

Tuning by Cardinality Feedback

Method and Examples




Wolfgang Breitling
www.centrexcc.com

The presentation will introduce a method of tuning which is based on the premise that whenever the CBO chooses a bad plan it can be traced back to an error in the estimation of the cardinality of one or more row sources.

It is important to remember that this is a premise, based on observation and experience, **not** a proven fact.


Tuning by cardinality feedback thus looks at discrepancies between estimated and real row source cardinalities of an execution plan and attempts to find ways to correct the CBO's error in estimation and trusting it to find a better plan based on the corrected, more accurate estimates.

The methodology is not dissimilar to that of profiles generated by `dbms_sqltune`.



Who am I

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

Member of the Oaktable Network 

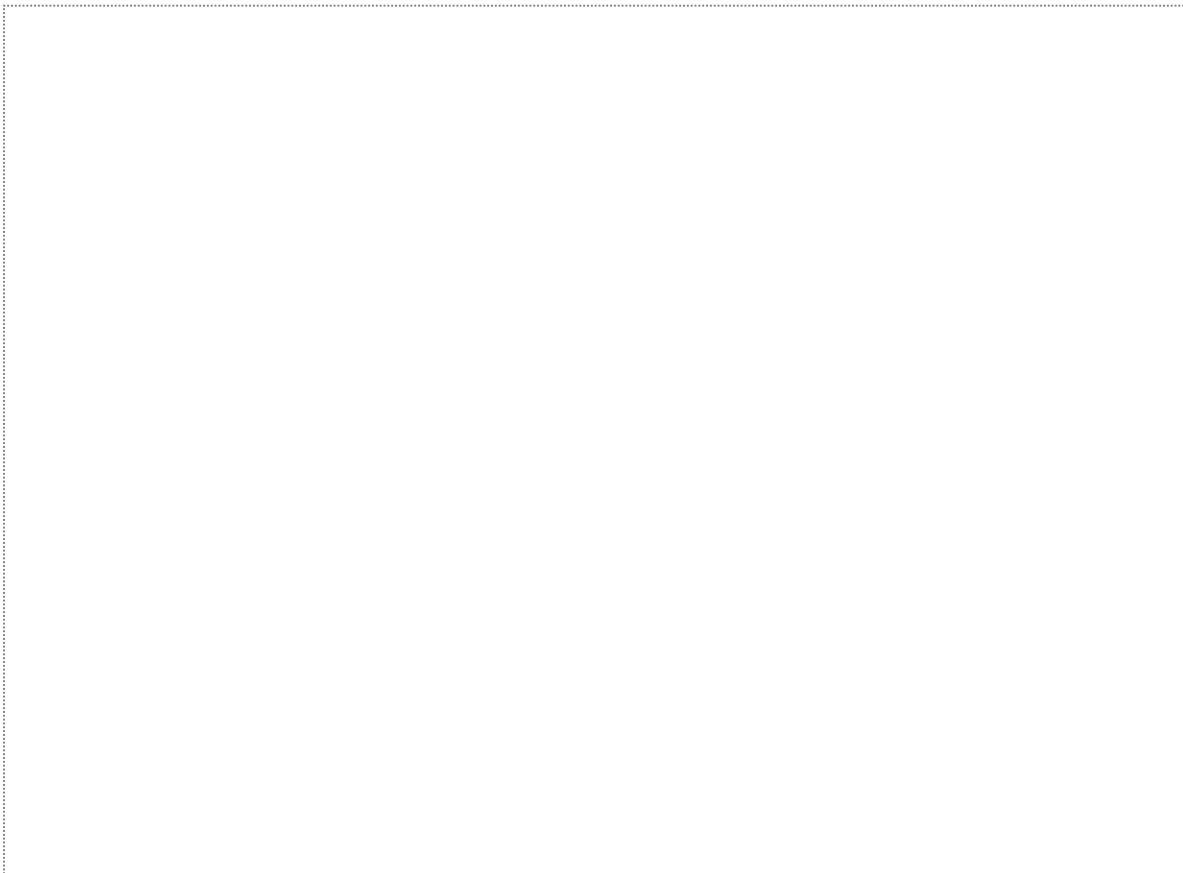
25+ years in database management
DL/1, IMS, ADABAS, SQL/DS, DB2, Oracle


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

Oracle since 1993 (7.0.12)

Mathematics major from University of Stuttgart

2 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005



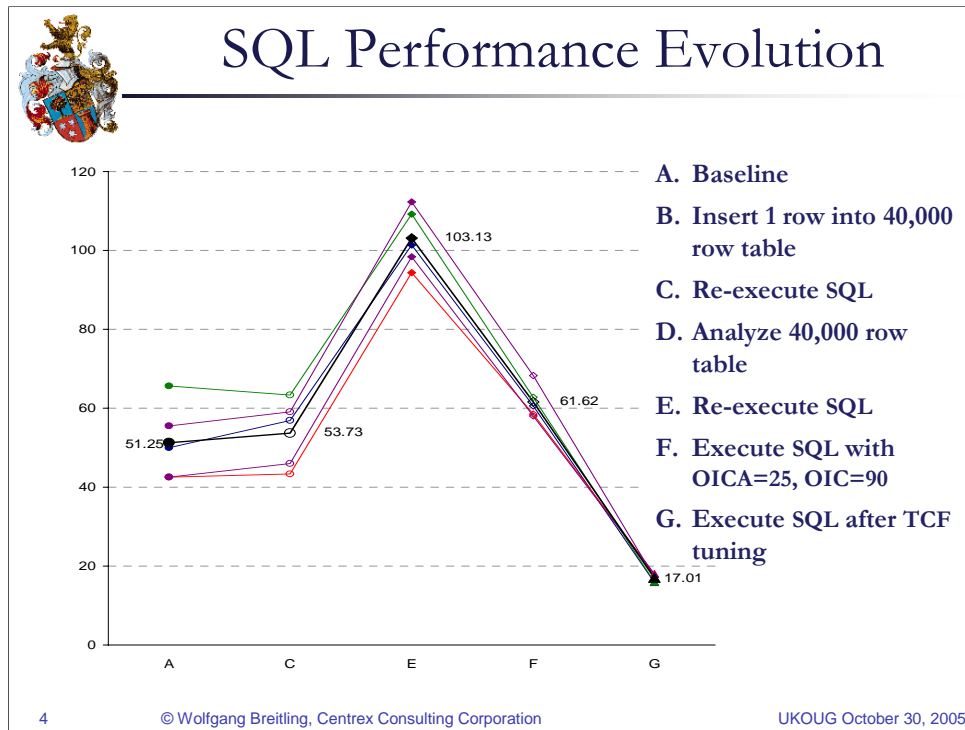


Focus


- ❖ The empirical Basis for the Method
- ❖ Explanation of the Method on a Test case
- ❖ 3 real-life examples of using the Method
- ❖ Comparing TCF to
 - ❖ Hints
 - ❖ Profiles

3 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

Also a word on how it stacks up to time-based tuning



The SQL at all five execution points is identical. All that changes between executions are the table statistics, or, at point F, optimizer parameters.



Tuning by Cardinality Feedback

Observation


If an access plan is not optimal it is because the cardinality estimate for one or more of the row sources is grossly incorrect.

5 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

The usual suggestion is to re-analyze all tables, possibly with a higher sampling percentage. But:

- a) Remember Dave Ensor's paradox: "It is only safe to gather statistics when to do so will make no difference"
- b) Outdated statistics on base tables and columns are only one possible reason for inaccurate cardinality estimates, especially of intermediate join results
- c) There are many reasons – i.e. assumptions – in the optimizer's logic which lead to over- or under-inflated estimates when violated, despite accurate statistics on the base tables and columns.
- d) Violation of the predicate independence assumption in particular, and its cousin, the join uniformity assumption, lead to cardinality estimates which are orders of magnitude too low, resulting in disastrous NL join plans
And setting optimizer_index_cost_adj to a value < 100 does nothing in those cases except entrench the NL choice even more firmly.

Of course, this is just an observation, not a proven fact.



Tuning by Cardinality Feedback

Conjecture

The CBO does an excellent job of finding the best access plan for a given SQL

provided

it is able – or given some help - to accurately estimate the cardinalities of the row sources in the plan

6 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

Based on the conviction that the CBO does an excellent job of finding the best access plan for a given SQL.

Again, this is conjecture, a hypothesis. It can not be derived at by logic. Even if the prior observation **were** provable:

- bad plan => [at least one] cardinality estimate is wrong

the inverse wouldn't follow from logic:

- cardinality estimate is correct => good plan

or even just

- cardinality estimate is wrong => bad plan



Tuning by Cardinality Feedback

Faced with an underperforming SQL, the question the “TCF” method is trying to answer is not

- ❖ What would be a better access plan?

But instead

- ❖ Why is this plan, which the CBO chose as optimal, performing so poorly?

7

© Wolfgang Breitling, Centrex Consulting Corporation

UKOUG October 30, 2005

Unlike many SQL tuning books and papers this method is **not** trying to direct the optimizer to a specific, “better” access path, be it through restructuring of the SQL or with hints.

TCF is trying to find what caused the CBO to choose a (presumably) sub-optimal access plan.

Once the answer to that question is found, the next goal is to find a way to remedy the cause for the miscalculation, but ultimately get out of the way and let the CBO do its job again.



Tuning by Cardinality Feedback

- ❶ List the explain plan with the cardinality projections
 - from explain or, preferably, from v\$sql_plan
- ❷ Get the actual row counts
 - from a sql trace or from v\$sql_plan_statistics.
 - Make sure the actual plan is identical to the explain plan!
- ❸ Look for the first (innermost) row source where the ratio of actual/estimated cardinality is orders of magnitude
 - usually at least in the 100s
- ❹ Find the predicates in the SQL for the tables that contribute to the row source with the miscalculated cardinality and look for violated assumptions:
 - Uniform distribution
 - Predicate independence
 - Join uniformity


8

© Wolfgang Breitling, Centrex Consulting Corporation

UKOUG October 30, 2005

1. Don't bother with the cost. Don't even display them. Once the CBO is done parsing they are really truly meaningless
2. If you can not get the actual row counts because the performance is so bad that the query does not finish (in a reasonable amount of time) or because it can not be repeated in the production setting and the test or QA database produces a different plan, you will need to get row counts by executing partial SQL which will provide the counts.
- 3.
4. This is easier said than done. It requires knowledge of the application data.

The initial analysis is akin to that of “Query tuning by eliminating throwaway”, aiming to identify steps that do too much unnecessary work.



Demo Case SQL

```


SELECT A.COMPANY, A.PAYGROUP, E.OFF_CYCLE, E.SEPCHK_FLAG, E.TAX_METHOD
, E.TAX_PERIODS, C.RETROPAY_ERNCND, sum(C.AMOUNT_DIFF)
from PS_PAY_CALENDAR A
, WB_JOB B
, WB_RETROPAY_EARNS C
, PS_RETROPAY_RQST D
, PS_RETROPAYFGM_TBL E
where A.RUN_ID = 'PDZ'
and A.PAY_CONFIRM_RUN = 'N'
and B.COMPANY = A.COMPANY
and B.PAYGROUP = A.PAYGROUP
and B.EFFDT = (SELECT MAX(F.EFFDT) from WB_JOB F
where F.EMPLID = B.EMPLID
and F.EMPL_RCD# = B.EMPL_RCD#
and F.EFFDT <= A.PAY_END_DT)
and B.EFFSEQ = (SELECT MAX(G.EFFSEQ) from WB_JOB G
where G.EMPLID = B.EMPLID
and G.EMPL_RCD# = B.EMPL_RCD#
and G.EFFDT = B.EFFDT)
and C.EMPLID = B.EMPLID
and C.EMPL_RCD# = B.EMPL_RCD#
and C.RETROPAY_PRCS_FLAG = 'C'
and C.RETROPAY_LOAD_SW = 'Y'
and D.RETROPAY_SEQ_NO = C.RETROPAY_SEQ_NO
and E.RETROPAY_PGM_ID = D.RETROPAY_PGM_ID
and E.OFF_CYCLE = A.PAY_OFF_CYCLE_CAL
group by A.COMPANY, A.PAYGROUP, E.OFF_CYCLE, E.SEPCHK_FLAG, E.TAX_METHOD
, E.TAX_PERIODS, C.RETROPAY_ERNCND

```

9 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

The WB_ prefixed tables are scaled back versions of the originals solely for performance reason. With the original sizes the query never finished [was not given the time to finish] . All the data in the demo tables are made up, but the tables, except for the scaled back WB_ tables, have sizes and other statistics to match the originals. The real PS_RETROPAY_EARNS tables was more than 10 times the size of its demo sibling, ~ 1.5 million rows.

The plans of the sql in this demo are different from the ones in reality, but the main point I want to make and the tuning solution are identical.




Demo Case Plan

	Rows	card	operation
		2	SELECT STATEMENT
	2	2	SORT GROUP BY
	6,274		FILTER
504.6	13,120	26	HASH JOIN
534.9	208,620	390	HASH JOIN
1.0	15	15	TABLE ACCESS FULL PS_RETROPAYPGM_TBL
858.1	44,621	52	NESTED LOOPS
353.3	14,131	40	HASH JOIN
1.0	5	5	TABLE ACCESS FULL PS_PAY_CALENDAR
3.0	40,000	13,334	TABLE ACCESS FULL WB_JOB
1.6	44,621	27,456	TABLE ACCESS BY INDEX ROWID WB_RETROPAY_EARNS
2.7	74,101	27,456	INDEX RANGE SCAN WB0RETROPAY_EARNS
1.0	13,679	13,679	TABLE ACCESS FULL PS_RETROPAY_RQST
	9,860	1	SORT AGGREGATE
	4,930	1	FIRST ROW
	4,930	1	INDEX RANGE SCAN (MIN/MAX) WB_JOB
	20,022	1	SORT AGGREGATE
	7,750	1	FIRST ROW
	10,011	1	INDEX RANGE SCAN (MIN/MAX) WB_JOB

10 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

The plan and the ratios



Tuning by Cardinality Feedback


table	column	NDV	density	lo	hi	bkts
PS_PAY_CALENDAR	COMPANY	11	9.0909E-02	ACE	TES	1
	PAYGROUP	15	6.6667E-02	ACA	TEP	1
	PAY_END_DT	160	6.2500E-03	1998-01-18	2004-02-22	1
	RUN_ID	240	4.1667E-03		PP2	1
	PAY_OFF_CYCLE_CAL	2	5.0000E-01	N	Y	1
	PAY_CONFIRM_RUN	2	5.0000E-01	N	Y	1
WB_JOB	EMPLID	26,167	3.8216E-05	000036	041530	1
	EMPL_RCD#	1	1.0000E+00	0	0	1
	EFFDT	10	1.0000E-01	1995-01-01	2004-02-01	1
	EFFSEQ	3	3.3333E-01	1	3	1
	COMPANY	10	1.0000E-01	ACE	TES	1
	PAYGROUP	14	7.1429E-02	ACA	TEP	1

11 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

One may assume that it is the low cardinalities of effdt and effseq that make the plan vulnerable to change from just small variations in the statistics (i.e. like a pantograph).

In fact, in the original case there were > 20,000 different effdt and 11 effseq. I do not know at which point the actual sql switched to the really bad plan. It was part of Peoplesoft's retro-pay program which is only run when salaries need to be recalculated because of a retro-active contract agreement. It had not been run for a year or more while the effdt and effseq cardinalities kept building up.

Even with this scaled down example, the sql continues to vacillate between different plans as the cardinalities of effdt and effseq change.



Adjusting the Statistics

Adjust the column statistics to counteract the violated assumption(s):

```
DBMS_STATS.SET_COLUMN_STATS('SCOTT','WB_JOB','EFFDT',density => 1);
DBMS_STATS.SET_COLUMN_STATS('SCOTT','WB_JOB','EFFSEQ',density => 1);
```

table	column	NDV	density	lo	hi	bkts
WB_JOB	EMPLID	26,167	3.8216E-05	000036	041530	1
	EMPL_RCD#	1	1.0000E+00	0	0	1
	EFFDT	10	1.0000E+00	1995-01-01	2004-02-01	1
	EFFSEQ	3	1.0000E+00	1	3	1
	COMPANY	10	1.0000E-01	ACE	TES	1
	PAYGROUP	14	7.1429E-02	ACA	TEP	1

12 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005


If emplid, effdt and effseq **were** independent then wb_job would need to have $26,167 * 10 * 3 = 785,010$ rows. To counteract the estimate miscalculation due to the violated predicate independence assumption, we neutralize the effdt and effseq cardinalities by setting them to 1.

In this case there is virtually no danger of side effects as it is extremely unlikely that EFFDT, much less EFFSEQ, will be used by themselves as predicates in a query.

The optimizer uses sometimes density and sometimes NDV – and other times yet other statistics – when estimating predicate selectivity.

When modifying column statistics I try to leave the real NDV in place if possible and only change the density. This makes it “obvious” that the statistics were modified after gathering.

- Note: setting set_column_stats(distcnt=>) not only changes NDV, but also density to $1/\text{distcnt}$, while set_column_stats(density=>) only changes density, leaving NDV as it is.



Plan after adjusting Statistics

	<u>Rows</u>		<u>card</u> <u>operation</u>
		2	SELECT STATEMENT
	2	2	SORT GROUP BY
	6,274		FILTER
17.5	13,120	750	HASH JOIN
1.0	15	15	TABLE ACCESS FULL PS_RETROPAYPGM_TBL
28.1	42,054	1,499	HASH JOIN
29.8	44,621	1,499	HASH JOIN
9.9	14,130	1,429	HASH JOIN
1.0	5	5	TABLE ACCESS FULL PS_PAY_CALENDAR
1.0	40,000	40,000	TABLE ACCESS FULL WB_JOB
4.5	122,813	27,456	TABLE ACCESS FULL WB_RETROPAY_EARNS
1.0	13,679	13,679	TABLE ACCESS FULL PS_RETROPAY_RQST
	11,212	1	SORT AGGREGATE
	5,606	1	FIRST ROW
	5,606	1	INDEX RANGE SCAN (MIN/MAX) WB_JOB
	17,374	1	SORT AGGREGATE
	6,418	2	FIRST ROW
	8,687	2	INDEX RANGE SCAN (MIN/MAX) WB_JOB

13 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

Comments on the plan after adjusting the column statistics:

1. The ratios of actual/estimated are much smaller
2. The cardinalities of 4 of the 5 tables are accurately estimated
3. Even though the estimates for the cardinalities of ps_pay_calendar and wb_job were correct, the CBO underestimated the cardinality of their join by a factor of 10, suggesting a violation of the join uniformity assumption.

It may appear that TCF is only about adjusting statistics. This is not strictly true. It is about addressing the cause for the cardinality estimate miscalculation. Virtually always this is caused by missing information. The additional information may come from an additional index, or from a histogram (which in turn changes the statistics). In this case, the information about the predicate dependency can not be given directly to the CBO. By faking the column statistics, two wrongs make a right – in this particular case



Tuning Examples

Unlike the demo case, which was an artificial test case
– albeit based on a real-life case –
the following examples are actual,
recent tuning cases.


For the record, the examples are from a system running Oracle 9.2.0.6 with system statistics.

14

© Wolfgang Breitling, Centrex Consulting Corporation

UKOUG October 30, 2005

SREADTIM	1.971
MREADTIM	2.606
CPUSPEED	618
MBRC	8
MAXTHR	-1
SLAVETHR	-1



Example 1

```

SELECT R.PRC SIN ST AN CE ,R. OR I G P RC S IN ST AN CE ,R. RE CU R O R I G P RC S IN ST ,
R. MA I N JO B I N ST AN CE ,R. P RC S JO B SE Q ,R. P RC S JO B NA M E ,R. P RC S NA M E ,R. P RC ST Y P E , R. RE CU R NA M E
FR O M P S P RC S Q U E R , P S _ P RC S R E CU R S
W H E R E ((R. R U N ST A T U S I N (: " S Y S _ B _ 0 0 " , : " S Y S _ B _ 0 1 " ) A N D S. I N I T I A T E W H E N = : " S Y S _ B _ 0 2 " )
O R (R. R U N ST A T U S I N (: " S Y S _ B _ 0 3 " , : " S Y S _ B _ 0 4 " , : " S Y S _ B _ 0 5 " , : " S Y S _ B _ 0 6 " ,
: " S Y S _ B _ 0 7 " , : " S Y S _ B _ 0 8 " , : " S Y S _ B _ 0 9 " ) A N D S. I N I T I A T E W H E N = : " S Y S _ B _ 1 0 " ) )
A N D R. I N I T I A T E D N E X T = : " S Y S _ B _ 1 1 "
A N D R. O P S Y S = : 1
A N D R. R U N L O C A T I O N = : " S Y S _ B _ 1 2 "
A N D R. R E CU R NA M E < > : " S Y S _ B _ 1 3 "
A N D R. P RC S JO B SE Q = : " S Y S _ B _ 1 4 "
A N D R. S E R V E R NA M E R U N = : 2
A N D R. R E CU R NA M E = S. R E CU R NA M E
    
```

COST	CARD	operation	ELAPSED	ROWS	CR_GETS
220		SELECT STATEMENT			
220	12,516	HASH JOIN	0.060	0	2,362
3	15	TABLE ACCESS FULL PS_PRC S R E CU R	0.000	15	3
214	34,057	TABLE ACCESS FULL PS P RC S Q U E	0.060	0	2,359

Rows	Row Source Operation
0	HASH JOIN (cr=2362 r=0 w=0 time=59799 us)
15	TABLE ACCESS FULL PS_PRC S R E CU R (cr=3 r=0 w=0 time=171 us)
0	TABLE ACCESS FULL PS P RC S Q U E (cr=2359 r=0 w=0 time=58218 us)


15
© Wolfgang Breitling, Centrex Consulting Corporation
UKOUG October 30, 2005

The explain and execution numbers are from v\$sql_plan and v\$sql_plan_statistics.

The operation detail with the gross cardinality mismatch is highlighted.

Note that on some platforms, all that I have worked with, you need to set statistics_level=all in order to get execution details from v\$sql_plan_statistics.

Contrary to my advice earlier, I **did** include the plan cost in the display for reasons which will become clear later.



Example 2

```

SELECT Q.PRCINSINSTANCE, Q.JOBINSTANCE, Q.MAINJOBINSTANCE, Q.SESSIONIDNUM
, Q.OPRID, Q.OUTDESTTYPE, Q.GENPRCSTYPE, Q.PRCSTYPE
, P.PRCOUTPUTDIR FROM PSPRCSQE Q
, PSPRCSPARMS P
WHERE Q.RUNSTATUS = :1
AND Q.SERVERNAMERUN = :2
AND Q.RUNLOCATION = : "SYS_B_0"
AND Q.PRCINSINSTANCE = P.PRCINSINSTANCE
    
```


COST	CARD	operation	ELAPSED	ROWS	CR GETS
546		SELECT STATEMENT			
546	1,065	HASH JOIN	0.240	1	6,922
201	1,101	TABLE ACCESS FULL PSPRCSQE	0.030	1	2,359
341	36,284	TABLE ACCESS FULL PSPRCSPARMS	0.080	38,539	4,563

Rows	Row Source	Operation	
1	HASH JOIN		(cr=6922 r=0 w=0 time=236770 us)
1	TABLE ACCESS FULL PSPRCSQE		(cr=2359 r=0 w=0 time=30341 us)
38539	TABLE ACCESS FULL PSPRCSPARMS		(cr=4563 r=0 w=0 time=77536 us)

16
© Wolfgang Breitling, Centrex Consulting Corporation
UKOUG October 30, 2005




Example 3

```

SELECT R.PRCINSTANCE, R.ORIGPRCSINSTANCE, R.JOBINSTANCE, R.MAINJOBINSTANCE
, R.MAINJOBNAME, R.PRCITEMLEVEL, R.PRCJOBSEQ, R.PRCJOBNAME, R.PRCSTYPE
, R.PRCNAME, R.PRCSPRTY, TO_CHAR(R.RUNDTM, 'SYS_B_00'), R.GENPRCSTYPE
, R.OUTDESTTYPE, R.RETRYCOUNT, R.RESTARTENABLED, R.SERVERNAMERQST, R.OPSYS
, R.SCHEDULENAME, R.PRCSCATEGORY, R.P_PRCINSTANCE, C.PRCSPRIORITY
, S.PRCSPRIORITY, R.PRCWINPOP, R.MCFREN_URL_ID
FROM PSPRCSQUE R, PS_SERVERCLASS S, PS_SERVERCATEGORY C
WHERE R.RUNDTM <= SYSDATE
AND R.OPSYS = :1 AND R.RUNSTATUS = :2
AND (R.SERVERNAMERQST = :3 OR R.SERVERNAMERQST = 'SYS_B_01')
AND S.SERVERNAME = :4 AND R.PRCSTYPE = S.PRCSTYPE
AND R.PRCSCATEGORY = C.PRCSCATEGORY AND S.SERVERNAME = C.SERVERNAME
AND ((R.PRCJOBSEQ = 'SYS_B_02' AND R.PRCSTYPE <> 'SYS_B_03')
OR (R.PRCJOBSEQ > 'SYS_B_04' AND R.MAINJOBINSTANCE IN (
SELECT A.MAINJOBINSTANCE FROM PSPRCSQUE A WHERE A.MAINJOBINSTANCE > 'SYS_B_05'
AND A.PRCSTYPE='SYS_B_06' AND A.RUNSTATUS='SYS_B_07'
AND A.PRCJOBSEQ = 'SYS_B_08' AND (A.SERVERNAMERUN = 'SYS_B_09' OR
A.SERVERNAMERUN = :5))))
AND C.MAXCONCURRENT > 'SYS_B_10'
ORDER BY C.PRCSPRIORITY DESC, R.PRCSPRTY DESC, S.PRCSPRIORITY DESC, R.RUNDTM ASC

```



Example 3


COST	CARD	operation	ELAPSED	ROWS	CR_GETS
221		SELECT STATEMENT			
221	108	SORT ORDER BY	0.040	0	2,363
		FILTER			2,363
220	108	HASH JOIN			2,363
5	8	MERGE JOIN CARTESIAN	0.000	7	4
3	1	TABLE ACCESS BY INDEX ROWID PS_SERVERCATEGORY		1	2
2	1	INDEX RANGE SCAN PS_SERVERCATEGORY			1
2	8	BUFFER SORT		7	2
3	8	TABLE ACCESS BY INDEX ROWID PS_SERVERCLASS			2
2	8	INDEX RANGE SCAN PS_SERVERCLASS			1
215	108	TABLE ACCESS FULL P\$PRCSQUE	0.040	0	2,359
		FILTER	0.000	0	0
3	1	TABLE ACCESS BY INDEX ROWID P\$PRCSQUE			0
2	5	INDEX RANGE SCAN P\$D\$PRCSQUE			0

Rows	Row Source Operation	
0	SORT ORDER BY	(cr=2363 r=0 w=0 time=39950 us)
0	FILTER	(cr=2363 r=0 w=0 time=39930 us)
0	HASH JOIN	(cr=2363 r=0 w=0 time=39926 us)
7	MERGE JOIN CARTESIAN	(cr=4 r=0 w=0 time=213 us)
1	TABLE ACCESS BY INDEX ROWID PS_SERVERCATEGORY	(cr=2 r=0 w=0 time=70 us)
1	INDEX RANGE SCAN PS_SERVERCATEGORY	(cr=1 r=0 w=0 time=40 us)
7	BUFFER SORT	(cr=2 r=0 w=0 time=97 us)
7	TABLE ACCESS BY INDEX ROWID PS_SERVERCLASS	(cr=2 r=0 w=0 time=50 us)
7	INDEX RANGE SCAN PS_SERVERCLASS	(cr=1 r=0 w=0 time=25 us)
0	TABLE ACCESS FULL P\$PRCSQUE	(cr=2359 r=0 w=0 time=39040 us)
0	FILTER	
0	TABLE ACCESS BY INDEX ROWID P\$PRCSQUE	
0	INDEX RANGE SCAN P\$D\$PRCSQUE	

18
© Wolfgang Breitling, Centrex Consulting Corporation
UKOUG October 30, 2005

In the three examples there are two (E1 and E3) where an actual row count is 0. This is another indicator to watch for – of course it can be viewed as a special case of the ratio indicator since any ratio with 0 as the denominator is out of bounds.

Provided the row source cardinality is **consistently** 0, then either that part of the sql and plan is obsolete, or the plan could benefit from executing that part early and reduce or eliminate subsequent work.




Statistics

TABLE_NAME	rows	blks	empty	row						
PSPRCSQUE	38,539	2,326	0	204						
table	index	column	NDV	DENS	#LB	lvl	#LB/K	#LB/K	CLUF	
PSPRCSQUE	PSAPSPRCSQUE		43		257	2	5	116	4,998	
		SERVERNAMERQST	3	3.3333E-01	0		0	0	0	
		SERVERNAMERUN	3	3.3333E-01	0		0	0	0	
		OPSYS	2	5.0000E-01	0		0	0	0	
		RUNSTATUS	11	9.0909E-02	0		0	0	0	
	PSBPSPRCSQUE		38,539		242	2	1	1	3,735	
		SERVERNAMERUN	3	3.3333E-01	0		0	0	0	
		PRCSINSTANCE	38,539	2.5948E-05	0		0	0	0	
	PSCPSPRCSQUE		38,539		315	1	1	1	2,658	
		PRCSINSTANCE	38,539	2.5948E-05	0		0	0	0	
		SESSIONIDNUM	9,015	1.1093E-04	0		0	0	0	
		OPRID	139	7.1942E-03	0		0	0	0	
	PSDPSPRCSQUE		7,249		174	1	1	1	4,395	
		MAINJOBINSTANCE	7,249	1.3795E-04	0		0	0	0	
	PSEPSPRCSQUE		10,735		221	2	1	1	3,121	
		RECURORIGPRCSINST	10,731	9.3188E-05	0		0	0	0	
		RECURNAME	4	2.5000E-01	0		0	0	0	
		INITIATEDNEXT	2	5.0000E-01	0		0	0	0	
	PS_PSPRCSQUE		38,539		166	1	1	1	2,658	
		PRCSINSTANCE	38,539	2.5948E-05	0		0	0	0	

19 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

Table and column statistics for the psprcsque table which was fingered in all three examples as the one with the cardinality estimate problem.

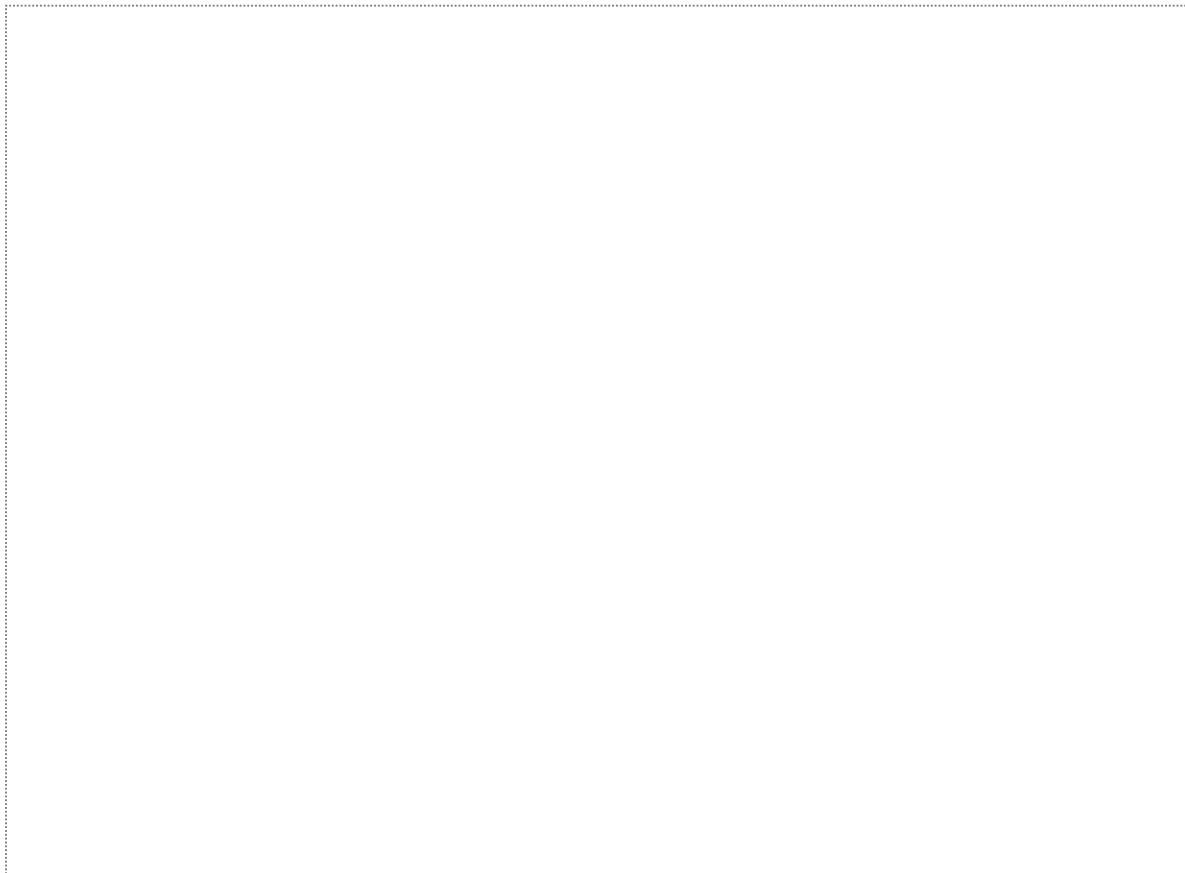


Tuning Example 1

20

© Wolfgang Breitling, Centrex Consulting Corporation


UKOUG October 30, 2005





Tuning Steps

- 1** Create an index on psprcsque
`create index uc_psprcsque_ix1 on psprcsque(prcsjobseq,recurname)`
But that alone did not change the plan
- 2** Create a (frequency) histogram on prcsjobseq
Did not change the plan either
- 3** Modifying the psprcsque.prcsjobseq statistics
finally did - in conjunction with the index



Histogram Attempt

With index and histogram on prcsjobseq?

```
create index UC_PSPRCSEQ_IX1 on PSPRCSEQ (PRCSJOBSEQ, RECURNAME);
```

Table	column	NDV	density	nulls	lo	hi	av lg	bkts
PSPRCSEQ	PRCSJOBSEQ	16	1.2974E-05	0	0	15	3	15

COST	CARD	operation	ELAPSED	ROWS	CR_GETS
209		SELECT STATEMENT			
209	236	HASH JOIN	0.040	0	2,362
3	15	TABLE ACCESS FULL PS_PRCRECUR	0.000	15	3
206	641	TABLE ACCESS FULL PSPRCSEQ	0.040	0	2,359

Rows	Row Source	Operation	
0	HASH JOIN		(cr=2362 r=0 w=0 time=36877 us)
15	TABLE ACCESS FULL PS_PRCRECUR		(cr=3 r=0 w=0 time=132 us)
0	TABLE ACCESS FULL PSPRCSEQ		(cr=2359 r=0 w=0 time=35994 us)

22 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

WHERE [...]


```
AND R.INITIATEDNEXT = : "SYS_B_11"
AND R.OPSYS = :1
AND R.RUNLOCATION = : "SYS_B_12"
AND R.RECURNAME <> : "SYS_B_13"
AND R.PRCJOBSEQ = : "SYS_B_14"
AND R.SERVERNAMERUN = :2
AND R.RECURNAME = S.RECURNAME
```

Obviously the system is running with `cursor_sharing=force`.

The index alone did not budge the plan, so a histogram on `prcsjobseq` was collected.

Why the focus on `prcsjobseq`?

- it was determined that the distribution of values (15) was skewed.
- Although the value in the predicate (`:sys_b_14` – remember in the original sql this is a constant) is the most frequently occurring value (28%), in combination with `recurname <> :sys_b_13` it is actually rather selective.



Statistics Attempt A

```
set_column_stats(USER,'PSPRCSQUE','PRCSJOBSEQ',distcnt=>250);
```

Table	column	NDV	density	nulls	lo	hi	av lg	bkts
PSPRCSQUE	PRCSJOBSEQ	250	4.0000E-03	0	0	15	3	1

COST	CARD	operation	ELAPSED	ROWS	CR_GETS
40		SELECT STATEMENT			
40	3	HASH JOIN	0.010	0	112
37	9	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			112
3	109	INDEX RANGE SCAN UC_PSPRCSQUE_IX1		112	49
3	15	TABLE ACCESS FULL PS_PRCRECUR	0.000	0	0

Rows	Row Source	Operation	
0	HASH JOIN		(cr=112 r=0 w=0 time=14021 us)
0	TABLE ACCESS BY INDEX ROWID PSPRCSQUE		(cr=112 r=0 w=0 time=13904 us)
112	INDEX RANGE SCAN UC_PSPRCSQUE_IX1		(cr=49 r=0 w=0 time=13465 us)
0	TABLE ACCESS FULL PS_PRCRECUR		


23 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

The index and the statistics change from the histogram were ineffective in adjusting the cardinalities such that the CBO would pick a different plan. In hindsight that should not be a surprise since the predicate uses the most frequently occurring value, not exactly a recipe for an index.

So the next attempt was to increase the selectivity of prcsjobseq by increasing its cardinality. That finally had the effect of giving a better performing plan.

Note: No sql change, no hint, no stored outline. Tuning achieved by adding an index and modifying a column statistic.

Note also that the cost of the plan went down, although that is not of primary importance. More important is that with the row source cardinalities, the row source elapsed time went down as well.



Statistics Attempt B

```

create index UC_PSPRCSQUE_IX1 on PSPRCSQUE (PRCSJOBSEQ, RECURNAME);
set_column_stats(USER, 'PSPRCSQUE', 'PRCSJOBSEQ', distcnt=>1000);

```

Table	column	NDV	density	nulls	lo	hi	av lg	bkts
PSPRCSQUE	PRCSJOBSEQ	1,000	1.0000E-03	0	0	15	3	1

COST	CARD	operation	ELAPSED	ROWS	CR_GETS
14		SELECT STATEMENT			
14	1	NESTED LOOPS	0.010	0	112
12	2	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			112
3	27	INDEX RANGE SCAN UC_PSPRCSQUE_IX1		112	49
2	1	TABLE ACCESS BY INDEX ROWID PS_PRCRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCRECUR			0

Rows	Row Source	Operation
0	NESTED LOOPS	(cr=112 r=48 w=0 time=6044 us)
0	TABLE ACCESS BY INDEX ROWID PSPRCSQUE	(cr=112 r=48 w=0 time=6040 us)
112	INDEX RANGE SCAN UC_PSPRCSQUE_IX1	(cr=49 r=48 w=0 time=5586 us)
0	TABLE ACCESS BY INDEX ROWID PS_PRCRECUR	
0	INDEX UNIQUE SCAN PS_PRCRECUR	


24 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

Different statistics – higher NDV => still another plan. Even lower cost

Note that in this case with the three different plans and the TCF method of adjusting **incorrect** cardinality estimates through purposely modified statistics, there is a clear correlation between the plan cost and the query execution speed.



Tuning Example 2



Tuning Steps


- 1 There is a – reasonably usable – index on psprcsque
- 2 Create a (frequency) histogram on runstatus
With the changed column statistics, the optimizer did use the index

26 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

```
WHERE Q.RUNSTATUS = :1  
AND Q.SERVERNAMERUN = :2  
AND Q.RUNLOCATION = "SYS_B_0"  
AND Q.PRC SINSTANCE = P.PRC SINSTANCE
```

The rationale for the histogram is again the skew in the distribution of runstatus values (see notes on next slide) combined with the fact that this time the predicate value, despite being a genuine bind variable, is almost always one of the least frequently occurring values ('1'), never the most frequently occurring (96%) value.

No surprise therefore that this time the histogram gave the CBO the necessary information to adjust the cardinality estimate and choose a different plan.



Tuned Example 2

Table	column	NDV	density	nulls	lo	hi	av lg	bkts
PSPRCSQUE	RUNSTATUS	11	1.3764E-05	0	1	9	3	10

COST	CARD	operation	ELAPSED	ROWS	CR GETS
133		SELECT STATEMENT			
133	1	NESTED LOOPS	0.020	1	16
132	1	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			13
131	1	INDEX SKIP SCAN PSAPSPRCSQUE			12
2	1	TABLE ACCESS BY INDEX ROWID PSPRCSPARMS	0.000	1	3
1	1	INDEX UNIQUE SCAN PS_PSPRCSPARMS			2


Rows	Row Source	Operation	
1	NESTED LOOPS		(cr=16 r=0 w=0 time=16426 us)
1	TABLE ACCESS BY INDEX ROWID PSPRCSQUE		(cr=13 r=0 w=0 time=16338 us)
1	INDEX SKIP SCAN PSAPSPRCSQUE		(cr=12 r=0 w=0 time=16313 us)
1	TABLE ACCESS BY INDEX ROWID PSPRCSPARMS		(cr=3 r=0 w=0 time=70 us)
1	INDEX UNIQUE SCAN PS_PSPRCSPARMS		(cr=2 r=0 w=0 time=49 us)

27
© Wolfgang Breitling, Centrex Consulting Corporation
UKOUG October 30, 2005

The explanation to the apparent contradiction of 11 distinct values, a low of 1, and a hi of 9 is that runstatus is not numeric, but character:
 select runstatus, count(0) from pspcrsque group by runstatus

RU	COUNT(0)
1	3
10	418
16	165
17	173
2	184
3	111
4	310
5	5
7	2
8	90
9	37078

Once more, the cost of the tuned query is lower, so there again is a correlation between cost and speed.

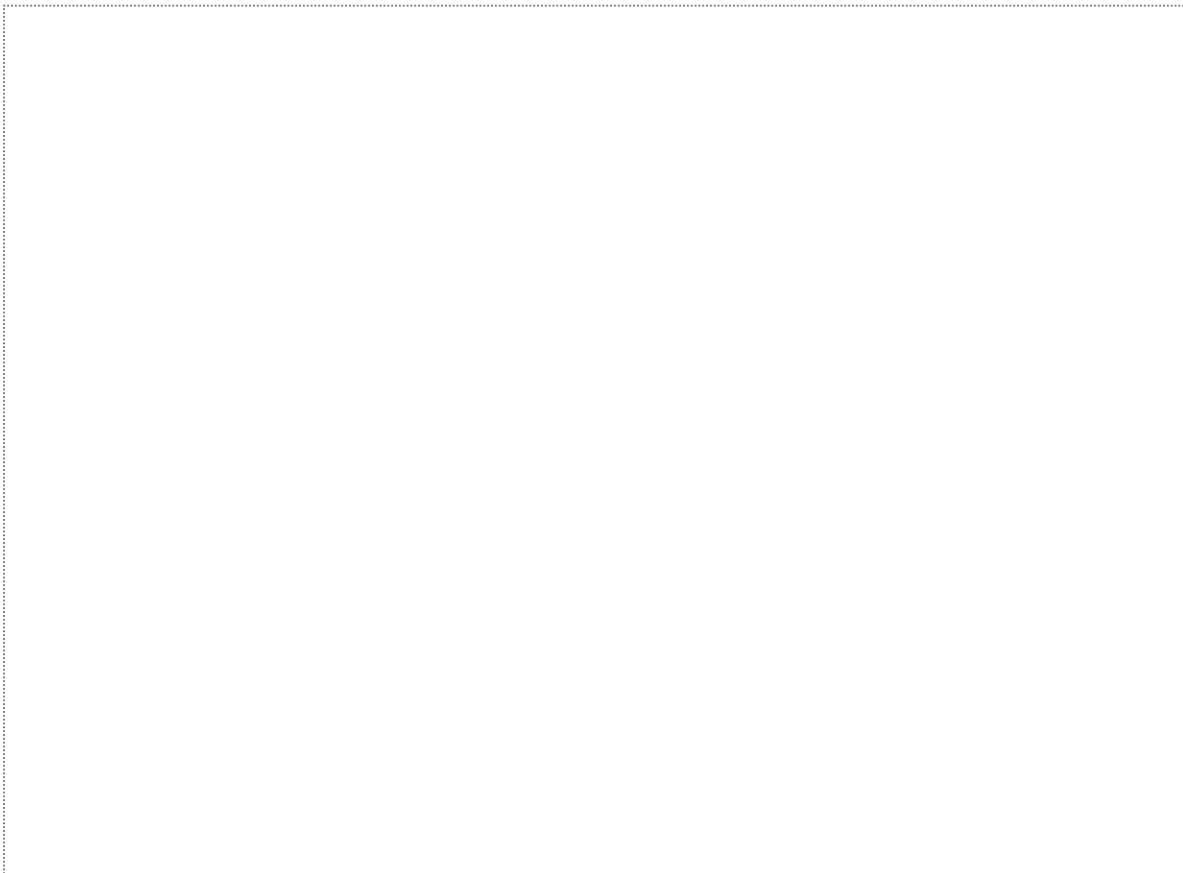


Tuning Example 3

28

© Wolfgang Breitling, Centrex Consulting Corporation

UKOUG October 30, 2005






Tuning Steps

- 1 By the time we got to tuning this SQL, there was nothing left to do.
The SQL got tuned as well by the actions to tune the other two.

It must be the histogram on `psprcsque.runstatus`, which occurs in the predicate of this sql as well, since the added index is not being used in the new plan.



Tuned Example 3


COST	CARD	operation	ELAPSED	ROWS	CR_GETS
160		SELECT STATEMENT			
160	5	SORT ORDER BY	0.010	0	2
		FILTER			2
159	5	HASH JOIN			2
154	5	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			2
156	5	NESTED LOOPS		2	2
3	1	TABLE ACCESS BY INDEX ROWID PS_SERVERCATEGORY	0.000	1	2
2	1	INDEX RANGE SCAN PS_SERVERCATEGORY			1
		INLIST ITERATOR		0	0
142	103	INDEX RANGE SCAN PSAPSPRCSQUE			0
3	8	TABLE ACCESS BY INDEX ROWID PS_SERVERCLASS			0
2	1	INDEX RANGE SCAN PS_SERVERCLASS			0
		FILTER			0
3	1	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			0
2	5	INDEX RANGE SCAN PSDPSRCSQUE			0

Rows	Row Source	Operation	
0		SORT ORDER BY	(cr=2 r=0 w=0 time=12398 us)
0		FILTER	(cr=2 r=0 w=0 time=12374 us)
0		HASH JOIN	(cr=2 r=0 w=0 time=12371 us)
0		TABLE ACCESS BY INDEX ROWID PSPRCSQUE	(cr=2 r=0 w=0 time=12212 us)
2		NESTED LOOPS	(cr=2 r=0 w=0 time=12195 us)
1		TABLE ACCESS BY INDEX ROWID PS_SERVERCATEGORY	(cr=2 r=0 w=0 time=117 us)
1		INDEX RANGE SCAN PS_SERVERCATEGORY	(cr=1 r=0 w=0 time=76 us)
0		INLIST ITERATOR	(cr=0 r=0 w=0 time=23 us)
0		INDEX RANGE SCAN PSAPSPRCSQUE	(cr=0 r=0 w=0 time=3 us)
0		TABLE ACCESS BY INDEX ROWID PS_SERVERCLASS	
0		INDEX RANGE SCAN PS_SERVERCLASS	
0		FILTER	
0		TABLE ACCESS BY INDEX ROWID PSPRCSQUE	
0		INDEX RANGE SCAN PSDPSRCSQUE	

30 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

And, for the third time, the cost of the tuned query is lower, pointing yet again to a correlation between plan cost and execution speed.

That same observation is not always (generally?) true of SQL tuned with hints.



Example 1 tuned with Hints

```

/**+ index(R, UC_PSPRCSEQ_IX1) index(S, PS_PRCRECUR) use_nl(S,R) */
    
```

COST	CARD	operation	ELAPSED	ROWS	CR_GETS
672		SELECT STATEMENT			
672	54	NESTED LOOPS	0.010	0	111
529	142	TABLE ACCESS BY INDEX ROWID PSPRCSEQ	0.010	0	111
9	1,703	INDEX RANGE SCAN UC_PSPRCSEQ_IX1	0.010	112	48
2	1	TABLE ACCESS BY INDEX ROWID PS_PRCRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCRECUR	0.000	0	0

Compare to plan with histogram on psprcseq.runstatus


COST	CARD	operation	ELAPSED	ROWS	CR_GETS
14		SELECT STATEMENT			
14	1	NESTED LOOPS	0.010	0	112
12	2	TABLE ACCESS BY INDEX ROWID PSPRCSEQ			112
3	27	INDEX RANGE SCAN UC_PSPRCSEQ_IX1		112	49
2	1	TABLE ACCESS BY INDEX ROWID PS_PRCRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCRECUR			0

31 © Wolfgang Breitling, Centrex Consulting Corporation UKOUG October 30, 2005

The apparent correlation between plan costs and query speed prompted me to compare the TCF results to the results of tuning with hints and profiles.

Despite being the same plan, the one resulting from hints has a much higher cost than both the untuned, slower plan and especially the TCF tuned plan.

Clearly the correlation between cost and sql performance is broken when tuning with hints.



Example 1 with Profile

FINDINGS SECTION (2 findings)

1- SQL Profile Finding (see explain plans section below)
 A potentially better execution plan was found for this statement.
Recommendation (estimated benefit: 89.1%)
 Consider accepting the recommended SQL profile.

2- Using SQL Profile

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	92	47 (0)	00:00:01
1	NESTED LOOPS		1	92	47 (0)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	PSPRCSQUE	1	73	46 (0)	00:00:01
3	INDEX SKIP SCAN	PSAPSPRCSQUE	1		45 (0)	00:00:01
4	TABLE ACCESS BY INDEX ROWID	PS_PRCRSRECUR	1	19	1 (0)	00:00:01
5	INDEX UNIQUE SCAN	PS_PRCRSRECUR	1		0 (0)	00:00:01

ATTR ATTR_VALUE

```

1 OPT_ESTIMATE(@"SEL$1", TABLE, "R@"SEL$1", SCALE_ROWS=0.00664262176)
2 OPT_ESTIMATE(@"SEL$1", INDEX_FILTER, "R@"SEL$1", PSAPSPRCSQUE, SCALE_ROWS=0.0001556864475)
3 OPT_ESTIMATE(@"SEL$1", INDEX_SKIP_SCAN, "R@"SEL$1", PSBSPRCSQUE, SCALE_ROWS=2.784763486)
4 OPT_ESTIMATE(@"SEL$1", INDEX_FILTER, "R@"SEL$1", UC_PSPRCSQUE_IX1, SCALE_ROWS=6.021536625)
5 OPT_ESTIMATE(@"SEL$1", INDEX_FILTER, "R@"SEL$1", PSEPSRCSQUE, SCALE_ROWS=6.919397666e-005)
6 OPT_ESTIMATE(@"SEL$1", INDEX_SKIP_SCAN, "R@"SEL$1", UC_PSPRCSQUE_IX1, SCALE_ROWS=4.516152469)
7 OPT_ESTIMATE(@"SEL$1", INDEX_SKIP_SCAN, "R@"SEL$1", PSEPSRCSQUE, SCALE_ROWS=5.18954825e-005)
    
```

32
© Wolfgang Breitling, Centrex Consulting Corporation
UKOUG October 30, 2005

The dbms_sqltune exercise came up with a very similar plan to the one derived at by adding the histogram. This plan uses an index skip scan instead of the range scan on our custom index.

Interestingly enough, the recommendations did not include one for a histogram.

Note that the profile essentially does the same what the TCF method attempts to achieve – correct the cardinality estimates. Profiles do that by applying a scale factor to individual row sources rather than adjusting the base statistics. The result is obviously a more targeted and precise adjustment and one that has fewer possible side-effects.



The User Perspective

The tale behind the tuning exercise

or

What I found when I visited a user



References

- Berg, Martin. *Query Tuning by Eliminating Throwaway*. 2000.
<http://www.miracleas.dk/tools/throwaway2.pdf>.
- Breitling, W. (2003). *Fallacies of the Cost Based Optimizer*. Paper presented at the Hotsos Symposium on Oracle Performance, Dallas, Texas.
- Bruno, N. and S. Chaudhuri (2002). *Exploiting Statistics on Query Expressions for Optimization*. Paper presented at the ACM SIGMOD international conference on Management of data, Madison, Wisconsin.
- Christodoulakis, S. (1984). *Implications of Certain Assumptions in Database Performance Evaluation*. ACM Transactions on Database Systems (TODS), 9(2).
- Markl, V. and G. Lohman (2002). *Learning Table Access Cardinalities with LEO*. Paper presented at the ACM SIGMOD international conference on Management of data, Madison, Wisconsin.
- Millsap, C. and J. Holt (2003). *Optimizing Oracle Performance*. O'Reilly. ISBN: 0-596-00527-X.



My favorite websites

asktom.oracle.com	(Thomas Kyte)
integrid.info	(Tanel Pöder)
www.evdbt.com	(Tim Gorman)
www.go-faster.co.uk	(David Kurtz)
www.ixora.com.au	(Steve Adams)
www.jlcomp.demon.co.uk	(Jonathan Lewis)
www.julandyke.com	(Julian Dyke)
www.hotsos.com	(Cary Millsap)
www.miracleas.dk	(Mogens Nørgaard)
www.oracledba.co.uk	(Connor McDonald)
www.oraperf.com	(Anjo Kolk)
www.orapub.com	(Craig Shallahamer)
www.scale-abilities.com	(James Morle)

Wolfgang Breitling

breitliw@centrexcc.com

Centrex Consulting Corp.

www.centrexcc.com

