What is new in the Oracle 9*i* CBO

Wolfgang Breitling

breitliw@centrexcc.com

Centrex Consulting Corporation

www.centrexcc.com

Who am I

OCP certified DBA - 7, 8, 8*i*, 9*i*

Independent consultant since 1996
specializing in Oracle and Peoplesoft setup, administration, and performance tuning

20+ years in database management
DL/1, IMS, ADABAS, SQL/DS, DB2

Oracle since 1993 (7.0.12)

Mathematics major at University of Stuttgart

Topics

- init.ora parameters changes
- Changes to Explain Plan
- New base access paths
- Histograms and bind value peeking
- New cost model and system statistics
- Dynamic sampling
- sql_trace and tkprof changes

init.ora parameter changes

Obsoleted parameters

	8.1.7	9.0	9.2
OPTIMIZER_PERCENT_PARALLEL	0		
_OPTIMIZER_PERCENT_PARALLEL		101	101

SORT_AREA_SIZE and HASH_AREA_SIZE replaced by PGA_AGGREGATE_TARGET

retained only for backward compatibility

init.ora parameter changes

Existing parameters with changed default values

	8.1.7	9.0	9.2
OPTIMIZER_MAX_PERMUTATIONS	80000	2000	2000
ALWAYS_ANTI_JOIN	nested_loops	choose	choose
ALWAYS_SEMI_JOIN	standard	choose	choose

init.ora parameter changes

Existing parameters with changed default values

	8.1.7	9.0	9.2
_B_TREE_BITMAP_PLANS	false	true	true
_COMPLEX_VIEW_MERGING	false	true	true
_INDEX_JOIN_ENABLED	false	true	true
_NEW_INITIAL_JOIN_ORDERS	false	true	true
_OR_EXPAND_NVL_PREDICATE	false	true	true
_ORDERED_NESTED_LOOP	false	true	true
_PUSH_JOIN_PREDICATE	false	true	true
_PUSH_JOIN_UNION_VIEW	false	true	true
_TABLE_SCAN_COST_PLUS_ONE	false	false	true
_UNNEST_SUBQUERY	false	true	true
_USE_COLUMN_STATS_FOR_FUNCTION	false	true	true

init.ora parameter changes

New parameters

	9.0	9.2
_SYSTEM_INDEX_CACHING	0	0
_OPTIMIZER_COST_MODEL	choose	choose
_GSETS_ALWAYS_USE_TEMPTABLES	false	false
_NEW_SORT_COST_ESTIMATE	true	true
_GS_ANTI_SEMI_JOIN_ALLOWED	true	true
_CPU_TO_IO	0	0
_PRED_MOVE_AROUND	true	true
_DYN_SEL_EST_ON	false	
_DYN_SEL_EST_NUM_BLOCKS	30	
OPTIMIZER_DYNAMIC_SAMPLING		1
_OPTIMIZER_DYN_SMP_BLKS		32

New Plan_table columns

CPU_COST

IO_COST

TEMP_SPACE

ACCESS_PREDICATES

FILTER_PREDICATES

Remember to create a new plan table after upgrading (?/rdbms/admin/utlxplan.sql)

New Explain Package

```
SELECT * FROM table
    (DBMS_XPLAN.DISPLAY(table, statement_id, format));
```

where

table	table name where the plan is stored. (default PLAN_TABLE)
statement_id	statement_id of the plan to be displayed. (default NULL)
format	the level of detail for the plan.

- BASIC
- TYPICAL
- ALL
- SERIAL

"New" Base Access Path

INDEX_JOIN

Technically not a new access path as it existed in 8*i*, but there it was disabled by default
(_INDEX_JOIN_ENABLED = false).

In 9*i* it is now enabled
(_INDEX_JOIN_ENABLED = true).

Index Join - Example

Name	Null?	Type
N1		NUMBER
N2		NUMBER
N3		NUMBER
N4		NUMBER
N5		NUMBER
C1		VARCHAR2(30)
C2		VARCHAR2(30)

table	index	column	NDV	density	#LB	lvl
A	A1		2,500		267	2
		N3	50	2.0000E-02		
		N1	2,500	4.0000E-04		
		C1	1	1.0000E+00		
	A2		2,500		267	2
		N4	20	5.0000E-02		
		N2	1,250	8.0000E-04		
		C2	1	1.0000E+00		

Index Join - Example

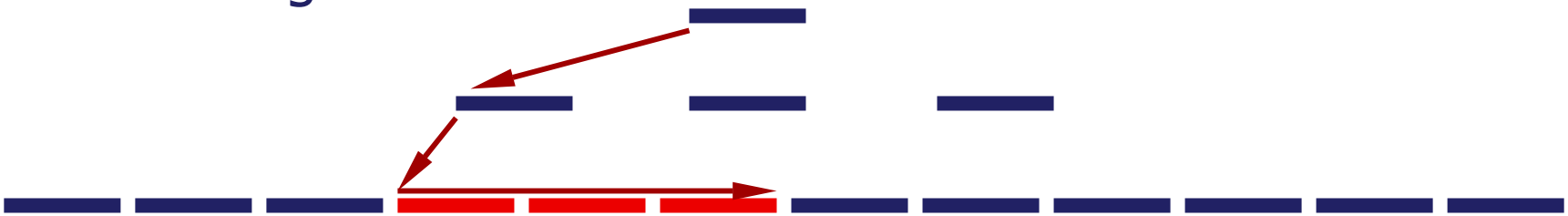
```
select a.c1, a.c2 from a where a.n3=10 and a.n4=10;

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation             |  Name               | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                     |    20 |  1520 |    31 |
|*  1 |  VIEW                 | index$_join$_001    |    20 |  1520 |    31 |
|*  2 |   HASH JOIN           |                     |    20 |  1520 |       |
|*  3 |    INDEX RANGE SCAN   | A1                  |    20 |  1520 |    45 |
|*  4 |    INDEX RANGE SCAN   | A2                  |    20 |  1520 |    45 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("A"."N3"=10 AND "A"."N4"=10)
   2 - access("indexjoin$_alias$_003".ROWID="indexjoin$_alias$_002".ROWID)
   3 - access("indexjoin$_alias$_002"."N3"=10)
   4 - access("indexjoin$_alias$_003"."N4"=10)
```

dbms_xplan.display(…'basic')

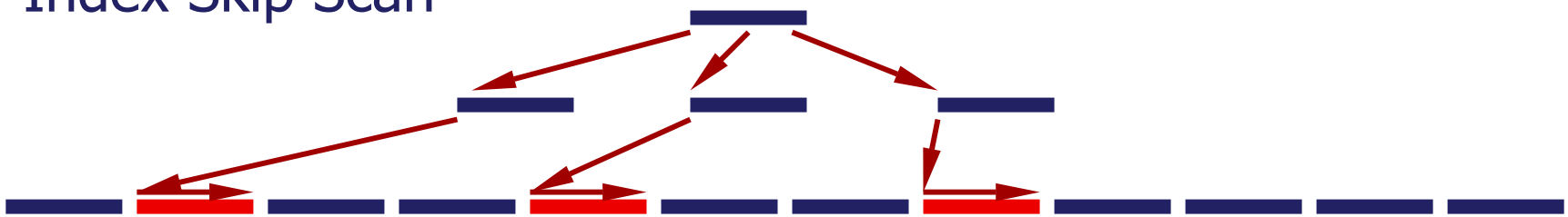
```
PLAN_TABLE_OUTPUT
-----------------------------------------------
| Id  | Operation             |  Name           |
-----------------------------------------------
|   0 | SELECT STATEMENT      |                 |
|   1 |  VIEW                 | index$_join$_001|
|   2 |   HASH JOIN           |                 |
|   3 |    INDEX RANGE SCAN   | A1              |
|   4 |    INDEX RANGE SCAN   | A2              |
-----------------------------------------------
```

New Base Access Path

Index Range Scan



Index Skip Scan



Index Skip Scan

provides two main benefits:

- improve the performance of certain queries, since queries which previously required table scans may now be able to take advantage of an existing index.
- allow an application to meet its performance goals with fewer indexes; fewer indexes will require less storage space and may improve the performance of DML and maintenance operations.

Index Skip Scan - Example

```
table  index  column    NDV         CLUF    CLUF   #LB lvl  #LB/K  #DB/K
------ ------ ------ ------ ----------- ------ ----- --- ------ ------
SS_A   SS_AI             746                 996    60   1      1      1
              N1           1  1.0000E+00
              N2          12  8.3333E-02
              N3          24  4.1667E-02

select ch from ss_a where n3=6;

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
| Id  | Operation                    |  Name       | Rows  | Bytes | Cost  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    42 | 31794 |    75 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SS_A        |    42 | 31794 |    75 |
|*  2 |   INDEX SKIP SCAN            | SS_AI       |     2 |       |    33 |
-----------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("SS_A"."N3"=6)
       filter("SS_A"."N3"=6)
```

Index Skip Scan - Example

```
table  index  column    NDV         CLUF    CLUF   #LB lvl  #LB/K  #DB/K
------ ------ ------ ------ ----------- ------ ----- --- ------ ------
SS_A   SS_AI             746                 996    60   1      1      1
              N1           1  1.0000E+00
              N2          12  8.3333E-02
              N3          24  4.1667E-02
```

```
select ch from ss_a where n3=6 and n1=3;
```

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
| Id  | Operation                    |  Name       | Rows  | Bytes | Cost  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    42 | 31920 |   103 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SS_A        |    42 | 31920 |   103 |
|*  2 |   INDEX RANGE SCAN           | SS_AI       |    42 |       |    61 |
-----------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("SS_A"."N1"=3 AND "SS_A"."N3"=6)
       filter("SS_A"."N3"=6)
```

Bind Value Peeking

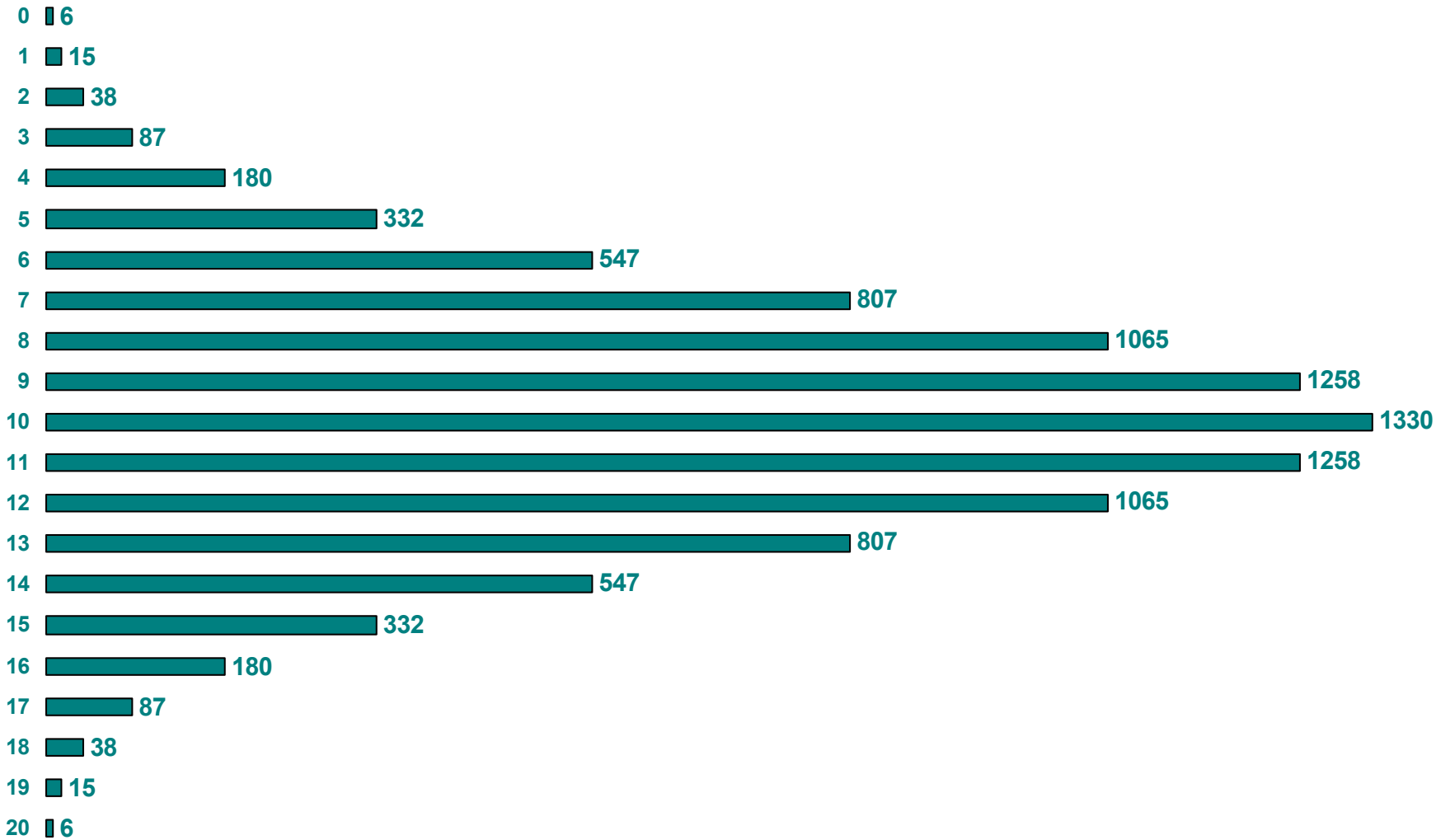
The CBO peeks at the values of user-defined bind variables on the first invocation [actually on a hard parse] of a cursor.

This feature lets the optimizer determine the selectivity of any where clause condition, as well as if literals have been used instead of bind variables.

On subsequent invocations of the cursor, no peeking takes place, and the cursor is shared, based on the standard cursor-sharing criteria, even if subsequent invocations use different bind values.

When bind variables are used in a statement, it is assumed that cursor sharing is intended and that different invocations are supposed to use the same execution plan.

Bind Value Peeking - Example



Bind Value Peeking - Example

```
alter system flush shared_pool;

exec :b1 := 11;
select sum(n2) from hist3 where n1 = :b1;
   SUM(N2)
----------
    635.37

 cost   card operation                          rows  cr gets  elapsed
----- ------ ------------------------------- -------- -------- --------
  130        SELECT STATEMENT
           1   SORT AGGREGATE                      1    3,379    58.16
  130  1,258     TABLE ACCESS FULL HIST3       1,258    3,379    55.56


exec :b1 := 0;
select sum(n2) from hist3 where n1 = :b1;
   SUM(N2)
----------
      2.98

 cost   card operation                          rows  cr gets  elapsed
----- ------ ------------------------------- -------- -------- --------
  130        SELECT STATEMENT
           1   SORT AGGREGATE                      1      638    47.34
  130  1,258     TABLE ACCESS FULL HIST3           6      638    47.20
```

Bind Value Peeking - Example

```
alter system flush shared_pool;

exec :b1 := 0;
select sum(n2) from hist3 where n1 = :b1;
   SUM(N2)
----------
      2.98

 cost   card operation                                 rows  cr gets  elapsed
----- ------ ---------------------------------------- ------ -------- --------
    8        SELECT STATEMENT
           1   SORT AGGREGATE                            1        8     0.42
    8      6     TABLE ACCESS BY INDEX ROWID HIST3        6        8     0.37
    2      6       INDEX RANGE SCAN HIST3_IX              6        2     0.18

exec :b1 := 11;
select sum(n2) from hist3 where n1 = :b1;
   SUM(N2)
----------
    635.37

 cost   card operation                                  rows  cr gets  elapsed
----- ------ ---------------------------------------- ------- -------- --------
    8        SELECT STATEMENT
           1   SORT AGGREGATE                             1       17    25.44
    8      6     TABLE ACCESS BY INDEX ROWID HIST3     1,258       17    22.49
    2      6       INDEX RANGE SCAN HIST3_IX           1,258        3     3.67
```

Explain Plan Dilemma

Name	Null?	Type
N1	NOT NULL	NUMBER
N2	NOT NULL	NUMBER
N3	NOT NULL	NUMBER
DT	NOT NULL	DATE
CH	NOT NULL	VARCHAR2(2000)

table	rows	blks	empty
PEEK	1,001	500	0

table	index	column	NDV	#LB	lvl	#LB/K	#DB/K
PEEK	PEEK_1		1,001	6	1	1	1
		N2	25				
		N1	1,001				
	PEEK_2		350	5	1	1	2
		N3	25				
		N2	25				

table	column	NDV	density	LO	HI	bkts
PEEK	N1	1,001	9.9900E-04	1	1020	1
PEEK	N2	25	4.9950E-04	1	25	24
PEEK	N3	25	4.0000E-02	0	24	1
PEEK	DT	25	4.0000E-02	2003-10-22	2003-11-15	1
PEEK	CH	61	1.6393E-02	0**********	Z**********	1

Explain Plan Dilemma

```
explain plan set statement_id = 'peek-21' for
select n1, dt, length(ch) from peek where n2 = :n2 and n3 > 5
```

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation                    |  Name       | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    32 | 31552 |    29 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PEEK        |    32 | 31552 |    29 |
|*  2 |   INDEX RANGE SCAN           | PEEK_2      |   792 |       |     5 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PEEK"."N3">5 AND "PEEK"."N2"=TO_NUMBER(:Z))
       filter("PEEK"."N2"=TO_NUMBER(:Z) AND "PEEK"."N3">5)
```

Explain Plan Dilemma

```
exec :n2 := 1;

select n1, dt, length(ch) from peek where n2 = :n2 and n3 > 5
```

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------
| Id  | Operation             |  Name       | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |             |       |       |    50 |
|*  1 |  TABLE ACCESS FULL    | PEEK        |   412 |   396K|    50 |
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("N2"=:N2 AND "N3">5)
```

Explain Plan Dilemma

```
exec :n2 := 21;

select n1, dt, length(ch) from peek where n2 = :n2 and n3 > 5
```

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation                    |  Name       | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |    16 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| PEEK        |    16 | 15776 |    16 |
|*  2 |   INDEX RANGE SCAN           | PEEK_1      |    20 |       |     2 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("N3">5)
   2 - access("N2"=:N2)
```

Dynamic Sampling

OPTIMIZER_DYNAMIC_SAMPLING a value from 0 to 10.

0 no dynamic sampling will be done.

1 dynamic sampling will be performed if

- There is more than one table in the query.
- Some table has not been analyzed, has no indexes, and a relatively expensive table scan would be required for this unanalyzed table.

2 Apply dynamic sampling to all unanalyzed tables.

3 Same as 2 plus all tables for which standard selectivity estimation used a guess for some predicate that is a potential dynamic sampling predicate.

4 Same as 3 plus all tables that have single-table predicates that reference 2 or more columns.

5-10 Same as 4 but with increasing multiples of the default number of sampling blocks sampled: 2, 4, 8, 32, 128 and all.

Dynamic Sampling

The dynamic sampling value can be set at 4 levels:

1. System wide in the init.ora
2. For the session with "alter session set …"
3. For the entire SQL by a hint

```
/*+ dynamic_sampling(5) */
```

4. For specific table(s) in a SQL by hint(s)

```
/*+ dynamic_sampling(e 5) */
```

however, optimizer_features_enable turns off dynamic sampling if set to a version prior to 9.0.2

New costing model

cost = (#srds * sreadtm + #mrds * mreadtm
+ #CPUcycles / CPUspeed) / sreadtm

cost = #srds
[+ #mrds * mreadtm/sreadtm]
[+ #CPUcycles / CPUspeed / sreadtm]

where

#srds	- number of single block reads
#mrds	- number of multi block reads
#CPUcycles	- number of CPU cycles
sreadtm	- single block read time
mreadtm	- multi block read time
CPUspeed	- CPU cycles per second

System Statistics

```
select * from sys.aux_stats$
```

SNAME	PNAME	PVAL1	PVAL2
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		02-24-2003 21:18
SYSSTATS_INFO	DSTOP		02-24-2003 21:18
SYSSTATS_INFO	FLAGS	1	
SYSSTATS_MAIN	SREADTIM	.684	
SYSSTATS_MAIN	MREADTIM	2.052	
SYSSTATS_MAIN	CPUSPEED	313	
SYSSTATS_MAIN	MBRC	5	
SYSSTATS_MAIN	MAXTHR	-1	
SYSSTATS_MAIN	SLAVETHR	-1	

System Statistics

If system statistics are manually set :

cpuspeed and sreadtim must be set for cpu_costing to become effective.

mreadtim, mbrc and sreadtim must be set for access path costing of multiblock paths (tsc, ffs) to use the system statistics.
Also, mreadtim must be bigger than sreadtim.

8.1.7 tsc = ceiling((nblks/dfmrc)*ratio)

9.2.0 tsc = ceiling((nblks/mbrc)*(mreadtim/sreadtim)) + 1*

New and extended v$ views

v$sql

v$sqlarea

v$sql_plan

v$sql_plan_statistics

v$sql_plan_statistics_all

new v$sql columns

- FETCHES
- PLAN_HASH_VALUE
- CPU_TIME
- ELAPSED_TIME
- OUTLINE_SID
- CHILD_ADDRESS
- SQLTYPE
- REMOTE
- OBJECT_STATUS
- LITERAL_HASH_VALUE

new v$sqlarea columns

FETCHES

CPU_TIME

ELAPSED_TIME

IS_OBSOLETE

CHILD_LATCH

v$sql_plan

Practically identical to PLAN_TABLE except for :

- hash_value and child_number
  instead of
  statement_id

- object_id
  instead of
  object_name and object_type

v$sql_plan_statistics

Only collected if init.ora parameter

STATISTICS_LEVEL = ALL

STATISTICS_LEVEL = {ALL | TYPICAL | BASIC}

Changes in sql trace and tkprof

If statistics_level is set to all, row source execution statistics are captured and reported by tkprof:

Without statistics_level=all:

```
Rows     Row Source Operation
-------  ---------------------------------------------------
   5338  SORT GROUP BY
  10710   HASH JOIN
   7865    TABLE ACCESS FULL PS_PAY_CHECK5
  10000    TABLE ACCESS FULL PS_JOB5
```

With statistics_level=all:

```
Rows     Row Source Operation
-------  ---------------------------------------------------
   5338  SORT GROUP BY (cr=4612 r=4549 w=0 time=1864965 us)
  10710   HASH JOIN  (cr=4612 r=4549 w=0 time=1776611 us)
   7865    TABLE ACCESS FULL PS_PAY_CHECK5 (cr=3687 r=3638 w=0 time=1439412 us)
  10000    TABLE ACCESS FULL PS_JOB5 (cr=925 r=911 w=0 time=151060 us)
```

Changes in sql trace and tkprof

If sql_trace was enabled with wait event details (level 8), tkprof now summarizes the waits:

```
Elapsed times include waiting on following events:
```

Event waited on	Times Waited	Max Wait	Total Waited
file open	6	0.00	0.00
db file sequential read	2948	0.05	4.64
log buffer space	1	0.01	0.01
SQL*Net message to client	3	0.00	0.00
SQL*Net message from client	3	0.01	0.01

References

Oracle9*i Release 2*	*Database Performance Tuning Guide and Reference*
Note:149560.1	*Collect and Display System Statistics (CPU and IO) for CBO usage*
Note:153761.1	*Scaling the system to improve CBO optimizer*

Wolfgang Breitling

Centrex Consulting Corporation

breitliw@centrexcc.com

www.centrexcc.com