Tuning
by
Cardinality Feedback

Method and Examples

Wolfgang Breitling
www.centrexcc.com

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*

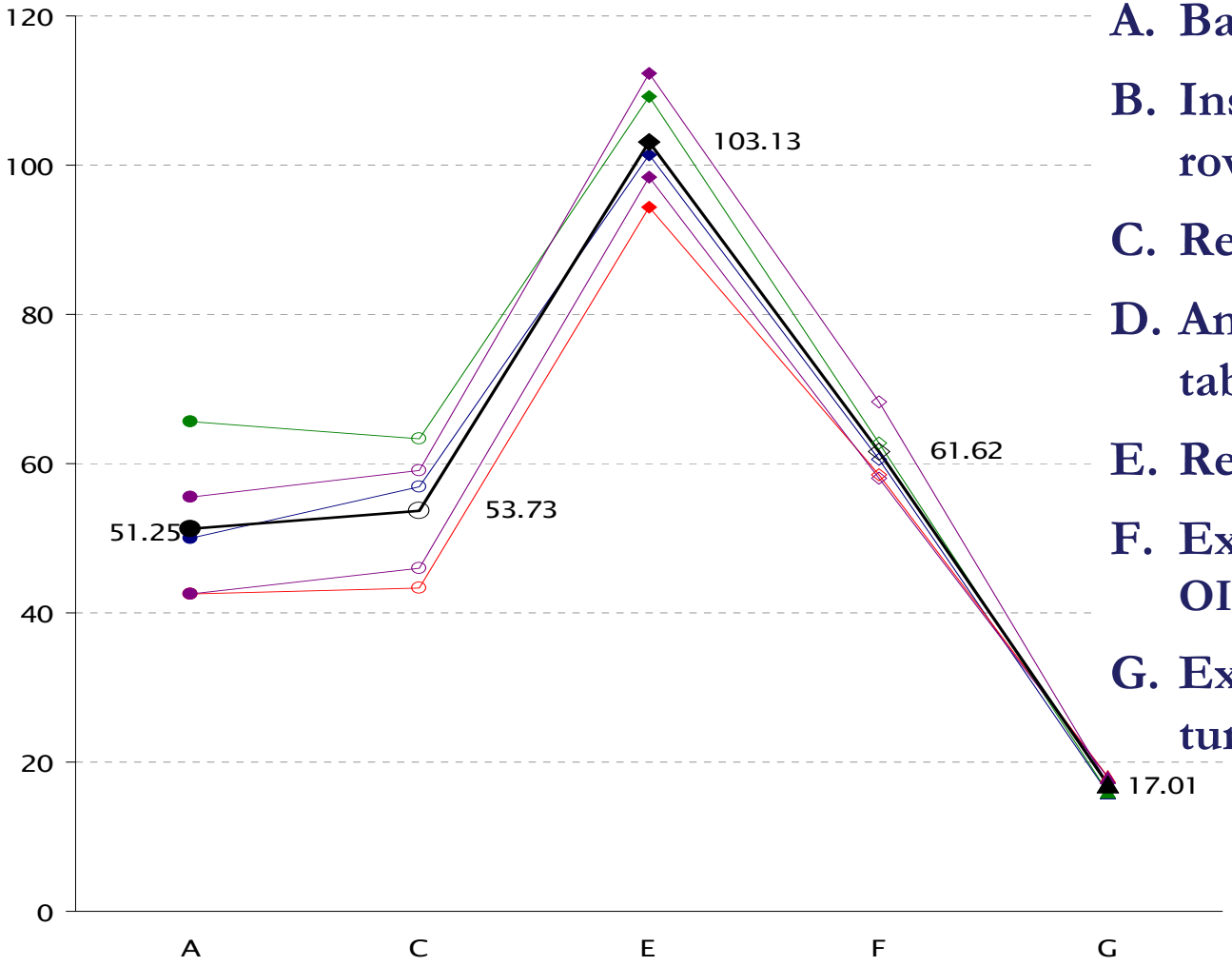
Oracle since 1993 (7.0.12)

Mathematics major from University of Stuttgart

Focus

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

SQL Performance Evolution

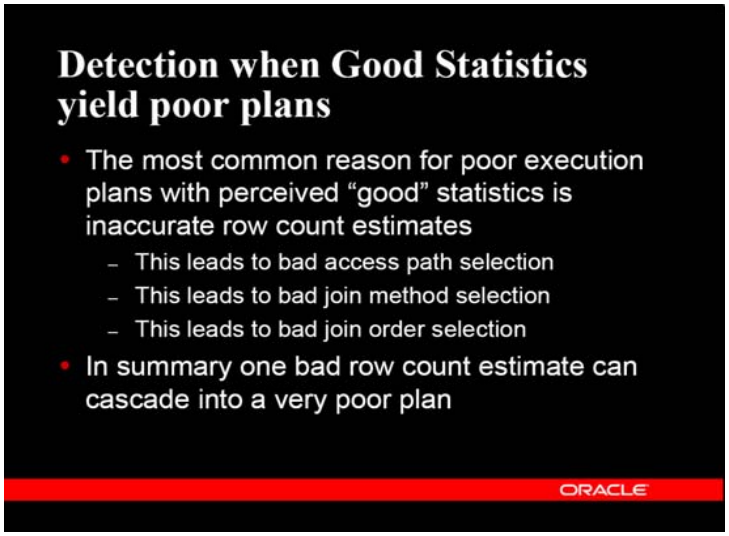


A. Baseline
B. Insert 1 row into 40,000 row table
C. Re-execute SQL
D. Analyze 40,000 row table
E. Re-execute SQL
F. Execute SQL with OICA=25, OIC=90
G. Execute SQL after TCF tuning

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.



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

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?
  Or, rather, why did it miscalculate the cardinality so badly?

Tuning by Cardinality Feedback

Once the answer to that question is found,
the next question is

What can I do to help/make the CBO
come to a more correct estimate

but ultimately

Butt out and let the CBO do its job
i.e. not try to tell the CBO what plan to use

Tuning by Cardinality Feedback

1. List the explain plan with the cardinality projections
   – from explain or, preferably, from v$sql_plan
2. Get the actual row counts
   – from a sql trace or from v$sql_plan_statistics.
   Make sure the actual plan is identical to the explain plan!
3. Look for the first (innermost) row source where the ratio of actual/estimated cardinality is orders of magnitude
   – usually at least in the 100s
4. 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

Demo Case SQL

```
SELECT A.COMPANY, A.PAYGROUP, E.OFF_CYCLE, E.SEPCHK_FLAG, E.TAX_METHOD
  , E.TAX_PERIODS, C.RETROPAY_ERNCD, sum(C.AMOUNT_DIFF)
from PS_PAY_CALENDAR A
  , WB_JOB B
  , WB_RETROPAY_EARNS C
  , PS_RETROPAY_RQST D
  , PS_RETROPAYPGM_TBL E
where A.RUN_ID = 'PD2'
  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_ERNCD
```

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

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

Adjust 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

Plan after Statistics Adjustment

	Rows	card	operation
		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

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.

Example 1

```
SELECT R.PRCSINSTANCE ,R.ORIGPRCSINSTANCE ,R.RECURORIGPRCSINST,
  R.MAINJOBINSTANCE ,R.PRCSJOBSEQ ,R.PRCSJOBNAME ,R.PRCSNAME ,R.PRCSTYPE, R.RECURNAME
FROM PSPRCSQUE R ,PS_PRCSRECUR S
WHERE ((R.RUNSTATUS IN (:"SYS_B_00", :"SYS_B_01") AND S.INITIATEWHEN = :"SYS_B_02")
 OR (R.RUNSTATUS IN (:"SYS_B_03", :"SYS_B_04", :"SYS_B_05", :"SYS_B_06",
:"SYS_B_07",:"SYS_B_08", :"SYS_B_09") AND S.INITIATEWHEN = :"SYS_B_10"))
 AND R.INITIATEDNEXT = :"SYS_B_11"
 AND R.OPSYS = :1
 AND R.RUNLOCATION = :"SYS_B_12"
 AND R.RECURNAME <>  :"SYS_B_13"
 AND R.PRCSJOBSEQ = :"SYS_B_14"
 AND R.SERVERNAMERUN = :2
 AND R.RECURNAME = S.RECURNAME
```

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_PRCSRECUR	0.000	15	3
214	34,057	TABLE ACCESS FULL PSPRCSQUE	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_PRCSRECUR (cr=3 r=0 w=0 time=171 us)
0	TABLE ACCESS FULL PSPRCSQUE (cr=2359 r=0 w=0 time=58218 us)

Example 2

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

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 PSPRCSQUE	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 PSPRCSQUE	(cr=2359 r=0 w=0 time=30341 us)
38539	TABLE ACCESS FULL PSPRCSPARMS	(cr=4563 r=0 w=0 time=77536 us)

Example 3

```
SELECT R.PRCSINSTANCE, R.ORIGPRCSINSTANCE, R.JOBINSTANCE, R.MAINJOBINSTANCE
, R.MAINJOBNAME, R.PRCSITEMLEVEL, R.PRCSJOBSEQ, R.PRCSJOBNAME, R.PRCSTYPE
, R.PRCSNAME, R.PRCSPRTY, TO_CHAR(R.RUNDTTM,:"SYS_B_00"), R.GENPRCSTYPE
, R.OUTDESTTYPE, R.RETRYCOUNT, R.RESTARTENABLED, R.SERVERNAMERQST, R.OPSYS
, R.SCHEDULENAME, R.PRCSCATEGORY, R.P_PRCSINSTANCE, C.PRCSPRIORITY
, S.PRCSPRIORITY, R.PRCSWINPOP, R.MCFREN_URL_ID
FROM PSPRCSQUE R, PS_SERVERCLASS S, PS_SERVERCATEGORY C
WHERE R.RUNDTTM <= 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.PRCSJOBSEQ = :"SYS_B_02" AND R.PRCSTYPE <> :"SYS_B_03")
   OR (R.PRCSJOBSEQ > :"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.PRCSJOBSEQ = :"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.RUNDTTM 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 PSPRCSQUE	0.040	0	2,359
		FILTER	0.000	0	0
3	1	TABLE ACCESS BY INDEX ROWID PSPRCSQUE			0
2	5	INDEX RANGE SCAN PSDPSPRCSQUE			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 PSPRCSQUE	(cr=2359 r=0 w=0 time=39040 us)
0	FILTER	
0	TABLE ACCESS BY INDEX ROWID PSPRCSQUE	
0	INDEX RANGE SCAN PSDPSPRCSQUE	

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	U	38,539		166	1	1	1	2,658
		PRCSINSTANCE	38,539	2.5948E-05	0		0	0	0

Tuning Example 1

Tuning Steps

1. Create an index on psprcsque

   ```
   create index uc_psprcsque_ix1 on psprcsque(prcsjobseq,recurname)
   ```

   But that did not change the plan

2. Create a (frequency) histogram on prcsjobseq

   That did not change the plan either

3. Modify the psprcsque.prcsjobseq statistics

   That finally did it - in conjunction with the index

Histogram Attempt

With index and histogram on prcsjobseq?

```
create index UC_PSPRCSQUE_IX1 on PSPRCSQUE (PRCSJOBSEQ, RECURNAME);
```

Table	column	NDV	density	nulls	lo	hi	av lg	bkts
PSPRCSQUE	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_PRCSRECUR	0.000	15	3
206	641	TABLE ACCESS FULL PSPRCSQUE	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_PRCSRECUR	(cr=3 r=0 w=0 time=132 us)
0	TABLE ACCESS FULL PSPRCSQUE	(cr=2359 r=0 w=0 time=35994 us)

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_PRCSRECUR	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_PRCSRECUR	

Statistics Attempt B

```
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_PRCSRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCSRECUR			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_PRCSRECUR	
0	INDEX UNIQUE SCAN PS_PRCSRECUR	

Tuning Example 2

Tuning Steps

1. There is a – reasonably usable – index on psprcsque

   But the optimizer doesn't use it

2. Create a (frequency) histogram on runstatus

   With the changed column statistics, the optimizer does use the index

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)

Tuning Example 3

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.

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 PSDPSPRCSQUE			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 PSDPSPRCSQUE	

Example 4

COST	CARD	operation	ROWS	ELAPSED	CR_GETS
144		SELECT STATEMENT			
144	1	VIEW	87	21.490	81,889,486
		FILTER	87	21.490	81,889,486
137	1	SORT GROUP BY	12,565	21.480	81,889,486
		FILTER	24,437	606.170	81,889,486
135	1	NESTED LOOPS	3,244,217	600.030	81,851,299
134	1	NESTED LOOPS	3,244,217	565.460	75,362,863
132	1	NESTED LOOPS	13,519,270	376.740	53,001,492
131	1	NESTED LOOPS	13,519,270	259.290	39,482,220
122	3	HASH JOIN	12,985,742	23.110	1,239
3	2	TABLE ACCESS FULL PS_RT_RATE_TBL	975	0.020	22
118	9	TABLE ACCESS FULL PS_CUST_CREDIT	34,676	0.390	1,217
3	1	TABLE ACCESS BY INDEX ROWID PS_CUSTOMER	13,519,270	194.890	39,480,981
2	1	INDEX RANGE SCAN PSBCUSTOMER	13,831,838	106.360	26,075,609
1	1	TABLE ACCESS BY INDEX ROWID PS_RT_INDEX_TBL	13,519,270	84.440	13,519,272
	1	INDEX UNIQUE SCAN PS_RT_INDEX_TBL	13,519,270	26.830	2
2	1	TABLE ACCESS BY INDEX ROWID PS_CUST_DATA	3,244,217	152.400	22,361,371
1	1	INDEX RANGE SCAN PSACUST_DATA	9,206,235	98.990	13,627,522
1	1	TABLE ACCESS BY INDEX ROWID PS_CUSTOMER	3,244,217	25.700	6,488,436
	1	INDEX UNIQUE SCAN PS_CUSTOMER	3,244,217	13.270	3,244,219
	1	SORT AGGREGATE	12,631	1.290	38,160
4	1	TABLE ACCESS BY INDEX ROWID PS_CUST_CREDIT	12,767	1.190	38,160
3	1	INDEX RANGE SCAN PS_CUST_CREDIT	12,771	0.340	25,378
	1	SORT AGGREGATE	4	0.040	27
3	2	NESTED LOOPS	749	0.040	27
2	1	TABLE ACCESS FULL PS_RT_INDEX_TBL	4	0.000	12
1	2	INDEX RANGE SCAN PS_RT_RATE_TBL	749	0.040	15

Tuning Example 4

Tuning Steps

1. Adjust the density of the effdt column of tables ps_rt_rate_tbl and ps_cust_credit:

```
set_column_stats(USER,'PS_CUST_CREDIT','EFFDT',density => 1);

set_column_stats(USER,'PS_RT_RATE_TBL','EFFDT',density => 1);
```

Tuned Example 4

COST	CARD	operation	ROWS	ELAPSED	CR_GETS
53825		SELECT STATEMENT			
53825	6,488	VIEW	87	17.720	49,408
		FILTER	87	17.720	49,408
8409	6,488	SORT GROUP BY	12,565	17.710	49,408
		FILTER	24,437	17.430	49,408
756	129,744	HASH JOIN	3,244,217	13.610	6,807
2	1	TABLE ACCESS FULL PS_RT_INDEX_TBL	1	0.000	3
748	129,744	HASH JOIN	3,244,217	7.570	6,804
3	975	TABLE ACCESS FULL PS_RT_RATE_TBL	975	0.030	22
742	3,726	HASH JOIN	24,578	2.800	6,782
570	2,651	HASH JOIN	36,097	1.980	5,212
492	2,651	HASH JOIN	36,097	1.210	4,724
338	5,267	TABLE ACCESS FULL PS_CUSTOMER	40,715	0.170	3,507
118	39,207	TABLE ACCESS FULL PS_CUST_CREDIT	34,676	0.210	1,217
48	42,133	INDEX FAST FULL SCAN PS0CUSTOMER	42,133	0.070	488
152	29,605	TABLE ACCESS FULL PS_CUST_DATA	29,609	0.040	1,570
	1	SORT AGGREGATE	14,091	0.890	42,574
4	1	TABLE ACCESS BY INDEX ROWID PS_CUST_CREDIT	14,229	0.860	42,574
3	1	INDEX RANGE SCAN PS_CUST_CREDIT	14,233	0.220	28,326
	1	SORT AGGREGATE	4	0.020	27
3	2	NESTED LOOPS	749	0.020	27
2	1	TABLE ACCESS FULL PS_RT_INDEX_TBL	4	0.000	12
1	2	INDEX RANGE SCAN PS_RT_RATE_TBL	749	0.020	15

Example 1 tuned with Hints

```
/*+ index(R, UC_PSPRCSQUE_IX1) index(S, PS_PRCSRECUR) 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 PSPRCSQUE	0.010	0	111
9	1,703	INDEX RANGE SCAN UC_PSPRCSQUE_IX1	0.010	112	48
2	1	TABLE ACCESS BY INDEX ROWID PS_PRCSRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCSRECUR	0.000	0	0

Compare to plan with histogram on psprcsque.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 PSPRCSQUE			112
3	27	INDEX RANGE SCAN UC_PSPRCSQUE_IX1		112	49
2	1	TABLE ACCESS BY INDEX ROWID PS_PRCSRECUR	0.000	0	0
1	1	INDEX UNIQUE SCAN PS_PRCSRECUR			0

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_PRCSRECUR	1	19	1 (0)	00:00:01
5	INDEX UNIQUE SCAN	PS_PRCSRECUR	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", PSBPSPRCSQUE, 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", PSEPSPRCSQUE, 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", PSEPSPRCSQUE, SCALE_ROWS=5.18954825e-005)

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 Põder)
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.juliandyke.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