Statistics in 11g

Jonathan Lewis

jonathanlewis.wordpress.com

www.jlcomp.demon.co.uk

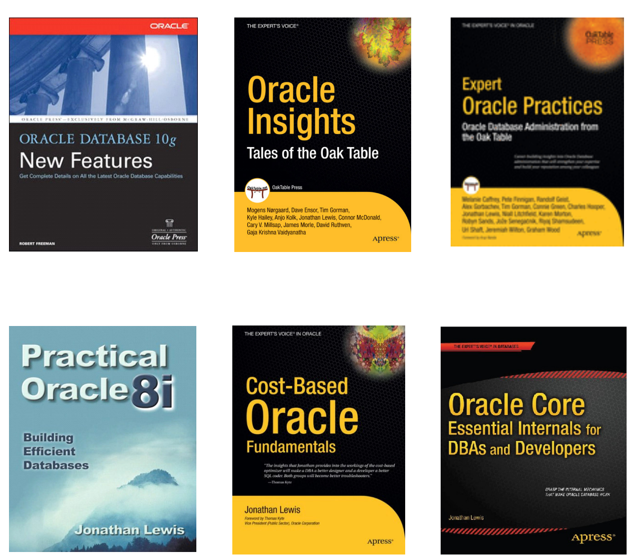
Who am I ?

Independent Consultant

28+ years in IT
24+ using Oracle

Strategy, Design, Review,
Briefings, Educational,
Trouble-shooting

Member of the Oak Table Network
Oracle *ACE Director*
Oracle author of the year 2006
Select Editor's choice 2007
UKOUG Inspiring Presenter 2011
ODTUG 2012 Best Presenter (d/b)
O1 visa for USA



Topics

- Auto_sample_size
  - Approximate NDV
  - Incremental partition stats
- Column groups
  - Shortcuts on indexes
  - Enhanced index stats

Collecting stats

```
procedure gather_table_stats(
   ownname  varchar2,    tabname varchar2,
   partname varchar2 default null,
   estimate_percent number default
        to_estimate_percent_type(get_param('ESTIMATE_PERCENT')),
   block_sample boolean default FALSE,
   method_opt   varchar2 default get_param('METHOD_OPT'),
   degree       number   default to_degree_type(get_param('DEGREE')),
   granularity  varchar2 default get_param('GRANULARITY'),
   cascade      boolean  default to_cascade_type(get_param('CASCADE')),
   stattab      varchar2 default null, statid varchar2 default null,
   statown      varchar2 default null,
   no_invalidate boolean default
        to_no_invalidate_type(get_param('NO_INVALIDATE')),
   stattype     varchar2 default 'DATA',
   force        boolean default FALSE
)
```

Suggestions

estimate_percent

- Use *auto_sample_size* from 11g
  - dbms_stats.set_global_prefs('approximate_ndv','true');
  - dbms_stats.set_param('approximate_ndv','true')
    - Need privileges "analyze any" and "analyze any dictionary"
    - Could execute as sys

method_opt

- For all columns size 1
  - (outside scope of this presentation)

Rationale for ***11g***

auto_sample_size

- Fast
- Accurate
- *Incremental on partitions*

Example - 1

```
create table orders as ...
select
        rownum                                  -- unique
        (sysdate - 25) + ((rownum-1)/(3*86400)) -- 2.16M
        trunc(dbms_random.value(1,1000000)),    -- 1M
        trunc(dbms_random.value(1, 500000)),    -- 0.5M
        trunc(dbms_random.value(1, 250000)),    -- 0.25M
        trunc(dbms_random.value(1, 125000)),    -- 0.125M
        trunc(100000 * dbms_random.normal),     -- lots
        trunc(100000 * exp(dbms_random.normal)),-- lots
        lpad(rownum,64,'0')
from ...
where
        rownum <= 86400 * 3 * 25   -- 3 rows per second, 25 days.
;                                  -- 6.48 Million rows
```


Example - 2

```
begin
   dbms_stats.gather_table_stats(
        ownname          => user,
        tabname          =>'ORDERS',
        estimate_percent => dbms_stats.auto_sample_size,
        method_opt       => 'for all columns size 1'
   );
end;
/

dbms_stats.set_param('APPROXIMATE_NDV','FALSE');
```

Sample 1: 8.5%	4 minutes 37 seconds	(auto selected)
Sample 2: 20%	7 minutes 3 seconds	(explicit sample size)

```
dbms_stats.set_param('APPROXIMATE_NDV','TRUE');
```

Result: 17 seconds

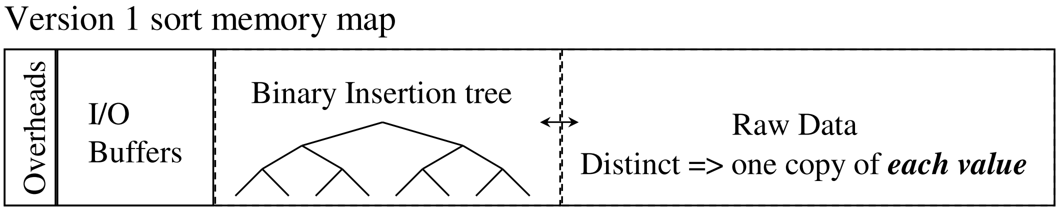
Example - 3 (false)

```
select  /*+ no_parallel(t) ... */
   count(*),
   count(distinct "ID"), sum(sys_op_opnsize("ID")),
   substrb(dump(min("ID"),16,0,32),1,120),
   substrb(dump(max("ID"),16,0,32),1,120),
   ...
   count(distinct "DATE_PLACED"), -- no sum(sys_op_opnsize)
   substrb(dump(min("DATE_PLACED"),16,0,32),1,120),
   substrb(dump(max("DATE_PLACED"),16,0,32),1,120),
   ...
   count(distinct "ID_PRODUCT"), sum(sys_op_opnsize("ID_PRODUCT")), ...
   count(distinct "N1"), sum(sys_op_opnsize("N1")), ...
   ...
   count(distinct "N5"), sum(sys_op_opnsize("N5")), ...
   count(distinct "PADDING"), sum(sys_op_opnsize("PADDING")), ...
from
   "TEST_USER"."ORDERS" sample (  8.5042889237) t
```


Count distinct (a)

- Uses a "version 1 sort"
- Appears to "sort" all columns in a single operation
  - does this prefix every column value with a column tag ?

Version 1 sort memory map



Binary tree:
The depth affects CPU
Each node is 3 pointers

e.g. 1M distinct values => depth 20

8 bytes per pointer

Count distinct (b)

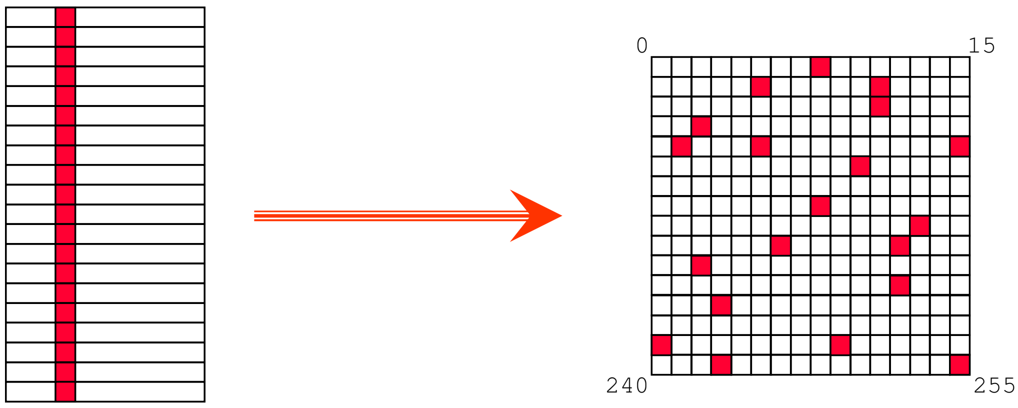
```
create table t1 as
with generator as ( ... )
select
       trunc(dbms_random.value(0,262144))       n_256K,
       trunc(dbms_random.value(0,131072))       n_128K,
       trunc(dbms_random.value(0,8192))         n_8k
from   generator v1, generator v2
where  rownum <= 8 * power(2,20);


select count(distinct n_8K)   from t1;          3.47 seconds
select count(distinct n_128K) from t1;          6.18 seconds
select count(distinct n_256K) from t1;          7.71 seconds
```


Example - 4 (true)

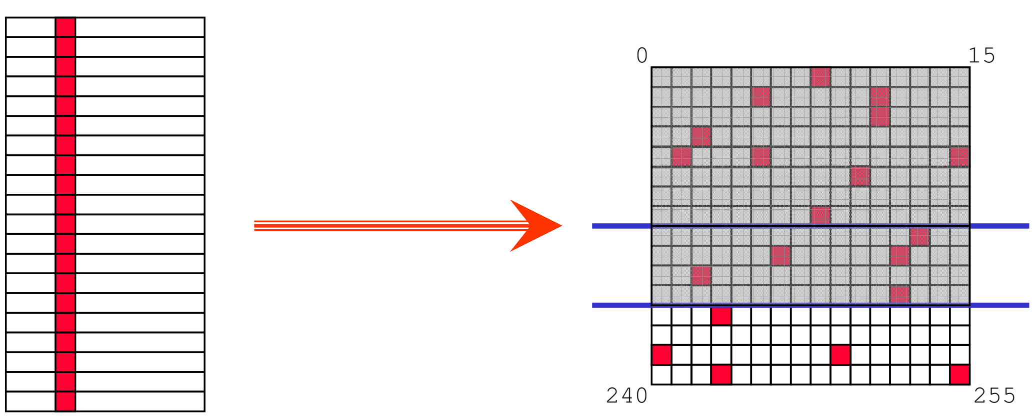
```
select /*+ full(t) no_parallel(t) ... */
   to_char(count("ID")),
   to_char(substrb(dump(min("ID"),16,0,32),1,120)),
   to_char(substrb(dump(max("ID"),16,0,32),1,120)),
   ...
   to_char(count("DATE_PLACED")),
   to_char(substrb(dump(min("DATE_PLACED"),16,0,32),1,120)),
   to_char(substrb(dump(max("DATE_PLACED"),16,0,32),1,120)),
   ...
   to_char(count("PADDING")),
   to_char(substrb(dump(min("PADDING"),16,0,32),1,120)),
   to_char(substrb(dump(max("PADDING"),16,0,32),1,120))
from "TEST_USER"."ORDERS" t  -- no sample clause
/*
   NDV,NIL,NIL, NDV,NIL,NIL,NDV,NIL,NIL, NDV,NIL,NIL, NDV,NIL,NIL,
   NDV,NIL,NIL, NDV,NIL,NIL, NDV,NIL,NIL,NDV,NIL,NIL
*/
```

Basic Principle



- The square is a visual aid only
- The number of hash buckets is 2^64 (= 10^19)

Minimising cost



- We only keep 16,384 items in the hash table for each column.
- We discard half the table each time we reach this limit

For visual simplicity the picture suggests we discard values based on the top N bits being zero, we actually discard values where the bottom N bits are one

Accuracy.

COLUMN_NAME	Approximate	Actual	Error	pct
ID	6,480,000	6,480,000	0	
DATE_PLACED	2,128,896	2,160,000	31,104	1.44%
ID_PRODUCT	***1,000,320***	998,453	1,867	0.19%
N1	506,528	499,999	6,529	1.3%
N2	253,072	249,999	3.073	1.23%
N3	126,096	124,999	1,097	0.88%
N4	545,088	546,115	1,027	0.19%
N5	747,008	743,297	3,711	0.50%
PADDING	6,480,000	6,480,000	0	

Note: 1,000,320 / 64 = 15,630 so we can deduce we did 6 splits for this column.

The hypothetical "best error" would be 64 (= 2 ^ 6)

Our error sub-optimal by a factor of 30.

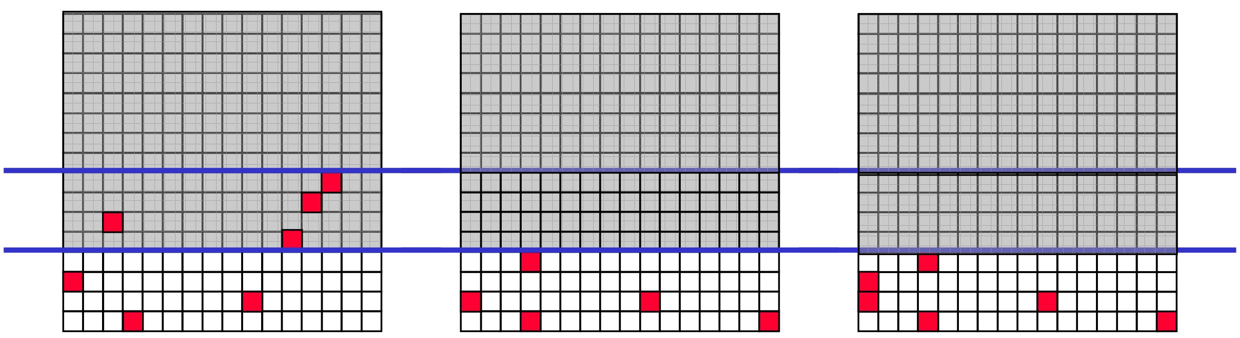
Incremental Partitions (a)

```
dbms_stats.set_table_prefs(
     ownname => 'TEST_USER',
     tabname => 'PT_COMPOSITE_1',
     pname   => 'INCREMENTAL',
     pvalue  => 'TRUE'
);

dbms_stats.gather_table_stats(
     ownname          => 'test_user',
     tabname          => 'pt_composite_1',
     granularity      => 'auto',   -- or 'all'
     estimate_percent => dbms_stats.auto_sample_size,
     method_opt       => 'for all columns size 1'

);
```

Incremental Partitions (b)



Incremental Partitions (c)

```
WRI$_OPTSTAT_SYNOPSIS_HEAD$
  BO#                    NOT NULL NUMBER   -- base object number
  GROUP#                 NOT NULL NUMBER   -- group of partitions
  INTCOL#                NOT NULL NUMBER   -- internal column number
  SYNOPSIS#              NOT NULL NUMBER   -- meaningless key
  SPLIT                           NUMBER
  ANALYZETIME                     DATE
  SPARE1                          NUMBER
  SPARE2                          CLOB

WRI$_OPTSTAT_SYNOPSIS$ (11.2)
  BO#                    NOT NULL NUMBER   (base object)
  GROUP#                 NOT NULL NUMBER   (group of partition)
  INTCOL#                NOT NULL NUMBER
  HASHVALUE              NOT NULL NUMBER
```

Incremental Partitions (d)

```
select  /*+ no_merge use_hash(sb s) */
        sb.intcol#,
        count(distinct(s.hashvalue)) * max(sb.maxsplit) ndv
from    (
                select /*+ no_merge */
                        t.intcol#, power(2, max(split)) maxsplit
                from    sys.wri$_optstat_synopsis_head$ t
                where   t.bo# = :tab_num
                group by
                        t.intcol#
        )                               sb,
        sys.wri$_optstat_synopsis$      s
where
        s.bo# = :tab_num
and     s.intcol# = sb.intcol#
and     mod(s.hashvalue + 1, sb.maxsplit) = 0
--              keep only if bottom N bits are all zero
group by
        sb.intcol#
```


Indexes (a)

```
select  /*+ {various hints} */
        count(*) as nrw,
        count(distinct sys_op_lbid(109360,'L',t.rowid)) as nlb,
        /*                      */
        count(distinct hextoraw(
                sys_op_descend("ID1") ||
                sys_op_descend("ID2") ||
                sys_op_descend("N1"))
        ) as ndk,
        sys_op_countchg(substrb(t.rowid,1,15),1) as clf
from
        "TEST_USER"."T1" t
where   "ID1" is not null
or      "ID2" is not null
or      "N1"  is not null
```

Indexes (b)

```
select
        /*+
        no_parallel_index(t, "PT1_I1")  dbms_stats
        cursor_sharing_exact use_weak_name_resl dynamic_sampling(0)
        no_monitoring no_substrb_pad no_expand index(t,"PT1_I1")
        */
        count(*) as nrw,
        count(distinct sys_op_lbid(66519,'L',t.rowid)) as nlb,
        null as ndk,
        sys_op_countchg(substrb(t.rowid,1,15),1) as clf
from
        "TEST_USER"."PT1" t
where
        "PT_COL" is not null
or      "SUBPT_COL" is not null
or      "ID" is not null
;       -- This example is a PRIMARY KEY, unique index.
```


Non-unique index (11.2a)

```
select
        /*+ {various hints} */
        count(*) as nrw,
        count(distinct sys_op_lbid(109360,'L',t.rowid)) as nlb,
        null as ndk,            -- HOW ?
        sys_op_countchg(substrb(t.rowid,1,15),1) as clf
from
        "TEST_USER"."T1" t
where   "ID1" is not null
or      "ID2" is not null
or      "N1"  is not null
;
```

So how did I make this happen ?
It's the same (non-unique) index as slide 20

Non-unique index (11.2b)

```
begin
	dbms_output.put_line(
		dbms_stats.create_extended_stats(
			ownname		=> user,
			tabname		=> 't1',
			extension	=> '(id1, id2, n1)'
		)
	);
end;
/
```


Non-unique index (11.2c)

```
begin
   dbms_stats.gather_table_stats(
	ownname		=> user,
	tabname		=>'T2',
	method_opt	=> 'for all hidden columns size 1'
--	method_opt	=> 'for columns (id1, id2, n1) size 1'
--	method_opt	=>
--		'for columns SYS_STUGE2F7LJEUJEE_QX_7ZVB3BH size 1'
   );
end;
/
```

Partial index stats

```
create index t1_i1 on (n1, n2, n3, n4);

execute :b1 := dbms_stats.create_extended_stats( -
                    ownname      => user, -
                    tabname      => 't1', -
                    extension    => '(n1, n2, n3)' -
              )

execute :b1 := dbms_stats.create_extended_stats( -
                    ownname      => user, -
                    tabname      => 't1', -
                    extension    => '(n1, n2)' -
              )
```


Summary

- Take advantage of approximate NDV for simple tables for faster, better stats.
- Be a little cautious, but consider using incremental partition stats
- Column groups may reduce the cost of index stats collection
- Create column groups (judiciously) to assist with "partial use" of indexes.