STATISTICS

DOING IT RIGHT, WHEN IT'S HARDER

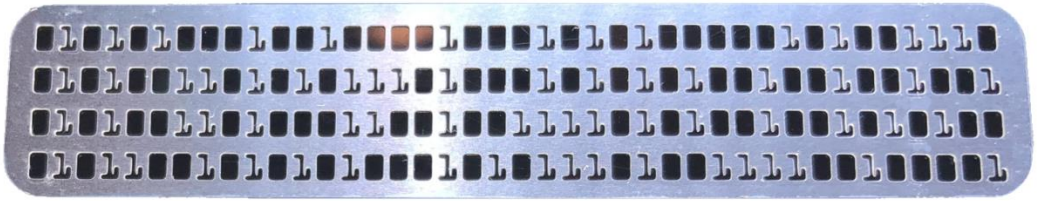
ITOUG
ITALIAN ORACLE USER GROUP

Neil Chandler
Chandler Systems

STATISTICS

DOING IT RIGHT, WHEN IT'S HARDER

Neil Chandler
Chandler Systems



Independent Database Consultant
Working in IT for over 30 years [WTF!]

ORACLE ACE Director

BLOG: http://chandlerdba.com
Twitter: **@chandlerDBA**
E: neil@chandler.uk.com

VIDEO NOT IN PDF

sorry :-/

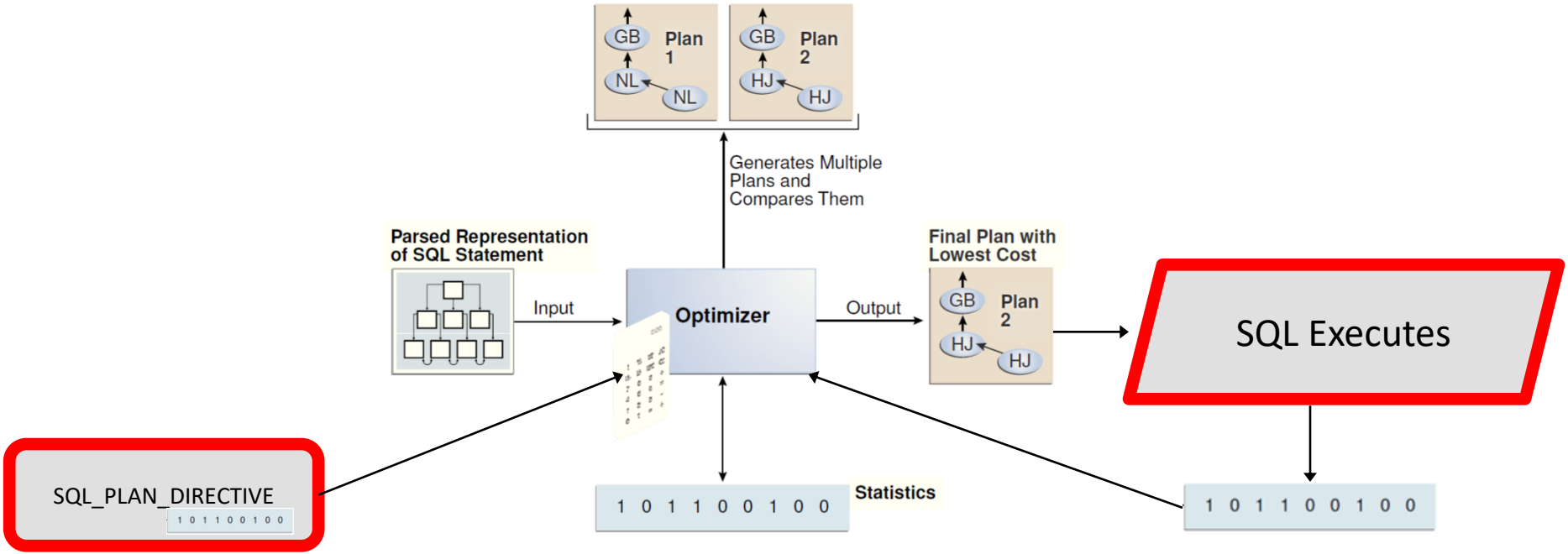
DISABLING AUTOSTATS JOB

Oracle automagically gathers your stats for you

If you gather your own, with your own job, don't disable the autostats job!

```
exec DBMS_STATS.SET_GLOBAL_PREFS (pname=>'AUTOSTATS_TARGET',pvalue=>'ORACLE' );
```

WHAT'S GOING ON?



WHAT ARE THESE STATISTICS ANYWAYS?

Extended Statistics

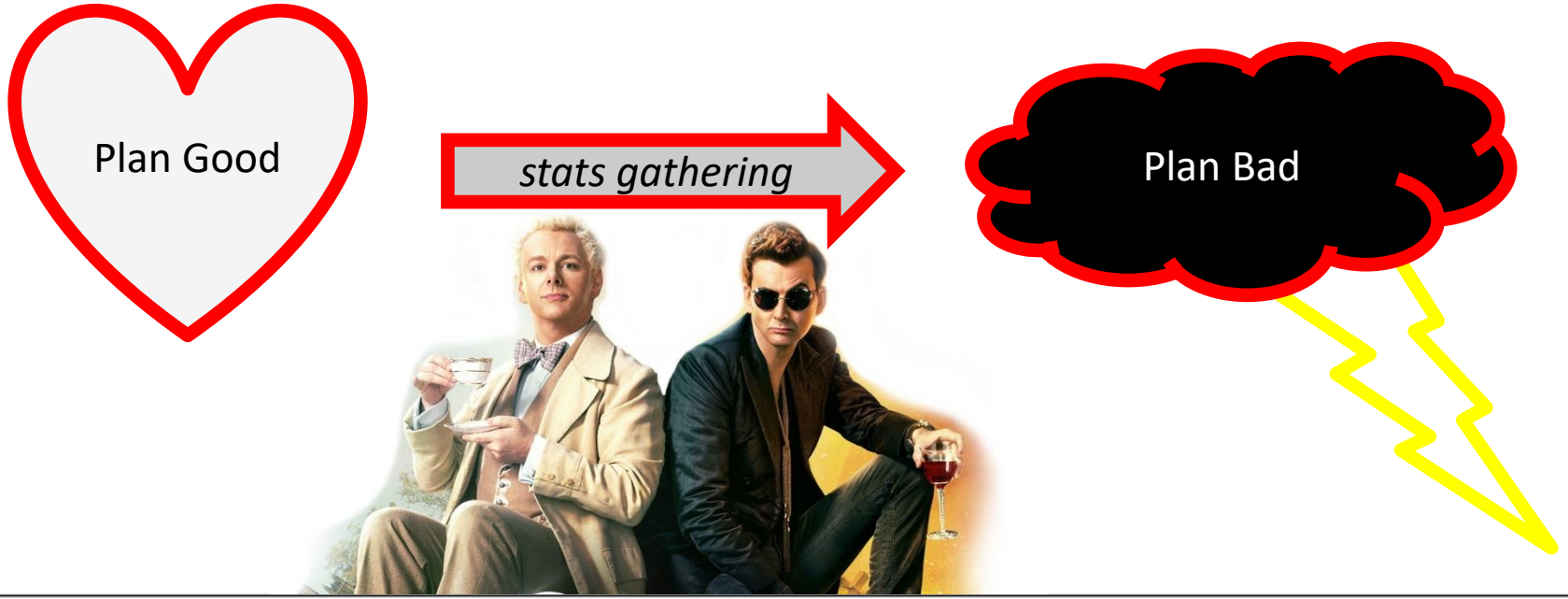
SPECIAL USE CASE:
If you are dropping a composite (multi-column) index, the optimizer will have been using the stats on the index, even if not using the index.
Creating extended stats will mitigate for plan changes

```
SH.CUSTOMERS.CUST_CITY           has 620 distinct values
SH.CUSTOMERS.CUST_STATE_PROVINCE has 145 distinct values

where cust_city = 'Brighton' and cust_state_province= 'East Sussex';

Estimated rows = (1/620 * 1/145) [0.0000111%] x 55,500 rows = 1 row

DBMS_STATS.CREATE_EXTENDED_STATS
('SH','CUSTOMERS','(cust_city,cust_state_province)') -- & gather stats

DBA_STAT_EXTENSIONS join DBA_TAB_COL_STATISTICS : 620 distinct values!
Estimated rows = (1/620)         [0.00161%] x 55,500 rows = 89 rows

Actual rows: 149
```

YESTERDAYS STATS

Yesterday, all my plans seem as fast as a jetway.
Now they've led my optimizer astray,
And I am filled with such dismay.
Suddenly, a table scan is all that I can see.
It used to use my favourite b-tree
Oh I believe, that the adaptive sample mechanism for histograms has gone wrong again.

Today's Plan Sucks!



YESTERDAYS STATS

Real-World Example (11.2.0.3)

select (*whatever*) from TICKETS where STATUS in (1,2) AND …..

```
Yesterday =  0.01s
Today     = 62.00s
```

DBA_TAB_COL_STATISTICS

TABLE	COLUMN_NAME	NUM_ROWS	SAMPLE_SIZE	HISTOGRAM	NUM_DISTINCT	LAST_ANALYZED
TICKETS	STATUS	22,000,000	198,000	FREQUENCY	1	last night

DBA_HISTOGRAMS

TABLE	COLUMN_NAME	ENDPOINT_NUMBER
TICKETS	STATUS	3

```
select status,count(*) from TICKETS group by STATUS order by 1;
```

STATUS	COUNT(*)
1	2,000
2	2,000
3	22,000,000

DBMS_STATS performing an "Adaptive Sample" (to get 5,500ish NOT NULL rows)
Another column has lots of NULLs, so Sample is upped to 198,000
Rare Values (1,2) chances of being in the sample?
(4,000/22000000) = 0.02% of the data contains these values
(198,000/22000000) = 0.90% of the data sampled so low chance you'll "hit" a rare value…

Frequency-Type Histograms are "fixed" in 12C for this. Hybrid/Height-Balanced still use Adaptive Sampling…

YESTERDAYS STATS

Real-World Example (11.2.0.3)

How to fix? Oracle retains 31 days of stats history…

DBA_TAB_STATS_HISTORY

TABLE	COLUMN_NAME	STATS_UPDATE_TIME
TICKETS	STATUS	*3 months ago*

```
select * from table(dbms_stats.diff_table_stats_in_history
                   (ownname => user,          tabname => 'TICKET',
                    time1   => systimestamp, time2   => to_timestamp('3 months ago','YYYY-MM-DD HH24:MI:SS'),
                    pctthreshold => 0));

COLUMN STATISTICS DIFFERENCE:
.............................
COLUMN_NAME     SRC NDV  HIST NULLS   LEN  MIN   MAX   SAMPSIZ
.....................................................................
STATUS          A   1    YES  0       3    C104  C104  198,000
                B   2    YES  0       3    C102  C104  180,000

dbms_stats.restore_table_stats(ownname=>'SCHEMA',tabname=>'TICKETS',AS_OF_TIMESTAMP=>'3 months ago')
dbms_stats.lock_table_stats    (ownname=>'SCHEMA',tabname=>'TICKETS');
```

And then, maybe, get the database upgraded to 19C before the "stale" stats become a problem!

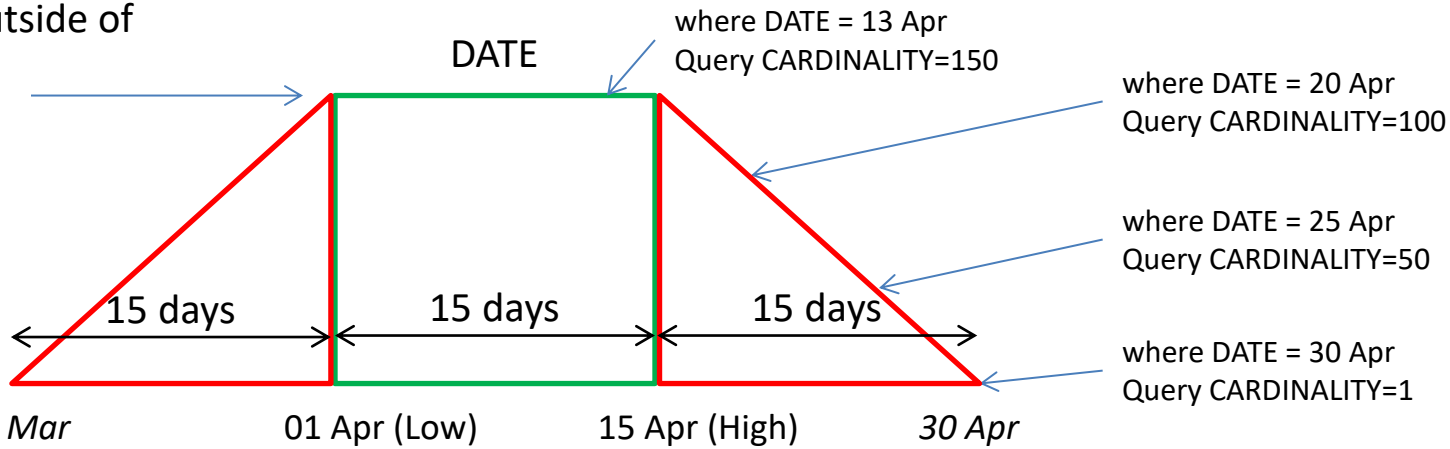
HIGH-LOW Value Threat

CARDINALITY reduces the further outside of the HIGH_VALUE - LOW_VALUE range in your predicate

CARDINALITY = How many rows oracle thinks it will return for your where clause

As your statistics get older, the risk of bad cardinality increases

Query Cardinality outside of 01-15 April reduced by *distance* from high/low range



A SLIGHTLY CLOSER LOOK

HIGH-LOW Threat

CARDINALITY reduces the further outside of the LOW_VALUE - HIGH_VALUE range in your predicate

SELECT … FROM DBA_TAB_COLUMNS

TABLE_NAME	COLUMN_NAME	DATA_TYPE	LOW_VALUE	HIGH_VALUE
PRODUCTS	PROD_CATEGORY	VARCHAR2	426F7973	576F6D656E
PRODUCTS	PROD_CAT_DESC	VARCHAR2	74686973206973207468	74686973206973207468
			652066616D6F75733A20	652066616D6F75733A20
			426F7973	576F6D656E
PRODUCTS	PROD_DESC	VARCHAR2	74686973206973207468	74686973206973207468
			652066616D6F75732031	652066616D6F7573205A
			2D506965636520436F7A	69702D46726F6E742053
			7920486F727365205061	7765617465722056657
			6A616D617320696E2063	7420696E20636F6C6F72
			6F6C6F7220626C61636B	204769726C73206F6620
			206F6620	73697A65
PRODUCTS	PROD_ID	NUMBER	C106	C306
PRODUCTS	PROD_LIST_PRICE	NUMBER	C104	C20D
PRODUCTS	PROD_MIN_PRICE	NUMBER	C10234	C20B0529
PRODUCTS	PROD_NAME	VARCHAR2	312D506965636520436F	5A69702D46726F6E7420
			7A7920486F7273652050	5377656174657220566
			616A616D6173	7374

Oracle Internal representation of the data

Need to be "decoded " using **`dbms_stats.convert_raw_value`**

this is trickier than it first seems as each data type needs to output into the correct… data type.

A SLIGHTLY CLOSER LOOK

HIGH-LOW Threat

CARDINALITY reduces the further outside of the LOW_VALUE - HIGH_VALUE range in your predicate

SELECT … FROM DBA_TAB_COLUMNS

TABLE_NAME	COLUMN_NAME	DATA_TYPE	LOW_VALUE	HIGH_VALUE	LOW_DECODE	HIGH_DECODE
PRODUCTS	PROD_CATEGORY	VARCHAR2	426F7973	576F6D656E	Boys	Women
PRODUCTS	PROD_CAT_DESC	VARCHAR2	74686973206973207468 652066616D6F75733A20 426F7973	74686973206973207468 652066616D6F75733A20 576F6D656E	this is the famous: Boys	this is the famous: Women
PRODUCTS	PROD_DESC	VARCHAR2	74686973206973207468 652066616D6F757320312D 506965636520436F7A 7920486F727365205061 6A616D617320696E2063 6F6C6F7220626C61636B 206F6620	74686973206973207468 652066616D6F7573205A 69702D46726F6E742053 7765617465722056657 3 7420696E20636F6C6F72 204769726C73206F6620 73697A65	this is the famous 1-Piece Cozy Horse Pajamas in color black of	this is the famous Zip-Front Swe ater Vest in color Girls of size
PRODUCTS	PROD_ID	NUMBER	C106	C306	5	50000
PRODUCTS	PROD_LIST_PRICE	NUMBER	C104	C20D	3	1200
PRODUCTS	PROD_MIN_PRICE	NUMBER	C10234	C20B0529	1.51	1004.4
PRODUCTS	PROD_NAME	VARCHAR2	312D506965636520436F 7A7920486F7273652050 616A616D6173	5A69702D46726F6E7420 5377656174657220566 5 7374	1-Piece Cozy Horse Pajamas	Zip-Front Sweater Vest

A SLIGHTLY CLOSER LOOK

HIGH-LOW Decode (including partitions) : https://chandlerdba.com/2015/11/13/decoding-dba_tab_columns-high_value-and-low_value/

```
WITH
FUNCTION raw_to_date(i_var in raw) return date  as
o_var date;
begin
dbms_stats.convert_raw_value(i_var,o_var);
return o_var;
end;
FUNCTION raw_to_number(i_var in raw) return number  as
o_var number;
begin
dbms_stats.convert_raw_value(i_var,o_var);
return o_var;
end;
FUNCTION raw_to_varchar2(i_var in raw) return varchar2  as
o_var varchar2(32767);
begin
dbms_stats.convert_raw_value(i_var,o_var);
return o_var;
end;
FUNCTION raw_to_float(i_var in raw) return binary_float  as
o_var binary_float;
begin
dbms_stats.convert_raw_value(i_var,o_var);
return o_var;
end;
FUNCTION raw_to_double(i_var in raw) return binary_double  as
o_var binary_double;
begin
dbms_stats.convert_raw_value(i_var,o_var);
return o_var;
end;
select utc.table_name,utc.column_name,utc.data_type,
upcs.partition_name,
decode(substr(utc.data_type,1,9),
'DATE'     , to_char(raw_to_date(utc.low_value),'YYYY-MM-DD HH24:MI:SS') ,
'TIMESTAMP', to_char(raw_to_date(utc.low_value),'YYYY-MM-DD HH24:MI:SS') ,
'NUMBER',    to_char(raw_to_number(utc.low_value)),
'VARCHAR2',  to_char(raw_to_varchar2(utc.low_value)),
'BINARY_FL', to_char(raw_to_float(utc.low_value)),
'BINARY_DO', to_char(raw_to_double(utc.low_value)),'Unknown') low_decode,
decode(substr(utc.data_type,1,9),
'DATE'     , to_char(raw_to_date(utc.high_value),'YYYY-MM-DD HH24:MI:SS') ,
'TIMESTAMP', to_char(raw_to_date(utc.high_value),'YYYY-MM-DD HH24:MI:SS') ,
'NUMBER',    to_char(raw_to_number(utc.high_value)),
'VARCHAR2',  to_char(raw_to_varchar2(utc.high_value)),
'BINARY_FL', to_char(raw_to_float(utc.high_value)),
'BINARY_DO', to_char(raw_to_double(utc.high_value)),'Unknown') high_decode,
decode(substr(utc.data_type,1,9),
'DATE'     , to_char(raw_to_date(upcs.low_value),'YYYY-MM-DD HH24:MI:SS') ,
'TIMESTAMP', to_char(raw_to_date(upcs.low_value),'YYYY-MM-DD HH24:MI:SS') ,
'NUMBER',    to_char(raw_to_number(upcs.low_value)),
'VARCHAR2',  to_char(raw_to_varchar2(upcs.low_value)),
'BINARY_FL', to_char(raw_to_float(upcs.low_value)),
'BINARY_DO', to_char(raw_to_double(upcs.low_value)),'Unknown') part_low_decode,
decode(substr(utc.data_type,1,9),
'DATE'     , to_char(raw_to_date(upcs.high_value),'YYYY-MM-DD HH24:MI:SS') ,
'TIMESTAMP', to_char(raw_to_date(upcs.high_value),'YYYY-MM-DD HH24:MI:SS') ,
'NUMBER',    to_char(raw_to_number(upcs.high_value)),
'VARCHAR2',  to_char(raw_to_varchar2(upcs.high_value)),
'BINARY_FL', to_char(raw_to_float(upcs.high_value)),
'BINARY_DO', to_char(raw_to_double(upcs.high_value)),'Unknown') part_high_decode
FROM user_tab_columns utc LEFT OUTER JOIN user_part_col_statistics upcs
  ON (utc.table_name = upcs.table_name AND utc.column_name = upcs.column_name)
order by utc.table_name,utc.column_name,upcs.partition_name;
```

PARTITIONS & SUBPARTITIONS

You probably have a lot of data

Which means a lot of data to read to get statistics

Which means a lot of resources & time

How does Oracle help with this?

some mornings it's just not worth chewing through the leather straps

PARTITIONS & SUBPARTITIONS

Gather Statistics Incrementally

You should be using INCREMENTAL Statistics (probably**)

Gather stats at a partition and/or subpartition level only (NOT Hash Subpartitions!)

This tells the stats job to

- gather statistics at a partition level
- gather SYNOPSES at a partition level
- aggregate statistics to a GLOBAL (table) level

**can be slower than a full gather if changing high % of partitions OR you have a LOT of partitions (100,000's)

Switch on INCREMENTAL stats for your partitioned tables:

```
dbms_stats.set_table_prefs('NEIL','MYDATA','INCREMENTAL','TRUE')
```

(NOTE #1: If you use OPTION=>'GATHER AUTO', it does not apply to 'INCREMENTAL' tables)
(NOTE #2: You only get Synopses at Partition level to roll-up to Global Stats, not at Subpartition level)

PARTITIONS & SUBPARTITIONS

Gather Statistics Incrementally - beware of default behaviour

You should be using INCREMENTAL Statistics

By default statistics are STALE when 10% of your data has changed

```
dbms_stats.set_table_prefs('NEIL','MYDATA','STALE_PERCENT',5)
```

ANY DML (1 row change) on your data in an INCREMENTAL partition will cause it to be regarded as STALE, and stats gathered [*NOTE: DBA_TAB_STATISTICS.STALE_STATS shows 'NO'!*]

From 12C, this can be controlled with INCREMENTAL_STALENESS

```
dbms_stats.set_table_prefs('NEIL','MYDATA','INCREMENTAL_STALENESS','USE_STALE_PERCENT')
```

If you LOCK the stats in partition and there is *ANY DML*, the stats become "unreliable" and the stats gather will stop running in INCREMENTAL mode. In 12C you can override this:

```
dbms_stats.set_table_prefs('NEIL','MYDATA','INCREMENTAL_STALENESS',
                                           'USE_STALE_PERCENT,USE_LOCKED_STATS')
```

PARTITIONS & SUBPARTITIONS

Other Reasons for Unexpected Partition Stats Gathers

DDL to add columns, including adding virtual columns and extended statistics

- regather stats for EVERY PARTITION

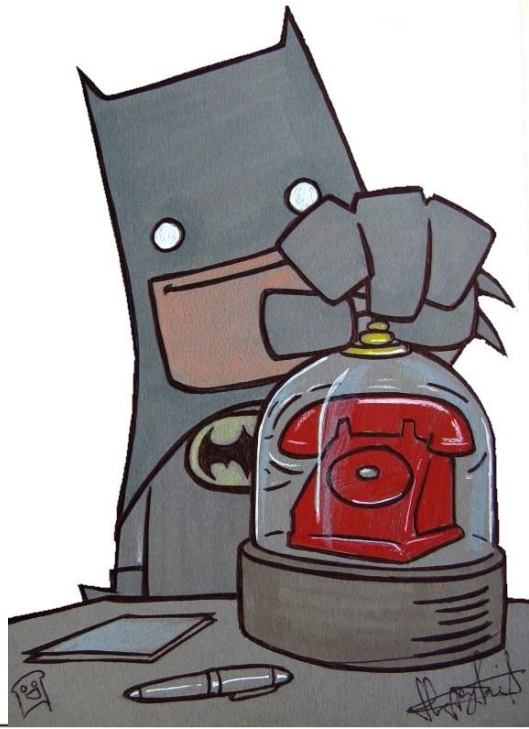
Someone messed with the statistics

- delete column statistics
- unlocked partition(s)
- did a stats gather in INCREMENTAL and then non-Incremental mode
  [the synopsis gather timestamp does not match the table/partition gather timestamp]

Table column usage has changed - some new SQL…

- **SYS.COL_USAGE$** tracks the use of predicates - where clauses - equalities, range scans, etc
- stats decides that, probably due to some new SQL, you need a new Histogram.
- regathers stats for EVERY PARTITION
- Seems like a very good use case to explicitly control your histograms!

```
[exec dbms_stats.set_table_prefs('NEIL','INTERVAL_TAB','METHOD_OPT',
'FOR ALL COLUMNS SIZE 1
 FOR COLUMNS SIZE 20   COL_MOD10
 FOR COLUMNS SIZE 2000 COL_MOD100,COL_MOD1000');]
```



What are they?

TABLE MYDATA [ID, PARTITION_COLUMN, CR_DATE, other stuff]

TABLE_NAME	COL	PARTITION	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
MYDATA	ID	PART01	**3**	1	3
MYDATA	CR_DATE	PART01	**2**	01-JAN-19	02-JAN-19
MYDATA	ID	PART02	**3**	4	6
MYDATA	CR_DATE	PART02	**2**	01-JAN-19	02-JAN-19
MYDATA	ID	PART03	**3**	7	9
MYDATA	CR_DATE	PART03	**2**	02-JAN-19	03-JAN-19
MYDATA	ID	*(global)*	**?**	1	9
MYDATA	CR_DATE	*(global)*	**?**	01-JAN-19	03-JAN-19

```
1, PART01, 01-JAN-19
2, PART01, 01-JAN-19
3, PART01, 02-JAN-19
```

```
4, PART02, 01-JAN-19
5, PART02, 02-JAN-19
6, PART02, 02-JAN-19
```

```
7, PART03, 02-JAN-19
8, PART03, 02-JAN-19
9, PART03, 03-JAN-19
```

Overall:
9 Distinct ID's
3 Distinct CR_DATE's

Cannot infer *Global* Number of Distinct Values from the Partitions

SYNOPSES 11.1 TO 12.1

TABLE_NAME	COL	PARTITION	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
MYDATA	ID	PART01	3	1	3
MYDATA	CR_DATE	PART01	2	01-JAN-19	02-JAN-19
MYDATA	ID	PART02	3	4	6
MYDATA	CR_DATE	PART02	2	01-JAN-19	02-JAN-19
MYDATA	ID	PART03	3	7	9
MYDATA	CR_DATE	PART03	2	02-JAN-19	03-JAN-19
MYDATA	ID	*(global)*	?	1	9
MYDATA	CR_DATE	*(global)*	?	01-JAN-19	03-JAN-19

SYS.WRI$_OPTSTAT_SYNOPSIS_HEAD$

TABLE_NAME	PARTITION_NAME	COLUMN_NAME	CREATED	BO#	GROUP#	INTCOL#
MYDATA	**PART01**	**CR_DATE**	**29-09-19**	**96471**	**192944**	**2**
MYDATA	PART01	COL_PART	29-09-19	96471	192944	3
MYDATA	**PART02**	**CR_DATE**	**29-09-19**	**96471**	**192946**	**2**
MYDATA	PART02	COL_PART	29-09-19	96471	192946	3
MYDATA	**PART03**	**CR_DATE**	**29-09-19**	**96471**	**192948**	**2**
MYDATA	PART03	COL_PART	29-09-19	96471	192948	3

SYS.WRI$_OPTSTAT_SYNOPSIS$;

BO#	GROUP#	INTCOL#	HASHVALUE	
96471	**192944**	**2**	**8230361653470229569**	**(#1)**
96471	**192944**	**2**	**15293244465252031082**	**(#2)**
96471	192944	3	14452862563684294119	
96471	**192946**	**2**	**8230361653470229569**	**(#1)**
96471	**192946**	**2**	**15293244465252031082**	**(#2)**
96471	192946	3	3287626035209197354	
96471	**192948**	**2**	**15293244465252031082**	**(#2)**
96471	**192948**	**2**	**7412358362535571944**	**(#3)**
96471	192948	3	4060662943914005775	

NOTE: No line for ID - it's a PK and therefore unique
If you read the HASHVALUE, there is a unique hash for every value in every column.
This table gets **big**, but quick to scan & compare.

SYNOPSES 12.2+

TABLE_NAME	COL	PARTITION	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
MYDATA	ID	PART01	3	1	3
MYDATA	CR_DATE	PART01	2	01-JAN-19	02-JAN-19
MYDATA	ID	PART02	3	4	6
MYDATA	CR_DATE	PART02	2	01-JAN-19	02-JAN-19
MYDATA	ID	PART03	3	7	9
MYDATA	CR_DATE	PART03	2	02-JAN-19	03-JAN-19
MYDATA	ID	*(global)*	?	1	9
MYDATA	CR_DATE	*(global)*	?	01-JAN-19	03-JAN-19

SYS.WRI$_OPTSTAT_SYNOPSIS_HEAD$

TABLE_NAME	PARTITION_NAME	COLUMN_NAME	CREATED	BO#	GROUP#	INTCOL#	SPLIT	SPARE1	SPARE2	SYNOPSIS_KB
MYDATA	PART01	ID	29-09-19	175780	351562	1	0	1	0D0C00E900003000000000	3.02148438
MYDATA	PART01	CR_DATE	29-09-19	175780	351562	2	0	1	0D0C07230002000000000	3.02148438
MYDATA	PART01	COL_PART	29-09-19	175780	351562	3	0	1	0D0C0C890001000000000	3.02148438
MYDATA	PART02	ID	29-09-19	175780	351564	1	0	1	0D0C02320003000000000	3.02148438
MYDATA	PART02	CR_DATE	29-09-19	175780	351564	2	0	1	0D0C07230002000000000	3.02148438
MYDATA	PART02	COL_PART	29-09-19	175780	351564	3	0	1	0D0C02D900010000000000	3.02148438
MYDATA	PART04	ID	29-09-19	175780	351566	1	0	1	0D0C00440003000000000	3.02148438
MYDATA	PART04	CR_DATE	29-09-19	175780	351566	2	0	1	0D0C066D0002000000000	3.02148438
MYDATA	PART04	COL_PART	29-09-19	175780	351566	3	0	1	0D0C03850001000000000	3.02148438

SYS.WRI$_OPTSTAT_SYNOPSIS$;

(table is empty)

SPARE2 (BLOB) has been repurposed to store the synopses.
Much smaller: 3-5% of previous size (e.g. 750GB => 30GB)
From 12.2 is uses the ***HyperLogLog*** algorithm
(single pass scan to generate NDV)

SYNOPSES WHEN UPGRADING

If you are upgrading to Oracle 12.2+ (19C!), you have choices!

```
DBMS_STATS.SET_TABLE_PREFS('schema','table','APPROXIMATE_NDV_ALGORITHM','options')
```

- REPEAT OR HYPERLOGLOG (default) - Keep doing it the old way, but use HLL for new tables
- ADAPTIVE SAMPLING - don't use the new synopses format anywhere
- HYPERLOGLOG - gather HLL and have a mix of formats in a table

If you also:

```
DBMS_STATS.SET_TABLE_PREFS('schema','table','APPROXIMATE_NDV_ALGORITHM','HYPERLOGLOG')
DBMS_STATS.SET_TABLE_PREFS('schema','table','INCREMENTAL_STALENESS','NULL') <-NULL is in quotes!
```

Forces a re-gather of all partitions in HLL format (default is **ALLOW_MIXED_FORMAT**)

SYNOPSES WHEN EXCHANGING PARTITIONS

In 12C, statistics generation for partition exchange got smarter, if you do it right!

Previously, you had no synopsis for a partition until it was gathered *as a partition.*

- create your LOAD table (e.g. MYDATA_LOAD) matching the partition and load the data
- create the empty partition for the swap
- `.set_table_prefs('NEIL','MYDATA_LOAD','INCREMENTAL','TRUE')`
  `.set_table_prefs('NEIL','MYDATA_LOAD','INCREMENTAL_LEVEL','TABLE')`
- add indexes / extended stats on the LOAD table to match the partitioned table
- gather table stats on the LOAD table. Ensure the **METHOD_OPT** is set such that you get *exactly* the same histograms as on the partitioned table

  You will need to be explicit as **sys.col_usage$** will be empty for the LOAD table -
  "*for all columns size auto*" won't work
  `"for all columns size 1 for columns size 254 ID,CR_DATE,etc.."`

I go from stool to stool in singles bars hoping to get lucky, but there's never any gum under any of them

SYNOPSES WHEN EXCHANGING PARTITIONS

```
TABLE_NAME           STATUS   LAST_ANALYZED     GLO   NUM_ROWS
-------------------- -------- ----------------- --- ----------
MYDATA               VALID    07-11-19 05:14:31 YES         15

TABLE_NAME           PARTITION_ STALE_STATS
-------------------- ---------- -----------
MYDATA                          NO
MYDATA               PART01     NO
MYDATA               PART02     NO
MYDATA               PART03     NO
MYDATA               PART_D     NO

SYNOPSES_BEFORE
TABLE_NAME           PARTITION_ COLUMN_NAME     CREATED_DATE
-------------------- ---------- --------------- -----------------
MYDATA               PART01     ID              07-11-19 05:14:31
MYDATA               PART01     CR_DATE         07-11-19 05:14:31
MYDATA               PART01     COL_PART        07-11-19 05:14:31
MYDATA               PART02     ID              07-11-19 05:14:31
MYDATA               PART02     CR_DATE         07-11-19 05:14:31
MYDATA               PART02     COL_PART        07-11-19 05:14:31
etc…
```

```
alter table MYDATA exchange partition PART01 with
      table MYDATA_LOAD without validation update global indexes;
```


I go from stool to stool in singles bars hoping to get lucky, but there's never any gum under any of them

STATS WHEN SPLITTING PARTITIONS

If we split a partition, the stats are deleted
(e.g. split the DEFAULT partition "PART_D" in MYDATA into a new PART04 and retain the DEFAULT partition)

```
ALTER TABLE neil.mydata
      SPLIT PARTITION PART_D VALUES ('PART04')
      INTO
       ( PARTITION PART04,
         PARTITION PART_D);
```

We lose the stats in PART_D and both PART04 and PART_D will have no stats following the split

TABLE_NAME	PARTITION_NAME	LAST_ANALYZED	GLO	USE	NUM_ROWS	BLOCKS	EMPTY_BLOCKS	AVG_ROW_LEN
MYDATA	PART01	30-09-19 14:53:29	YES	NO	3	124	0	18
MYDATA	PART02	30-09-19 14:53:29	YES	NO	3	124	0	18
MYDATA	PART03	30-09-19 14:53:29	YES	NO	3	124	0	18
MYDATA	PART04		NO	NO				
MYDATA	PART_D		NO	NO				

STATS WHEN SPLITTING PARTITIONS

UNLESS the split moves EVERY value in the DEFAULT partition to the new partition

This is known as a Fast Split Partition, which is more like a rename and Partition ADD (e.g. a new DEFAULT with LIST partitions)

```
ALTER TABLE neil.mydata
      SPLIT PARTITION PART_D VALUES ('PART04','PART05')
      INTO
       ( PARTITION PART04_05,
         PARTITION PART_D);
```

NOTE: The SYNOPSIS is NOT copied!
Stats DO NOT show as STALE but will be gathered next time

TABLE_NAME	PARTITION_NAME	LAST_ANALYZED	GLO	USE	NUM_ROWS	BLOCKS	EMPTY_BLOCKS	AVG_ROW_LEN
MYDATA	PART01	30-09-19 15:52:27	YES	NO	3	124	0	18
MYDATA	PART02	30-09-19 15:52:27	YES	NO	3	124	0	18
MYDATA	PART03	30-09-19 15:52:27	YES	NO	3	124	0	18
MYDATA	PART04_05	30-09-19 15:52:27	YES	NO	**6**	**124**	**0**	**18**
MYDATA	PART_D	30-09-19 15:52:27	YES	NO	**6**	**124**	**0**	**18**

See how the new partition PART04_05 gets STATS from the DEFAULT partition but the OLD partition doesn't lose its stats.

SYNOPSIS (SQL)

```
SELECT o.name          TABLE_NAME,
       p.subname       PARTITION_NAME,
       c.name          COLUMN_NAME,
       h.analyzetime   CREATED_DATE,
       h.BO# ,
       h.GROUP#,
       h.INTCOL#,
       h.SYNOPSIS#,
       h.SPLIT,
       h.SPARE1,
       DBMS_LOB.GetLength("H"."SPARE2")/1024 AS Synopsis_KB
FROM   SYS.WRI$_OPTSTAT_SYNOPSIS_HEAD$ h,
       SYS.OBJ$ o,
       SYS.USER$ u,
       SYS.COL$ c,
       ( ( SELECT TABPART$.bo#  BO#,
                  TABPART$.obj# OBJ#
           FROM   SYS.TABPART$ tabpart$ )
         UNION ALL
         ( SELECT TABCOMPART$.bo#  BO#,
                  TABCOMPART$.obj# OBJ#
           FROM   SYS.TABCOMPART$ tabcompart$ ) ) tp,
       SYS.OBJ$ p
WHERE  u.name = 'NEIL' AND
       o.name = '&&TABLE' AND
       tp.obj# = p.obj# AND
       h.bo# = tp.bo# AND
       h.group# = tp.obj# * 2 AND
       h.bo# = c.obj#(+) AND
       h.intcol# = c.intcol#(+) AND
       o.owner# = u.user# AND
       h.bo# = o.obj#
ORDER  BY 4,1,2,3;
```

```
--(and Before 12.2)
select * from SYS.WRI$_OPTSTAT_SYNOPSIS$;
```

NEW PARTITIONS

New Partitions are Empty, with no stats

Have larger stats rather than smaller stats - an incorrect HJ is "better" than an incorrect NL Join

Consider SCALE_FACTOR to make them larger

Manual Partitions: copy the stats from (usually your largest, or most recent) partition

```
dbms_stats.copy_table_stats
```

Split Partitions: gather your stats - consider if you need to copy stats to a partition if you have moved all values from it (just gather where you have a representative data set in place)

Interval Partitions: Appear "magically". You can't catch the DDL with a trigger…

- pre-create expected partition with LOCK TABLE PARTITION or INSERT…. then ROLLBACK and copy stats

or

- run a regular job looking for stale or empty stats on the table
  - High-Frequency Automatic Optimizer Statistics Collection (19C Cloud/Exadata only)

FREQUENTLY GATHERING STATS

Home-Cooked "High-Frequency Automatic Optimizer Statistics Collection"

Gather Empty/Stale stats - need to gather schema stats with 'gather auto'

dbms_stats.**gather_table_stats**('NEIL', 'INTERVAL_TAB',options=>'gather auto')

'gather auto' a table level is NOT for identifying stale state - it is to enhance "online" stats from Bulk Data Loads

… but you might not want to run a full GATHER_SCHEMA_STATS regularly…

FREQUENTLY GATHERING STATS

Home-Cooked "High-Frequency Automatic Optimizer Statistics Collection"

Gather Empty/Stale stats - need to gather schema stats with 'gather auto'

Use **dbms_stats.gather_schema_stats** but filtering

```
DECLARE
   l_filter_array  dbms_stats.objecttab := dbms_stats.objecttab();
BEGIN
   l_filter_array.extend(2);
   l_filter_array(1).ownname := 'NEIL';
   l_filter_array(1).objname := 'INTERVAL_TAB';
   l_filter_array(2).ownname := 'NEIL';
   l_filter_array(2).objname := 'MYDATA';

   dbms_stats.gather_schema_stats(
           ownname         => 'NEIL',
           obj_filter_list=> l_filter_array,
           options         => 'gather auto');
END;
```

FREQUENTLY GATHERING STATS

Home-Cooked "High-Frequency Automatic Optimizer Statistics Collection"

```
DECLARE
    l_filter_array dbms_stats.objecttab := dbms_stats.objecttab ();
    l_from_partition dba_tab_statistics.partition_name%TYPE;
BEGIN
   --Get Largest Partition to copy from
    SELECT partition_name INTO l_from_partition
    FROM ( SELECT partition_name
             FROM dba_tab_statistics dts
            WHERE dts.owner = 'NEIL' AND dts.table_name = 'INTERVAL_TAB' AND dts.partition_name IS NOT NULL AND stale_stats = 'NO'
            ORDER BY num_rows DESC NULLS LAST )
    WHERE ROWNUM = 1;

   -- Loop round all empty partitions and copy stats
    FOR empty_partition IN ( SELECT partition_name
                             FROM dba_tab_statistics dts
                             WHERE dts.owner = 'NEIL'            AND dts.table_name = 'INTERVAL_TAB'
                              AND dts.partition_name IS NOT NULL AND dts.stale_stats IS NULL )
    LOOP
        dbms_stats.copy_table_stats(ownname     => 'NEIL',           tabname     => 'INTERVAL_TAB',
                                    srcpartname => l_from_partition, dstpartname => empty_partition.partition_name);
    END LOOP;

    -- And gather for any stale partitions
    l_filter_array.extend(2);
    l_filter_array(1).ownname := 'NEIL';
    l_filter_array(1).objname := 'INTERVAL_TAB';
    l_filter_array(2).ownname := 'NEIL';
    l_filter_array(2).objname := 'MYDATA';
    dbms_stats.gather_schema_stats(ownname         => 'NEIL',
                                   obj_filter_list => l_filter_array,
                                   options         => 'gather auto');

END;
```

UNDOCUMENTED OPTIONS

Where do you even start?

"Bernard of Chartres (CE 1124-*ish*) used to say that we are like dwarves perched on the shoulders of giants, and thus we are able to see more and farther than the latter. "

trace the execution with strace/dtrace/gdb (erm… I'd rather not!)

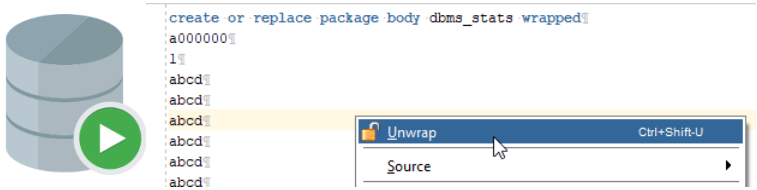
look at the source code!

DBA_SOURCE WHERE NAME='DBMS_STATS'

```
 procedure set_global_prefs(
    pname   varchar2,
    pvalue  varchar2);
--
-- This procedure is used to set the global statistics preferences.
-- This setting is honored only of there is no preference specified
-- for the table to be analyzed.
--
-- To run this procedure, you need to have the SYSDBA OR
-- both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.
--
-- Input arguments:
--   pname    - preference name
--              The default value for following preferences can be set.
--                 CASCADE
--                 DEGREE
--                 ESTIMATE_PERCENT
--                 METHOD_OPT
--                 NO_INVALIDATE
--                 GRANULARITY
--                 PUBLISH
--                 INCREMENTAL
--                 INCREMENTAL_LEVEL
--                 INCREMENTAL_STALENESS
--                 GLOBAL_TEMP_TABLE_STATS
--                 STALE_PERCENT
--                 AUTOSTATS_TARGET
--                 CONCURRENT
--                 TABLE_CACHED_BLOCKS
--                 OPTIONS
--                 STAT_CATEGORY
--                 PREFERENCE_OVERRIDES_PARAMETER
--                 APPROXIMATE_NDV_ALGORITHM
--                 AUTO_STAT_EXTENSIONS
--                 WAIT_TIME_TO_UPDATE_STATS
--                 ROOT_TRIGGER_PDB
--                 COORDINATOR_TRIGGER_SHARD
.
--  PUBLISH: The "PUBLISH" value determines whether or not newly gathered
--    statistics will be published once the gather job has completed.
--    Prior to 11g, once a statistic gathering job completed, the new
--    statistics were automatically published into the dictionary tables.
--    The user now has the ability to gather statistics but not publish
--    them immediately. This allows the DBA to test the new statistics
--    before publishing them.
```

UNDOCUMENTED OPTIONS

The source is "wrapped"




```
CASE PNAMEU
   WHEN 'PUBLISH' THEN
    BOOLTYPE := TO_PUBLISH_TYPE(PVALU);
   WHEN 'INCREMENTAL' THEN
    BOOLTYPE := TO_INCREMENTAL_TYPE(PVALU);
   WHEN 'INCREMENTAL_INTERNAL_CONTROL' THEN
    VALIDATE_INCREMENTAL_INTERNAL(PVALU);
    DBMS_STATS_INTERNAL.INCREMENTAL_INTERNAL_CONTROL := PVALU;
   WHEN 'INCREMENTAL_LEVEL' THEN
    BOOLTYPE := VALIDATE_INCREMENTAL_LEVEL(
              TRUE,
              PVALU);
   WHEN 'INCREMENTAL_STALENESS' THEN
    BOOLTYPE := VALIDATE_INCREMENTAL_STALENESS(PVALU);
```

PUBLIC - 24 parameters

```
APPROXIMATE_NDV_ALGORITHM

AUTO_STAT_EXTENSIONS
AUTOSTATS_TARGET




CASCADE
CONCURRENT
COORDINATOR_TRIGGER_SHARD

DEGREE



ESTIMATE_PERCENT


GLOBAL_TEMP_TABLE_STATS
GRANULARITY

INCREMENTAL
INCREMENTAL_LEVEL
INCREMENTAL_STALENESS




METHOD_OPT
NO_INVALIDATE
OPTIONS
PREFERENCE_OVERRIDES_PARAMETER
PUBLISH
ROOT_TRIGGER_PDB

STALE_PERCENT
STAT_CATEGORY

TABLE_CACHED_BLOCKS

WAIT_TIME_TO_UPDATE_STATS
```

SOURCE - 17 undocumented parameters

```
ANDV_ALGO_INTERNAL_OBSERVE
APPROXIMATE_NDV_ALGORITHM
APPROXIMATE_NDV
AUTO_STAT_EXTENSIONS
AUTOSTATS_TARGET
AUTO_TASK_INTERVAL
AUTO_TASK_MAX_RUN_TIME
AUTO_TASK_STATUS
CASCADE
CONCURRENT
COORDINATOR_TRIGGER_SHARD
DEBUG
DEGREE
ENABLE_HYBRID_HISTOGRAMS
ENABLE_TOP_FREQ_HISTOGRAMS
ESTIMATE_PERCENT
GATHER_AUTO
GATHER_SCAN_RATE
GLOBAL_TEMP_TABLE_STATS
GRANULARITY
INCREMENTAL_INTERNAL_CONTROL
INCREMENTAL
INCREMENTAL_LEVEL
INCREMENTAL_STALENESS
JOB_OVERHEAD_PERC
JOB_OVERHEAD
MAINTAIN_STATISTICS_STATUS
METHOD_OPT
NO_INVALIDATE
OPTIONS
PREFERENCE_OVERRIDES_PARAMETER
PUBLISH
ROOT_TRIGGER_PDB
SCAN_RATE
STALE_PERCENT
STAT_CATEGORY
SYS_FLAGS
TABLE_CACHED_BLOCKS
TRACE
WAIT_TIME_TO_UPDATE_STATS
```


I solemnly swear that I am up to no good

TRACING DBMS_STATS

Undocumented Trace Option [grep -E "DSC_.*_TRC *CONSTANT" dbms_stats_internal.unwrapped]

```
dbms_stats.set_global_prefs('trace',value)
```

```
     1 = use dbms_output.put_line instead of writing into trace file
     2 = enable dbms_stat trace only at session level
     4 = trace table stats
     8 = trace index stats
    16 = trace column stats
    32 = trace auto stats - logs to sys.stats_target$_log
    64 = trace scaling
   128 = dump backtrace on error
   256 = dubious stats detection
   512 = auto stats job
  1024 = parallel execution tracing
  2048 = print query before execution
  4096 = partition prune tracing
  8192 = trace stat differences
 16384 = trace extended column stats gathering
 32768 = trace approximate NDV (number distinct values) gathering
 65536 = Online Trace
131072 = Adaptive Operations?
262144 = System Stats
524288 = Advisor
```

WARNING! Setting instance-wide parameters can have unexpected side effects!

```
exec dbms_stats.set_global_prefs('trace',65532);
                =32768 + 16384 + 8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4
```

TRACING DBMS_STATS

Undocumented Trace Option - proving 1 row change leads to staleness

```
select * from user_tab_modifications order by 1,2,3;

TABLE_NAME   PARTITION_NAME  INSERTS    UPDATES DELETES
------------ --------------- ---------- ------- -------
INTERVAL_TAB SYS_P82847            0          1       0
INTERVAL_TAB SYS_P82848            0        358       0
INTERVAL_TAB                       0        359       0
```

Are the statistics marked as stale?

```
select table_name, partition_name, subpartition_name,stale_stats from dba_tab_statistics
where  table_name = 'INTERVAL_TAB' order by partition_position, subpartition_position;

TABLE_NAME   PARTITION_NAME  SUBPARTITION_NAME              STALE_S
------------ --------------- ------------------------------ -------
INTERVAL_TAB COL_PART01                                     NO
INTERVAL_TAB SYS_P82847                                     NO
INTERVAL_TAB SYS_P82848                                     YES
```

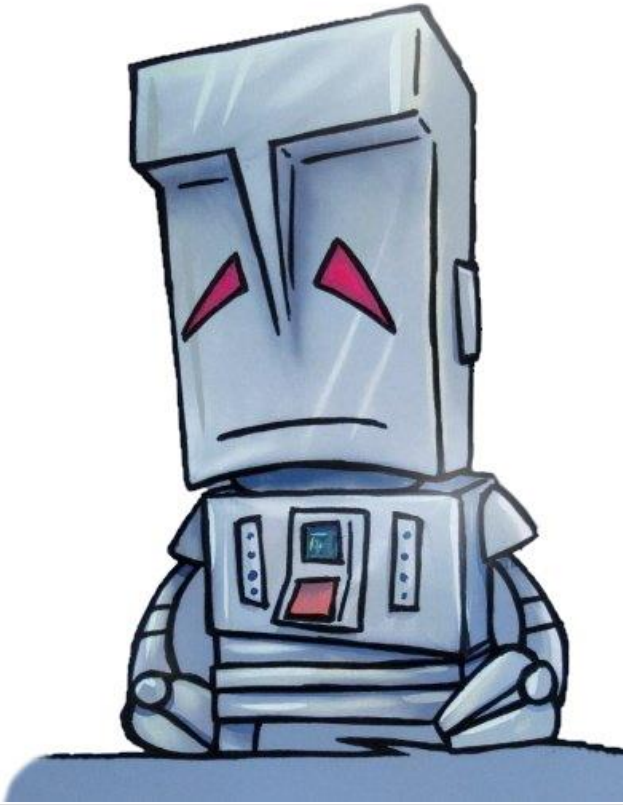
TRACING DBMS_STATS

Undocumented Trace Option - proving 1 row change leads to staleness

```
select * from user_tab_modifications order by 1,2,3;

TABLE_NAME    PARTITION_NAME  INSERTS    UPDATES DELETES
------------ --------------- ---------- ------- -------
INTERVAL_TAB SYS_P82847             0         1       0
INTERVAL_TAB SYS_P82848             0       358       0
INTERVAL_TAB                        0       359       0


From the trace file:
DBMS_STATS: gather stats on partition SYS_P82847: stats are stale;
DBMS_STATS: Start gather_stats.. pfix:  ownname: NEIL tabname: INTERVAL_TAB pname: SYS_P82847
spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL in incremental (recent stats use HLL)
DBMS_STATS: reporting_man_log_task: target: "NEIL"."INTERVAL_TAB"."SYS_P82847" objn: 175085
auto_stats: FALSE status: IN PROGRESS ctx.batching_coeff: 0
DBMS_STATS: delete synopses of a single partition
DBMS_STATS: Starting query at 29-SEP-19 04.21.35.959229000 PM +01:00

dbms_stats.set_table_prefs('NEIL','INTERVAL_TAB','INCREMENTAL_STALENESS','USE_STALE_PERCENT');
```

WHAT DIDN'T I COVER?

- Doing Stats Manually (and Faking Histograms)
- Pending Statistics to test plans before you unleash them
- Stats over Database Links
- +lots more - stats is a big subject!



I've calculated your chance of survival, but I don't think you'll like it.

Any Questions?

BLOG: http://chandlerdba.com
Twitter: **@chandlerDBA**
E: neil@chandler.uk.com

APPROXIMATE_NDV

Let $h : \mathcal{D} \to [0,1] \equiv \{0,1\}^\infty$ *hash data from domain* $\mathcal{D}$ *to the binary domain.*
Let $\rho(s)$*, for* $s \in \{0,1\}^\infty$*, be the position of the leftmost 1-bit* ($\rho(0001\cdots) = 4$).

Algorithm HYPERLOGLOG (**input** $\mathcal{M}$: multiset of items from domain $\mathcal{D}$).
assume $m = 2^b$ with $b \in \mathbb{Z}_{>0}$;
initialize a collection of m registers, $M[1], \ldots, M[m]$, to $-\infty$;

for $v \in \mathcal{M}$ **do**
  set $x := h(v)$;
  set $j = 1 + \langle x_1 x_2 \cdots x_b \rangle_2$; {*the binary address determined by the first b bits of* x}
  set $w := x_{b+1} x_{b+2} \cdots$; **set** $M[j] := \max(M[j], \rho(w))$;

compute $Z := \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$; {*the "indicator" function*}

return $E := \alpha_m m^2 Z$ with α_m as given by Equation (3).

$$E := \frac{\alpha_m m^2}{\sum_{j=1}^{m} 2^{-M(j)}}, \qquad \text{with} \quad \alpha_m := \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}.$$

DBMS_STATS.APPROXIMATE_NDV_ALGORITHM
"ADAPTIVE SAMPLING" / "HYPERLOGLOG" / "REPEAT OR HYPERLOGLOG"
(only the old way / only new way - recalculate it all / keep the old but recalc new stats)

APPROXIMATE_NDV

SCAN ALL VALUES (e.g. a table scan), plus 10 "random" actual values

Oracle hashes every new value and stores the hash in a "column synopsis"

When the hash table (16384 values) is full
it throws ½ of the lower half values away "domain splitting"
& keeps going

If it fills again, it throws half of the values away again
where the 1st bit is 0 and keeps going

Repeat but store only where the 1st bit is 1

Then it multiples the NDV in the has table by the #splits