       ACC BUFFER_GETS
------------------------------ ------------------------------ --- -----------
SYS_SQL_a338a0b4441126ef       select * from t where x = 1    YES           0
SYS_SQL_a338a0b4441126ef       select * from t where x = 1    YES           0

Execution Plan
----------------------------------------------------------
Plan hash value: 470836197
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       | 10000 |  1005K|   171   (0)| 00:00:03 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T     | 10000 |  1005K|   171   (0)| 00:00:03 |
|*  2 |   INDEX RANGE SCAN           | T_IDX | 10000 |       |    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Novosti u upravljanju CBO statistikama

Poboljšana AUTO sampling opcija

- Jedna od najvažnijih statistika koju optimizer koristi je broj distinct (različitih) vrijednosti u kolonama tablice (NDV – number of distinct values)
- U situaciji kada ne postoje histogrami, Oracle koristi NDV za izračun selektivnosti predikata nad nekom kolonom tablice
- Za dobivanje točnih i deterministčkih vrijednosti NDV u kolonama (c1, c2) tablice (T), moguće je izvršiti SQL:

```
select count(distinct C1), count(distinct C2) from T;
```

- Prednost takvog načina je točnost, dok je u slučaju velike tablice ta metoda jako spora i zahtjeva masivne sort operacije i korištenje TEMP tablespacea

Novosti u upravljanju CBO statistikama

Poboljšana AUTO sampling opcija

- U verziji 10g, DBMS_STATS koristi metodu slučajnog sampliranja uzorka kako bi izračunao NDV za neku tablicu
- Sampliranje inherentno podrazumijeva mogućnost greške u procjeni i ne garantira determinističnost rezultata
- Važno je znati da postoji velika razlika u verziji baze 10g i 11g, kada je riječ o veličini uzorka (sample size) kojega sustav analizira prilikom korištenja `DBMS_STATS.AUTO_SAMPLE_SIZE` opcije:
- 10g: **`DBMS_STATS.AUTO_SAMPLE_SIZE`** koristi vrlo malenu vrijednost uzorka
- 11g: **`DBMS_STATS.AUTO_SAMPLE_SIZE`** koristi veliki uzorak ~ 100%

Novosti u upravljanju CBO statistikama

Poboljšana AUTO sampling opcija

- BRZINA IZVOĐENJA: U 11g, unatoč činjenici da se analizira uzorak od 100% podataka, proces prikupljanja statistika je izuzetno brz iz razloga što sustav koristi novi hashing algoritam za izračun NDV statistika, umjesto sortiranja
- TOČNOST: Novi algoritam skenira cijelu tablicu, umjesto da radi nad sampliranim uzorkom
- DETERMINISTIČNOST: Dobivena vrijednost NDV-a deterministička je, što dovodi do povećanja stabilnosti planova izvršavanja

Novosti u upravljanju CBO statistikama

11g: Vrijeme trajanja izračuna NDV statistika (sample vs. hash-based)

	Trajanje u sekundama
Oracle Database 10g R2 (10 %)	84
Oracle Database 10g R2 (100 %)	696
Oracle Database 11g R1	78

```
10g R2 (10%) : dbms_stats.gather_table_stats(user, 'FACT', estimate_percent=>10);
10g R2 (100%): dbms_stats.gather_table_stats(user, 'FACT', estimate_percent=>100);
11g R1       : dbms_stats.gather_table_stats(user, 'FACT');
```

Novosti u upravljanju CBO statistikama

11g: Preciznost izračuna NDV statistika

Stupac	Oracle Database 10g R2 (10%)	Oracle Database 11g
C1	70,06 %	99,89 %
C2	98,08 %	100,00 %
C6	90,00 %	190,00 %
C7	85,40 %	100,00 %
C8	89,61 %	100,00 %
C10	92,11 %	100,00 %
C11	60,59 %	100,18 %
C12	61,15 %	100,00 %
C13	30,07 %	99,10 %
C14	28,15 %	100,22 %
C15	28,79 %	100,00 %
C16	46,35 %	100,41 %
C17	47,15 %	100,00 %
C18	46,02 %	99,37 %
C19	25,11 %	99,11 %
C20	49,19 %	100,00 %
C21	42,36 %	100,00 %

Napomena: izostavljeni stupci gdje sve vrijednosti iznose 100,00 %

Novosti u upravljanju CBO statistikama

Inkrementalno održavanje globalnih statistika tablice (particionizirane tablice)

- Za particionizirane tablice statistike se prikupljaju na dvije razine: na razini particija i na razini tablice (globalne statistike)
- Oba tipa statistika važna su za optimizer
- Particijske statistike važne su za funkcioniranje mehanizma za eliminaciju particija
- Globalne statistike važne su za određivanje selektivnosti na razini cijele tablice

Novosti u upravljanju CBO statistikama

Inkrementalno održavanje globalnih statistika tablice (particionizirane tablice)

- Većina globalnih statistika može se izračunati iz statistika na razini particija: broj redova u tablici, broj blokova, prosječna veličina retka u byteovima, broj null vrijednosti u kolonama, min i max vrijednosti u nekoj koloni i sl.
- Npr. broj NULL vrijednosti u nekoj koloni predstavlja sumu broja NULL vrijednosti u svakoj particiji.
- Jedina iznimka navedenom pravilu je globalna NDV statistika za neku kolonu

Novosti u upravljanju CBO statistikama

Inkrementalno održavanje globalnih statistika tablice (particionizirane tablice)

- Pretpostavimo da tablica T ima dvije particije: P1 sa 10 distinct vrijednosti i P2 sa 20 distinct vrijednosti u koloni C. Broj distinct vrijednosti u koloni C može varirati u rasponu od 20 do 30 !
- 10g: koristi metodu duplog prolaza za izračun NDV. U prvom prolazu skenira particije kako bi izračunao partition statistike, dok u drugom prolazu ponovno skenira sve particije (cijelu tablicu) kako bi izračunao globalne statistike
- 11g: moguće je dobiti točne NDV statistike kako na razini particije, tako i na globalnoj razini u samo jednom koraku.

Novosti u upravljanju CBO statistikama

Inkrementalno održavanje globalnih statistika tablice (particionizirane tablice)

- 11g koristi novi hash bazirani algoritam, koji izračunava statistike na razini particija, te za svaku particiju generira svojevrsni “međuzbroj”.
- Kada nastupi promjena u nekoj od particija, 11g prikuplja statistike samo za tu particiju
- Kako međuzbrojevi ne-modificiranih particija ostaju nepromjenjeni, globalne se statistike mogu derivirati na temelju novog međuzbroja particije u kojoj su nastupile promjene i starih međuzbrojeva.
- Takvo inkrementalno održavanje NDV globalnih statistika omogućuje bazi podataka osvježavanje globalnih statistika bez potrebe za ponovnim skeniranjem cijele tablice, te time značajno poboljšava performanse prikupljanja statistika.

Novosti u upravljanju CBO statistikama

Objavljene (published) i neobjavljene (pending) statistike

- Prilikom prikupljanja statistika, moguće je odrediti da statistike budu objavljene na kraju prikupljanja ili da se one pohrane u data dictionary kao neobjavljene (pending)
- To nam omogućava validaciju novih statistika, te njihovo objavljivanje tek nakon što se utvrdi da su one zadovoljavajuće
- Prvi primjer prikazuje način na koji je moguće provjeriti da li će statistike biti objavljene po završetku prikupljanja:

```
select dbms_stats.get_prefs('PUBLISH') from dual;
```

- Za korištenje neobjavljenih statistika koristimo sljedeći inicijalizacijski parametar na razini sesije ili instance, dok pozivom na dbms_stats.publish_pending_statistics objavljujemo statistike:

```
alter session set optimizer_pending_statistics = TRUE;

exec dbms_stats.publish_pending_statistics(null, null);
```

Novosti u upravljanju CBO statistikama

Extended statistike – MultiColumn statistike

- 11g: kada se više kolona iz jedne tablice koristi zajedno u WHERE uvjetu, veze između tih klona mogu značajno utjecati na kombiniranu selektivnost na razini te grupe kolona.
- Pretpostavimo da imamo tablicu sa sljedećom strukturom podataka:

```
SQL> select mjesec, naziv, count(*) from t1
  2  group by mjesec, naziv order by mjesec
  3  /
    MJESEC NAZIV      COUNT(*)
---------- -------- ----------
         1 Sijecanj       1000
         2 Veljaca        1000
         3 Ozujak         1000
         4 Travanj        1000
         5 Svibanj        1000
         6 Lipanj         1000
         7 Srpanj         1000
         8 Kolovoz        1000
         9 Rujan          1000
        10 Listopad       1000
        11 Studeni        1000
        12 Prosinac       1000
```

Novosti u upravljanju CBO statistikama

Extended statistike – MultiColumn statistike

- Ukoliko postavimo upit u kojem tražimo koliko postoji redaka koji sadrže u koloni MJESEC vrijednost “1”, a u koloni NAZIV vrijednost “SIJEČANJ”, optimizer bi trebao prikazati vrijednost 1000.
- Iz sljedećeg primjera vidljivo je da optimizer ne procjenjuje dobro selektivnost takvog upita. SQL vraća 1000 redaka, ali optimizer procjenjuje samo 83 retka !!!

```
SQL> select count(*)
  2  from t1
  3  where mjesec = 1 and naziv = 'Sijecanj';
  COUNT(*)
----------
      1000
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    10 |    10   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |      |     1 |    10 |            |          |
|*  2 |   TABLE ACCESS FULL| T1   |    83 |   830 |    10   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Novosti u upravljanju CBO statistikama

Extended statistike – MultiColumn statistike

- Kako je nastalo takvo odstupanje ?
- Optimizer zna da u svakoj od te dvije kolone postoji 12 distinct vrijednosti, a tablica sadrži ukupno 12.000 redaka. Selektivnost na razini jedne kolone izračunava se kao **`1/NDV = 1/12 = 0,833333333`**
- **`1 / 12 * 12.000 = 1.000`**, što predstavlja točnu selektivnost za jednu kolonu
- Kombinirana AND selektivnost za dvije kolone izračunava se kao **`(1/12 * 1/12) * 12.000 = 83`**, iz čega i proizlazi greška.

Novosti u upravljanju CBO statistikama
Extended statistike – MultiColumn statistike

- 11g nudi rješenje u obliku MultiColumn statistika.
- Te ekstenzije označuju optimizeru da posebno prikupi statistike za kombinirane vrijednosti kolona definirane kroz te ekstenzije

```
SQL> variable cg varchar2(100);
SQL> begin
  2   :cg := dbms_stats.create_extended_stats(
  3    user, 'T1', '(mjesec, naziv)');
  4  end;
  5  /

PL/SQL procedure successfully completed.
SQL> print cg

CG
--------------------------------------------------------------------------------
-
SYS_STU#946WW5EBA7Z7G2MJTNHU_F

SQL> select * from user_stat_extensions;

TABLE_NAME EXTENSION_NAME                    EXTENSION            CREATO DRO
---------- --------------------------------- -------------------- ------ ---
T1         SYS_STU#946WW5EBA7Z7G2MJTNHU_F    ("MJESEC","NAZIV")   USER   YES
```

Novosti u upravljanju CBO statistikama

Extended statistike – MultiColumn statistike

- Nakon ponovnog prikupljanja statistika nad T1 tablicom, prikupiti će se i statistike za grupu kolona ('MJESEC', 'NAZIV') koje će reflektirati njihovu međuovisnost:

```
SQL> select count(*)
  2  from t1
  3  where mjesec = 1
  4  and   naziv = 'Sijecanj';

  COUNT(*)
----------
      1000

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    11 |    10   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |      |     1 |    11 |            |          |
|*  2 |   TABLE ACCESS FULL| T1   |  1000 | 11000 |    10   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Novosti u upravljanju CBO statistikama

Extended statistike – Expression statistike

- Kada je aplicirana funkcija nad kolonom u WHERE uvjetu (function(c1)=x), optimizer ne može znati kako to utječe na selektivnost
- Prikupljajući statistike nad rezultatom kojega generira funkcija nad tom kolonom, optimizer može preciznije odrediti selektivnost

```
select dbms_stats.create_extended_stats(null, 'T1', '(lower(naziv))') from dual;
```

SQL Result Cache i PL/SQL Function Result Cache

SQL Result Cache

- 11g Result Cache komponenta uvedena je kao dio memorijske arhitekture 11g servera i sastavni je dio shared pool-a
- Result Cahe se sastoji od SQL Query Result Cachea, te PL/SQL Function Result Cachea
- Rezultati SQL upita ili dijelova (blokova) složenijih upita mogu biti pohranjeni u memoriju SQL query result cachea, a baza podataka može koristiti te pohranjene rezultate iz memorije prilikom budućih izvršavanja istih upita

SQL Result Cache i PL/SQL Function Result Cache

- Najpraktičnije je uključiti cachiranje rezultata SQL upita korištenjem **result_cache** hinta kako slijedi:

```
SQL> select /*+ result_cache */ trunc(dat, 'yyyy'), sum(kol), avg(kol)
  2  from fact
  3  group by trunc(dat, 'yyyy')
  4  /
....
Elapsed: 00:00:00.01
Execution Plan
----------------------------------------------------------
Plan hash value: 424147104
-------------------------------------------------------------------------------------------------
| Id  | Operation           | Name                         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                              |   365 |  5840 |  2756   (5)| 00:00:34 |
|   1 |  RESULT CACHE       | 58zud2xvf13g2fqzk1h2fs6ydh |       |       |            |          |
|   2 |   HASH GROUP BY     |                              |   365 |  5840 |  2756   (5)| 00:00:34 |
|   3 |    TABLE ACCESS FULL| FACT                         |  2106K|    32M|  2653   (1)| 00:00:32 |
-------------------------------------------------------------------------------------------------
Result Cache Information (identified by operation id):
------------------------------------------------------
   1 - column-count=3; dependencies=(ALEN.FACT); name="select /*+ result_cache */ trunc(dat,
'yyyy'), sum(kol), avg(kol)
from fact
group by trunc(dat, 'yyyy')"
```

SQL Result Cache i PL/SQL Function Result Cache

PL/SQL Function Result Cache

- Sljedeći primjer prikazuje kako se uključuje PL/SQL function result cache

```
SQL> create function f
  2  return number result_cache
  3  is
  4  begin
  5   dbms_lock.sleep(10);
  6   return 0;
  7  end;
  8  /
Function created.
SQL> select f from dual;
         F
----------
         0
Elapsed: 00:00:10.06
SQL> /
         F
----------
         0
Elapsed: 00:00:00.00  <<<= Result Cache na djelu
```

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- Bind varijable su od esencijalne važnosti za optimalno upravljanje cursorima
- Ne korištenje bind varijabli uzrokuje neprestano hard parsanje i rekompilaciju gotovo istovjetnih SQL naredbi, a to je posebno važno u OLTP sustavima

```
select c1 from T where c2 = 1;
select c1 from T where c2 = 2;
```

- Gornji SQL predstavlja primjer istovjetnog cursora, a jedina je razlika u vrijednosti filtera WHERE uvjeta
- Kako bi izbjegli nepotrebno parsanje umjesto literala u WHERE uvjetu koristimo bind varijablu:

```
select c1 from T where c2 = :x
```

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- Iako dobro rješenje za skalabilnost sustava, bind varijable skrivaju stvarnu vrijednost korištenih filtera od optimizera
- Stoga je u Oracle 9i uveden bind variable peeking mehanizam, koji omogućuje optimizeru “uvid” u stvarnu vrijednost bind varijable prilikom prvog poziva (hard parse) nekog cursora
- To omogućuje određivanje selektivnosti kao da su korišteni literali
- Bind variable peeking ima problem ukoliko distribucija vrijednosti u koloni nije ravnomjerna – drugim rječima, plan je optimiziran za onu vrijednost koju je sadržavala bind varijabla prilikom prvog (hard) parsanja

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- 11g: optimizer je poboljšan te je moguće korištenje više različitih planova izvršavanja za jednu SQL naredbu koja koristi bind varijable
- To nam osigurava da će se koristiti najbolji plan izvršavanja ovisno o vrijednosti koju sadrži bind varijabla

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- Pretpostavimo da imamo sljedeću tablicu, te prvo izvođenje cursora radimo za X = 1:

```
SQL> select x, count(*)
    from t2
    group by rollup(x);

         X   COUNT(*)
---------- ----------
         1  2.000.000
         2     20.000
         3        100
         5          1
```

- Prvo izvođenje radimo za vrijednost X = 1:

```
variable x number;
exec :x := 1;
select * from t2 where x = :x;
```

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- Iz V$SQL viewa vidimo da je cursor koji ima child_number = 0 bind sensitive, shareable, ali NIJE bind aware.
- Ukoliko postoje histogrami nad kolonama iz WHERE uvjeta, prilikom bind variable peekinga kod prvog izvođenja takav cursor automatski postaje bind sensitive, što znači da sustav prati da li promjena vrijednosti bind varijable utječe na promjene u performansama

```
select sql_id, child_number, is_bind_sensitive, is_bind_aware, is_shareable,
executions, buffer_gets, rows_processed from v$sql
where sql_text like 'select * from%t2%'
/
SQL_ID        CHILD_NUMBER I I I EXECUTIONS BUFFER_GETS ROWS_PROCESSED
------------- ------------ - - - ---------- ----------- --------------
5vx71m5pfh346            0 Y N Y          1      158536        2000000
```

- Iz V$SQL_PLAN viewa vidimo da je za child_number (0) optimizer odradtio full table scan:

```
select sql_id, child_number, operation
from v$sql_plan
where sql_id = '5vx71m5pfh346'
and child_number = 0;
SQL_ID        CHILD_NUMBER OPERATION
------------- ------------ ----------------------------
5vx71m5pfh346            0 SELECT STATEMENT
5vx71m5pfh346            0 TABLE ACCESS
```

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- U drugom izvođenju u bind varijablu :X stavljamo vrijednost 5, te zatim provjerimo da li je nastupila promjena V$SQL viewu:

```
SQL_ID        CHILD_NUMBER I I I EXECUTIONS BUFFER_GETS ROWS_PROCESSED
------------- ------------ - - - ---------- ----------- --------------
5vx71m5pfh346             0 Y N Y          2      183689        2000001
```

- Iz kolone executions, vidljivo je da se je cursor još jednom izvršio, ali bez promjene plana izvršavanja.
- Ako pokušamo ponovnog izvršiti isti SQL, vidjeti ćemo da je došlo do promjene u V$SQL viewu:

```
SQL_ID        CHILD_NUMBER I I I EXECUTIONS BUFFER_GETS ROWS_PROCESSED
------------- ------------ - - - ---------- ----------- --------------
5vx71m5pfh346             0 Y N Y          2      183689        2000001
5vx71m5pfh346             1 Y Y Y          1           4              1
```

- Prilikom prvog izvođenja SQL sa bind varijablom X=5, Oracle je detektirao da postoji razlika u resursima potrebnim za izvršavanje upita, te je time cursor postao bind aware, što je kod drugog izvođenja SQL-a sa bind X=5 rezultiralo otvaranjem novog child cursora, sa potpuno novim planom izvršavanja, koji umjesto FTS koristi index access path

Adaptive Cursor Sharing

Inteligento dijeljenje cursora

- Iz V$SQL_PLAN vidljivo je da za child_number 1 umjesto FTS, Oracle korsti INDEX access path za ulazak u tablicu:

```
SQL_ID        CHILD_NUMBER OPERATION
------------- ------------ ------------------------------
5vx71m5pfh346            1 SELECT STATEMENT
5vx71m5pfh346            1 TABLE ACCESS
5vx71m5pfh346            1 INDEX
```

- Što se događa ako ponovno pokušamo izvršiti isti SQL, ali sa bind varijablom X = 1 ?

```
SQL_ID        CHILD_NUMBER I I I EXECUTIONS BUFFER_GETS ROWS_PROCESSED
------------- ------------ - - - ---------- ----------- --------------
5vx71m5pfh346            0 Y N N          2      183689        2000001
5vx71m5pfh346            1 Y Y Y          1           4              1
5vx71m5pfh346            2 Y Y Y          1      158487        2000000
```

- Iz V$SQL vidljivo je da je Oracle otvorio novi child (2), umjesto da je iskoristio početni (0), iz razloga što je u međuvremenu cursor postao bind aware, pa će stari non-bind aware child biti LRU algoritmom izbačen iz library cachea

Poboljšanja u performansama SQL operacija

Native Hash Full Outer Join

- U 11g dolazi podrška za nativni hash full outer join
- Novi mehanizam moguće je koristiti samo za equi-joinove
- Novi mehizam zahtjeva korištenje ANSI join sintakse (FULL OUTER JOIN) umjesto tradicionalne Oracle sintakse (+)
- Novi mehanizam je manje zahtjevan za resurse jer može izvršiti full outer join u jednoj operaciji, dok se je stari mehanizam izvršavao kao unija običnog outer joina (lijevi ili desni), te dodatnog hash anti joina, pa je zahtjevao dupli prolaz kroz tablice

Poboljšanja u performansama SQL operacija

Native Hash Full Outer Join

- Novu funkcionalnost moguće je kontrolirati skrivenim inicijalizacijskim parametrom **_optimizer_native_full_outer_join** koja ima vrijednosti **choose**, **force** i **off**, a default vrijednost je **force**. Slijedi primjer iz kojega je vidljiv dupli prolaz kroz tablice outer join + anti-hash:

```
SQL>  select
  2  /*+ opt_param('_optimizer_native_full_outer_join', 'off') */
  3  count(*)
  4  from t1
  5  full outer join t2
  6* on t1.x = t2.x;
---------------------------------------
| Id  | Operation                | Name |
---------------------------------------
|   0 | SELECT STATEMENT         |      |
|   1 |  SORT AGGREGATE          |      |
|   2 |   VIEW                   |      |
|   3 |    UNION-ALL             |      |
|*  4 |     HASH JOIN OUTER      |      |
|   5 |      TABLE ACCESS FULL   | T1   |
|   6 |      PARTITION RANGE ALL |      |
|   7 |       TABLE ACCESS FULL  | T2   |
|*  8 |     HASH JOIN RIGHT ANTI |      |
|   9 |      TABLE ACCESS FULL   | T1   |
|  10 |      PARTITION RANGE ALL |      |
|  11 |       TABLE ACCESS FULL  | T2   |
---------------------------------------
```

Poboljšanja u performansama SQL operacija

Native Hash Full Outer Join

- Primjer plana izvršavanja novog nativnog FULL OUTER JOIN mehanizma (jednostruki prolaz kroz tablice):

```
SQL> select count(*)
  2  from t1
  3  full outer join t2
  4* on t1.x = t2.x;
-------------------------------------------------
| Id  | Operation               | Name     | Rows  |
-------------------------------------------------
|   0 | SELECT STATEMENT        |          |     1 |
|   1 |  SORT AGGREGATE         |          |     1 |
|   2 |   VIEW                  | VW_FOJ_0 |    10G|
|*  3 |    HASH JOIN FULL OUTER |          |    10G|
|   4 |     TABLE ACCESS FULL   | T1       |   100K|
|   5 |     PARTITION RANGE ALL |          |  1000K|
|   6 |      TABLE ACCESS FULL  | T2       |  1000K|
-------------------------------------------------
```

Poboljšanja u performansama SQL operacija

Join Filter Pruning

- Jedna od temeljnih prednosti partitioning opcije odnosi se na partition pruning mehanizam, odnosno mogućnost eliminacije particija
- Optimizer može izvršiti partition pruning koristeći literale, bind varijable ili informacije iz join uvjeta
- Sve do Oracle 10g Release 2, kod hash i merge joinova optimizer koristi subquery pruning kako bi mogao točno odrediti koje particije treba eliminirati

Poboljšanja u performansama SQL operacija

Join Filter Pruning

- Primjer: tablica T1 je obična tablica, dok je T2 particionizirana. Upit radi hash-join:

```
select /*+ use_hash(a b) */
count(*)
from t1 a, t2 b
where a.x = b.x
and a.vc = 'y'
```

- Kako bi optimizer mogao izvršiti eliminaciju particija kod korištenja hash join mehanizma, on se mora osloniti na subquery partition pruning kako bi odredio koje particije sadrže željene podatke, a to ima smisla jedino ukoliko je dodatni trošak subqueryija zanemariv u odnosu na uštede koje se dobiju eliminacijom particija

- Optimizer rekurzivno izvodi upit koji vraća brojeve particija koje sadrže željene podatke uz pomoć **`tbl$or$idx$part$num`** kao u primjeru:

```
SELECT distinct TBL$OR$IDX$PART$NUM("T2", 0, 1, 0, "X")
FROM
 (SELECT "A"."X" "X" FROM "T1" "A" WHERE "A"."VC"='y') ORDER BY 1
```

Poboljšanja u performansama SQL operacija

Join Filter Pruning

- 11g pruža novi mehanizam eliminiranja particija koji se naziva join-filter pruning, a temelji se na bloom filteru
- Bloom filter predstavlja memorijsku podatkovnu strukturu koja se koristi za testiranje da li određeni element pripada nekom skupu koristeći više hash funkcija nad jednim arrayem bitova

```
SQL> select /*+ use_hash(a b) */
  2  count(*)
  3  from t1 a, t2 b
  4  where a.x = b.x
  5* and a.vc = 'y'
-------------------------------------------------------------------------
| Id  | Operation                          | Name    |  Pstart| Pstop |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |         |        |       |
|   1 |  SORT AGGREGATE                    |         |        |       |
|*  2 |   HASH JOIN                        |         |        |       |
|   3 |    PART JOIN FILTER CREATE         | :BF0000 |        |       |
|*  4 |     TABLE ACCESS FULL              | T1      |        |       |
|   5 |    PARTITION RANGE JOIN-FILTER     |         |:BF0000 |:BF0000|
|   6 |     TABLE ACCESS FULL              | T2      |:BF0000 |:BF0000|
-------------------------------------------------------------------------
```

LOGIN 14

Pitanja i Odgovori

software