Using DBMS_STATS in Access Path Optimization

Wolfgang Breitling
breitliw@centrexcc.com

Who am I

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

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 at University of Stuttgart

Who are YOU ?

DBA

Developer

Management

Oracle 9 R1 / R2

Oracle 8

Oracle 7 ??

Oracle 10 ??

Topics

- The DBMS_STATS Package
  - Oracle 8*i* vs. Oracle 9*i*
- Uses of DBMS_STATS beyond analyze
  - The "stattab" Table
  - Transporting Statistics
- Using DBMS_STATS to deliberately change an Access Plan
- Life Demo

DBMS_STATS

Procedures for

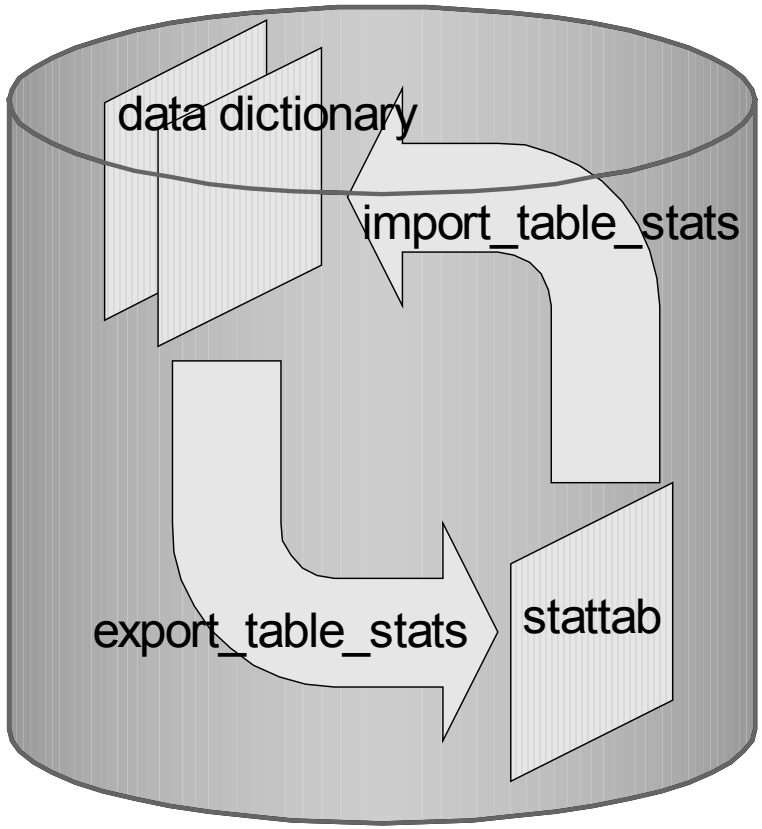
- Gathering Statistics
- Getting or Setting Statistics
- Transporting Statistics

Uses for the stattab table

- Backup / Rollback Changes in Statistics
- Transfer Statistics
  - Within same database
  - Between databases

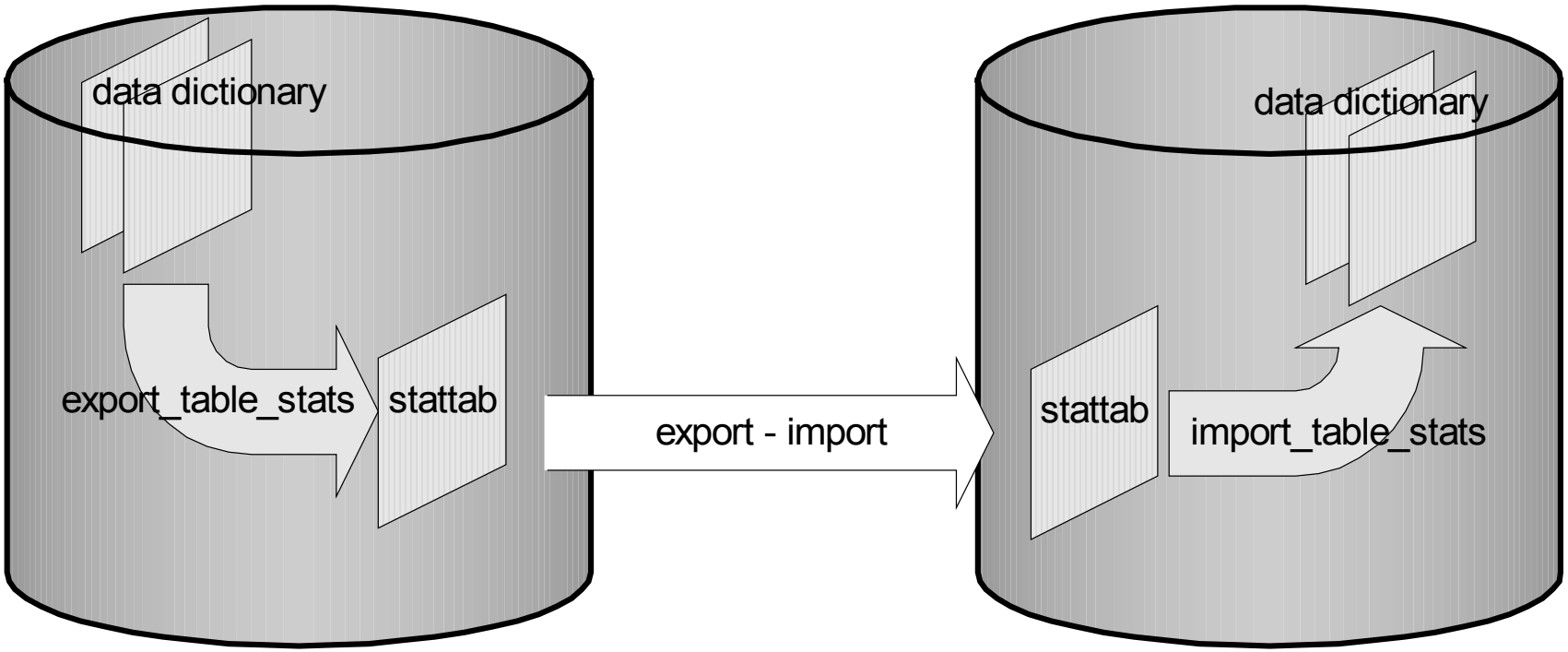
Transferring Statistics

within a database



Transferring Statistics

between databases



Transferring Statistics

Another way to transfer statistics is export and import:

```
ANALSTATS I "EMP"
BEGIN
  DBMS_STATS.SET_INDEX_STATS(NULL,  '"EMP_1"',  NULL,  NULL,  NULL,
    72996,  308,  72996,  1,  1,  15338,  1,  0);
END;

ANALSTATS T "EMP"
 BEGIN  DBMS_STATS.SET_TABLE_STATS(NULL,  '"EMP"',  NULL,  NULL,  NULL,
    72130,  895,  42,  0);
END;

ANALSTATS T "EMP"
DECLARE
 SREC DBMS_STATS.STATREC;
BEGIN
 SREC.MINVAL := 'C2022A'; SREC.MAXVAL := 'C3083925'; SREC.EAVS := 0;
 SREC.CHVALS := NULL; SREC.NOVALS := DBMS_STATS.NUMARRAY(141,  75636);
 SREC.BKVALS := DBMS_STATS.NUMARRAY(0,  1); SREC.EPC := 2;
 DBMS_STATS.SET_COLUMN_STATS(NULL,  '"EMP"',  '"EMPNO"',  NULL ,  NULL,
    NULL,  72130,  .00001386385692499 65,  0,  SREC,  4,  0);
END;
```

Transferring Statistics

Import will load the exported statistics even if

- the table already exists (ignore=Y)

or

- no data is imported (rows=N)

unless

- analyze = N

or

- recalculate statistics = Y

Transferring Statistics

From the Fine Manual:

In some cases, Export will place the precomputed statistics in the export file as well as the ANALYZE commands to regenerate the statistics.
However, the precomputed optimizer statistics will not be used at export time if:

- A table has indexes with system-generated names (including LOB indexes)
- A table has columns with system-generated names
- There were row errors while exporting
- The client character set or NCHARSET does not match the server character set or NCHARSET
- You have specified a QUERY clause
- Only certain partitions or subpartitions are to be exported
- Tables have indexes based upon constraints that have been analyzed (check, unique, and primary key constraints)
- Tables have indexes with system-generated names that have been analyzed (IOTs, nested tables, type tables that have specialized constraint indexes)

Note: Specifying ROWS=N does not preclude saving the precomputed statistics in the export file.

Transferring Statistics

Oracle 9*i* (9.2) relaxes the restrictions:

However, the precalculated optimizer statistics will not be used at export time if a table has columns with system-generated names.

The precalculated optimizer statistics are flagged as questionable at export time if:

- There are row errors while exporting
- The client character set or NCHAR character set does not match the server character set or NCHAR character set
- A QUERY clause is specified
- Only certain partitions or subpartitions are exported

analyze vs gather_table_stats

In Oracle 8*i* there is not much difference between dbms_stats and analyze, provided equivalent requests are used, since dbms_stats is using analyze under the covers for index statistics and histograms.

Note that

```
gather_table_stats('owner', 'table_name', NULL, estimate_pct)
```

is not equivalent to

```
analyze table owner.table_name {compute | estimate} statistics
```

The former does not gather statistics on indexes since CASCADE defaults to false.

gather_table_stats – Oracle 8*i* vs. 9*i*

Oracle 8*i*

- `DBMS_STATS.GATHER_TABLE_STATS(NULL, 't1'`
  `[, method_opt => 'FOR ALL COLUMNS SIZE 1'], cascade => TRUE);`

```
select /*+ */ count(*), count("PK1"), count(distinct "PK1")
  , sum(vsize("PK1")), min("PK1"), max("PK1")
  , count("PK2"), count(distinct "PK2")
  , sum(vsize("PK2")), min("PK2"), max("PK2")
  , count("D1"), count(distinct "D1")
  , 8, min("D1"), max("D1")
  , count("D2"), count(distinct "D2")
  , sum(vsize("D2")), min("D2"), max("D2")
  , count("D3"), count(distinct "D3")
  , sum(vsize("D3")), min(substrb("D3",1,32)), max(substrb("D3",1,32))
 from "SCOTT"."T1" t

analyze index "SCOTT"."T1P" COMPUTE statistics
```

gather_table_stats – Oracle 8*i* vs. 9*i*

Oracle 9*i*

- `DBMS_STATS.GATHER_TABLE_STATS(NULL, 't1'`
  `[, method_opt => 'FOR ALL COLUMNS SIZE 1'], cascade => TRUE);`

```
select /*+ */ count(*), count("PK1"), count(distinct "PK1")
  , sum(vsize("PK1")), min("PK1"), max("PK1")
  , count("PK2"), count(distinct "PK2")
  , sum(vsize("PK2")), min("PK2"), max("PK2")
  , count("D1"), count(distinct "D1")
  , 8, min("D1"), max("D1")
  , count("D2"), count(distinct "D2")
  , sum(vsize("D2")), min("D2"), max("D2")
  , count("D3"), count(distinct "D3")
  , sum(vsize("D3")), min(substrb("D3",1,32)), max(substrb("D3",1,32))
 from "SCOTT"."T1" t

select /*+ */ count(*) as nrw
  , count(distinct sys_op_lbid(43258,'L',t.rowid)) as nlb
  , count(distinct hextoraw(sys_op_descend("PK1")||sys_op_descend("PK2")))
   as ndk, sys_op_countchg(substrb(t.rowid,1,15),1) as clf
  from "SCOTT"."T1"  t where "PK1" is not null or "PK2" is not null
```

gather_table_stats – Oracle 8*i* vs. 9*i*

Oracle 8*i*

- ```
  DBMS_STATS.GATHER_TABLE_STATS(NULL, 't1'
    , method_opt => 'FOR ALL COLUMNS', cascade => TRUE);
  ```

  ```
  analyze table "SCOTT"."T1" COMPUTE statistics
    FOR TABLE
    FOR ALL INDEXES
    FOR ALL COLUMNS [ size 75 ]
  ```

- ```
  DBMS_STATS.GATHER_TABLE_STATS(NULL,'t1', method_opt => NULL);
  ```

  ```
  select /*+ */ count(*) from "SCOTT"."T1" t
  ```

gather_table_stats – Oracle 8*i* vs. 9*i*

Oracle 9*i*

- `DBMS_STATS.GATHER_TABLE_STATS(NULL, 't1'`
  `, method_opt => 'FOR ALL COLUMNS', cascade => TRUE);`

No Count(distinct), min( ), max( )

Frequency histogram

Height-balanced histogram

```
select  /*+ */ count(*), count("PK1"), sum(vsize("PK1")), count("PK2"),
  sum(vsize("PK2")), count("D1"), count("D2"), sum(vsize("D2")),
  count("D3"), sum(vsize("D3")) from "SCOTT"."T1" t
```

```
select substrb(dump(val,16,0,32),1,120) ep, cnt
  from (select /*+ */ "PK1" val,count(*) cnt from "SCOTT"."T1" t
  where "PK1" is not null group by "PK1" order by 1)
```

```
select min(minbkt),maxbkt,substrb(dump(min(val),16,0,32),1,120) minval
  , substrb(dump(max(val),16,0,32),1,120) maxval, sum(rep) sumrep
  , sum(repsq) sumrepsq, max(rep) maxrep, count(*) bktndv
  from (select val,min(bkt) minbkt, max(bkt) maxbkt, count(val) rep
  , count(val)*count(val) reps from (select /*+ */ "D1" val
  , ntile(75) over (order by "D1") bk from "SCOTT"."T1"
  where "D1" is not null) group by val) group by maxbkt order by maxbkt
```

...

Get index statistics as before

Automatisms

- options => '{ gather | list } stale'

  uses sys.mon_mods$
  which is maintained if monitoring is enabled for a table:

```
select u.name own, o.name tab
from mon_mods$ m, tab$ t, obj$ o, user$ u
where m.obj# = t.obj# and t.obj# = o.obj#
  and o.owner# = u.user# and bitand(t.flags,16) = 16
  and ((bitand(m.flags,1) = 1 )
     or ((m.inserts + m.updates  + m.deletes ) > (.1 * t.rowcnt ) ))
```

Automatisms

New in Oracle 9*i*

- method_opt => 'FOR COLUMNS SIZE { auto | skewonly | repeat }'

  uses sys.col_usage$ which is maintained by CBO during parsing

```
select  … , cu.timestamp cu_time, cu.equality_preds cu_ep
    , cu.equijoin_preds cu_ejp, cu.range_preds cu_rp, cu.like_preds cu_lp
    from user$ u, obj$ o, col$ c, col_usage$ cu, hist_head$ h
    where u.name = :b1 and o.name = :b2
      and c.obj# = cu.obj# (+) and c.intcol# = cu.intcol# (+) …
```

- estimate_percent => dbms_stats.auto_sample_size

```
select /*+ … */ count(*) from "SCOTT"."BIG_TABLE_16K" sample block (.001);
select /*+ … */ count(*) from "SCOTT"."BIG_TABLE_16K" sample block (.1);
select /*+ … */ count(*) from "SCOTT"."BIG_TABLE_16K" sample block (10);
```

- options => 'gather auto'

Oracle 10g parameters

```
select * from SYS.OPTSTAT_HIST_CONTROL$

SNAME                    SVAL1      SPARE1 SPARE4
---------------- ---------- ---------- ----------------------------
SKIP_TIME
STATS_RETENTION          31          1
TRACE                                1 0
CASCADE                              1 DBMS_STATS.AUTO_CASCADE
ESTIMATE_PERCENT                     1 DBMS_STATS.AUTO_SAMPLE_SIZE
DEGREE                               1 NULL
METHOD_OPT                           1 FOR ALL COLUMNS SIZE AUTO
NO_INVALIDATE                        1 DBMS_STATS.AUTO_INVALIDATE
GRANULARITY                          1 AUTO
AUTOSTATS_TARGET                     1 AUTO
```

Dave Ensor's dbms_stats paradox

it is only safe to gather statistics when to do so will make no difference

Live Demonstration

- Performance degradation after a table is analyzed

  "Never change a winning team"

  "If it ain't broke, don't fix it"

- The use of set_column_stats to "correct" the statistics and improve the performance

Using Plan Stability to Preserve Execution Plans

Plan stability prevents certain database environment changes from affecting the performance characteristics of your applications. Such changes include changes in optimizer statistics, changes to the optimizer mode settings, and changes to parameters affecting the sizes of memory structures, …

Means to changing access plan

- Change the statement
- Use hint
- Use Stored Outline
  - Change statistics
  - Create or drop an index
    - Change initialization parameters

Using DBMS_STATS

1. Overcome a deficiency of analyze and dbms_stats of partitioned tables in Oracle 8
2. Tune a SQL where the source is inaccessible

Case 1 – global histogram in 8*i*

analyze table tp1 compute statistics for columns n2;

table	column	NDV	density	nulls	lo	hi	av lg	bkts	G	U
TP1	N1	10,000	1.0000E-04	0	0	9999	3	1	N	N
	N2	3	3.3333E-01	0	0	999	2	1	N	N
	N3	1,000	1.0000E-03	0	0	999	3	1	N	N

table	column	NDV	density	nulls	lo	hi	av lg	bkts	G	U
TP1.P1	N1	1,000	1.0000E-03	0	0	999	3	1	N	N
	N2	3	4.9373E-05	0	0	999	2	2	N	N
	N3	1,000	1.0000E-03	0	0	999	3	1	N	N

table	column	EP	value
TP1	N2	0	0
TP1	N2	1	999

table	column	EP	value
TP1.P1	N2	10106	0
TP1.P1	N2	10118	998
TP1.P1	N2	10127	999

Case 1 – global histogram in 8*i*

select a.n1, b.n1, a.n3, a.d1, b.d1
from tp1 a, tp1 b
where a.n1 between 100 and 900 and a.n2 = 999
and a.n1 = b.n1 and a.n3 = b.n3

```
Rows     Row Source Operation
      9  NESTED LOOPS
     10   TABLE ACCESS BY LOCAL INDEX ROWID TP1 PARTITION: START=1 STOP=1
     10    INDEX RANGE SCAN PARTITION: START=1 STOP=1 (object id 101102)
      9   TABLE ACCESS BY LOCAL INDEX ROWID TP1 PARTITION: START=1 STOP=1
     90    INDEX RANGE SCAN PARTITION: START=1 STOP=1 (object id 101102)
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	10	0.00	0.00	0	0	0	0
Execute	11	0.00	0.00	0	0	0	0
Fetch	20	0.13	0.13	0	2090	0	90
total	41	0.13	0.13	0	2090	0	90

Case 1 – global histogram in 8*i*

```
select a.n1, b.n1, a.n3, a.d1, b.d1
from tp1 a, tp1 b
where a.n1 between 600 and 1400 and a.n2 = 999
and a.n1 = b.n1 and a.n3 = b.n3
```

```
Rows     Row Source Operation
     10  PARTITION RANGE ITERATOR PARTITION: START=1 STOP=2
     10   HASH JOIN
     10    TABLE ACCESS FULL TP1 PARTITION: START=1 STOP=2
   4017    TABLE ACCESS FULL TP1 PARTITION: START=1 STOP=2
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	10	0.00	0.00	0	0	0	0
Execute	10	0.00	0.00	0	0	0	0
Fetch	20	4.68	4.83	96260	100160	1390	100
total	40	4.68	4.83	96260	100160	1390	100

Backup, backup, backup

Before modifying* statistics, make a backup.

```
DBMS_STATS.EXPORT_TABLE_STATS (
  ownname => 'abc', tabname => 'xyz',
  stattab => 'stats_table', statid => 'bkup');
```

* And that includes analyze !

Case 1 – global histogram in 8*i*

```
DECLARE
   SREC DBMS_STATS.STATREC;
   NOVALS DBMS_STATS.NUMARRAY;
BEGIN

   SREC.EAVS := 0;
   SREC.CHVALS := NULL;
   SREC.EPC := 3;
   NOVALS := DBMS_STATS.NUMARRAY(0,998,999);

   SREC.BKVALS := DBMS_STATS.NUMARRAY(99943,33,25);

   DBMS_STATS.PREPARE_COLUMN_VALUES (SREC,NOVALS);

   DBMS_STATS.SET_COLUMN_STATS(NULL, 'TP1', 'N2', NULL, NULL,
       NULL, 3, .000049373, 0, SREC, 2, 2);

END;
```

```
select n2, count(0)
from tp1 group by n2;
```

N2	COUNT(0)
0	99943
998	33
999	25

Case 1 – global histogram in 8*i*

table	column	NDV	density	nulls	lo	hi	av lg	bkts	G	U
TP1	N1	10,000	1.0000E-04	0	0	9999	3	1	N	N
	N2	3	4.9373E-05	0	0	999	2	2	Y	N
	N3	1,000	1.0000E-03	0	0	999	3	1	N	N

table	column	EP	value
TP1	N2	99943	0
TP1	N2	99976	998
TP1	N2	100001	999

Rows	Row Source Operation
10	PARTITION RANGE ITERATOR PARTITION: START=1 STOP=2
10	NESTED LOOPS
12	TABLE ACCESS BY LOCAL INDEX ROWID TP1 PARTITION: START=1 STOP=2
12	INDEX RANGE SCAN PARTITION: START=1 STOP=2 (object id 101102)
10	TABLE ACCESS BY LOCAL INDEX ROWID TP1 PARTITION: START=1 STOP=2
110	INDEX RANGE SCAN PARTITION: START=1 STOP=2 (object id 101102)

call	count	cpu	elapsed	disk	query	current	rows
Parse	10	0.02	0.02	0	0	0	0
Execute	11	0.00	0.00	0	0	0	0
Fetch	20	0.15	0.24	65	2270	0	100
total	41	0.17	0.26	65	2270	0	100

Case 2 – Tuning by Cardinality Feedback

```
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
```

Tuning by Cardinality Feedback

card	cost	operation
2	617	SELECT STATEMENT
2	617	SORT GROUP BY
		FILTER
26	615	HASH JOIN
390	543	HASH JOIN
15	1	TABLE ACCESS FULL PS_RETROPAYPGM_TBL
52	541	NESTED LOOPS
40	221	HASH JOIN
5	5	TABLE ACCESS FULL PS_PAY_CALENDAR
13,334	208	TABLE ACCESS FULL WB_JOB
27,456	8	TABLE ACCESS BY INDEX ROWID WB_RETROPAY_EARNS
27,456	3	INDEX RANGE SCAN WB0RETROPAY_EARNS
13,679	20	TABLE ACCESS FULL PS_RETROPAY_RQST
1		SORT AGGREGATE
1	3	FIRST ROW
1	3	INDEX RANGE SCAN (MIN/MAX) WB_JOB
1		SORT AGGREGATE
1	3	FIRST ROW
1	3	INDEX RANGE SCAN (MIN/MAX) WB_JOB

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 off the mark.

Tuning by Cardinality Feedback

1. List the explain plan with the cardinality projections
2. Get the actual row counts from a sql trace or from v$sql_plan.
   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

Tuning by Cardinality Feedback

	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

Tuning by Cardinality Feedback

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

Tuning by Cardinality Feedback

	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

References

Breitling, W. (2003). *Fallacies of the Cost Based Optimizer*. Paper presented at the Hotsos Symposium on Oracle Performance, Dallas, Texas.

Bruno, N., & Chaudhuri, S. (2002). *Exploiting Statistics on Query Expressions for Optimization*. Paper presented at the ACM SIGMOD international conference on Management of data, Madison, Wisconsin.

Markl, V., & Lohman, G. (2002). *Learning Table Access Cardinalities with LEO*. Paper presented at the ACM SIGMOD international conference on Management of data, Madison, Wisconsin.

Stillger, M., Lohman, G., Markl, V., & Kandil, M. (2001). *LEO – DB2's LEarning Optimizer*. Paper presented at the 27th Intl. Conference on Very Large Data Bases (VLDB), Rome, Italy.

Christodoulakis, S. (1984). *Implications of Certain Assumptions in Database Performance Evaluation*. ACM Transactions on Database Systems (TODS), 9(2).

Tuning by Cardinality Feedback

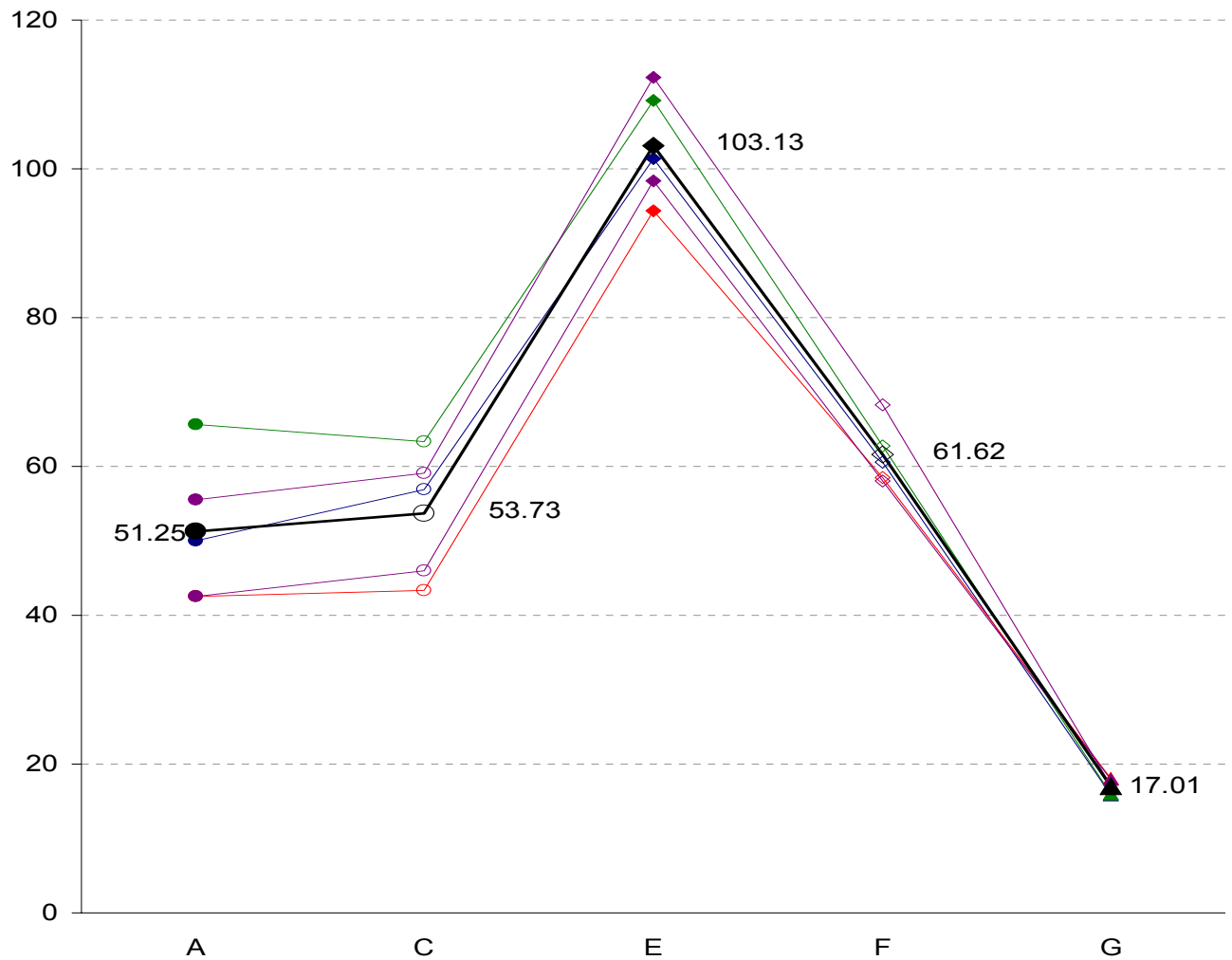
Conjecture

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

Demo Plan Timings



Demo Plan Profiles

NAME	C	E	F	G
LATCH.cache buffers chains	353,482	371,852	547,254	128,987
LATCH.multiblock read objects	948	948	850	1,232
LATCH.sort extent pool	18	61	6	25
STAT...CPU used by this session	267	617	320	188
STAT...buffer is not pinned count	110,977	110,977	233,992	10,326
STAT...buffer is pinned count	121,211	121,118	166,025	33,847
STAT...consistent gets	197,895	197,714	283,380	78,089
STAT...db block gets	83	83	72	170
STAT...physical reads	7,418	18,751	8,488	6,466
STAT...physical reads direct	1,604	12,678	17	2,770
STAT...session logical reads	197,978	197,797	283,452	78,259
STAT...sorts (disk)	1	1	1	1
STAT...sorts (memory)	11	11	11	11
STAT...sorts (rows)	14,124	14,124	14,124	14,124
STAT...table fetch by rowid	88,160	88,160	149,864	0
STAT...table scan blocks gotten	3,786	3,786	3,399	10,324
STAT...table scan rows gotten	55,974	55,974	40,015	220,707
STAT...table scans (long tables)	2	2	1	3
STAT...table scans (short tables)	2	2	1	2

Metalink Notes

114671.1	Gathering Statistics for the Cost Based Optimizer
130899.1	How to Set User-Defined Statistics Instead of RDBMS Statistics
122009.1	How to Retrieve Statistics Generated by ANALYZE SQL Statement
130688.1	Report Statistics for a Table, it's columns and it's indexes with DBMS_STATS
130911.1	How to Determine if Dictionary Statistics are RDBMS-Generated or User-Defined
102334.1	How to automate ANALYZE TABLE when changes occur on tables
1074354.6	DBMS_STATS.CREATE_STAT_TABLE: What Do Table Columns Mean?
117203.1	How to Use DBMS_STATS to Move Statistics to a Different Database
149560.1	Collect and Display System Statistics (CPU and IO) for CBO usage
153761.1	Scaling the system to improve CBO optimizer

Resources

See also Dave Ensor's session paper for IOUG 2002 and his presentation at UKOUG December 2003

asktom.oracle.com	**(Thomas Kyte)**
www.evdbt.com	**(Tim Gorman)**
www.hotsos.com	**(Cary Millsap)**
www.ixora.com.au	**(Steve Adams)**
www.jlcomp.demon.co.uk	**(Jonathan Lewis)**
www.miracleas.dk	**(Mogens Nørgaard)**
www.oraperf.com	**(Anjo Kolk)**
www.orapub.com	**(Craig Shallahamer)**

Wolfgang Breitling

breitliw@centrexcc.com

Centrex Consulting Corp.

www.centrexcc.com