HISTOGRAMS ARE EVIL LIKE CHOCOLATE IS EVIL

Neil Chandler,
Chandler Systems

22. hroug
godišnja konferencija
17.-20.10.2017.

HISTOGRAMS ARE EVIL
LIKE
CHOCOLATE IS EVIL

Neil Chandler,
Chandler Systems

22. hroug
godišnja konferencija
17.-20.10.2017.

HISTOGRAMS ARE EVIL LIKE CHOCOLATE IS EVIL

Neil Chandler, Chandler Systems

Working in IT since 1988
Working with Oracle since about 1991

Chairman of the UKOUG RAC, Cloud, Infrastructure and Availability SIG
Organiser of UKOUG Tech17

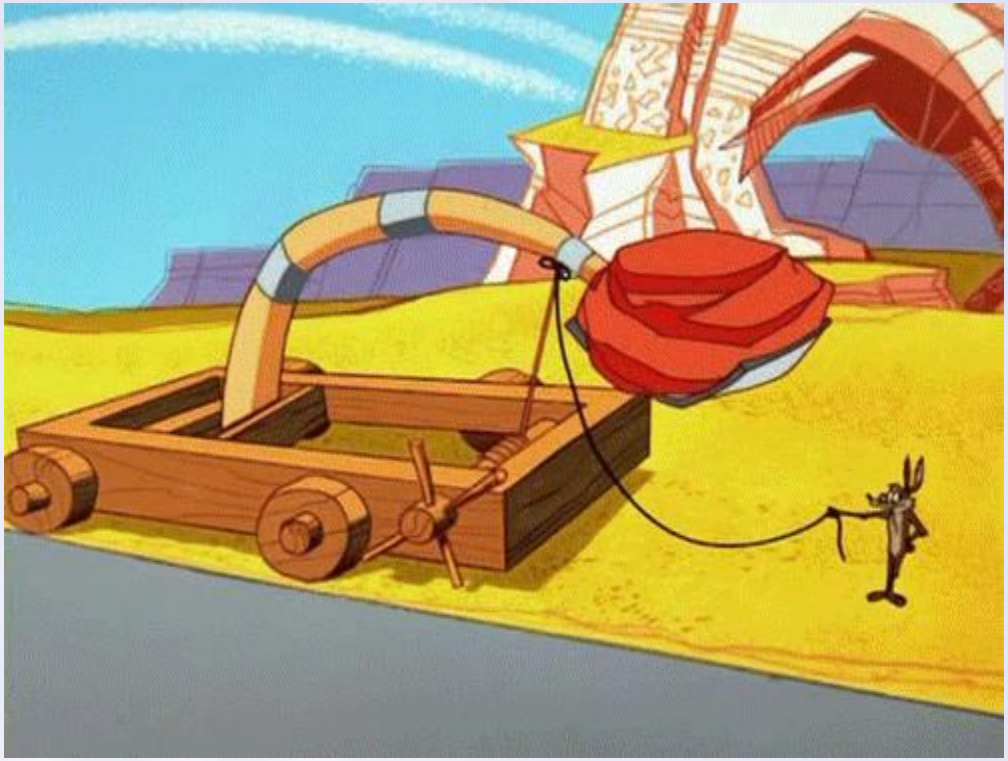
BLOG: http://chandlerdba.wordpress.com/
Tweets: @chandlerdba

INTRODUCTION

What is your least favourite thing about Oracle

INTRODUCTION

DON'T INSULT THE OPTIMIZER
WHEN YOU ARE SITTING NEXT TO
THE OPTIMIZER LADY



INTRODUCTION

DON'T INSULT THE OPTIMIZER
WHEN YOU ARE SITTING NEXT TO
THE OPTIMIZER LADY

AGENDA

- What are Histograms
- Chocolate is Evil
- Some Histogram Details

  Frequency / Top Frequency / Height Balanced / Hybrid
- A couple of Myths

OPTIMIZER

Poor Stats =>

Incorrect **Cardinality** Estimates =>

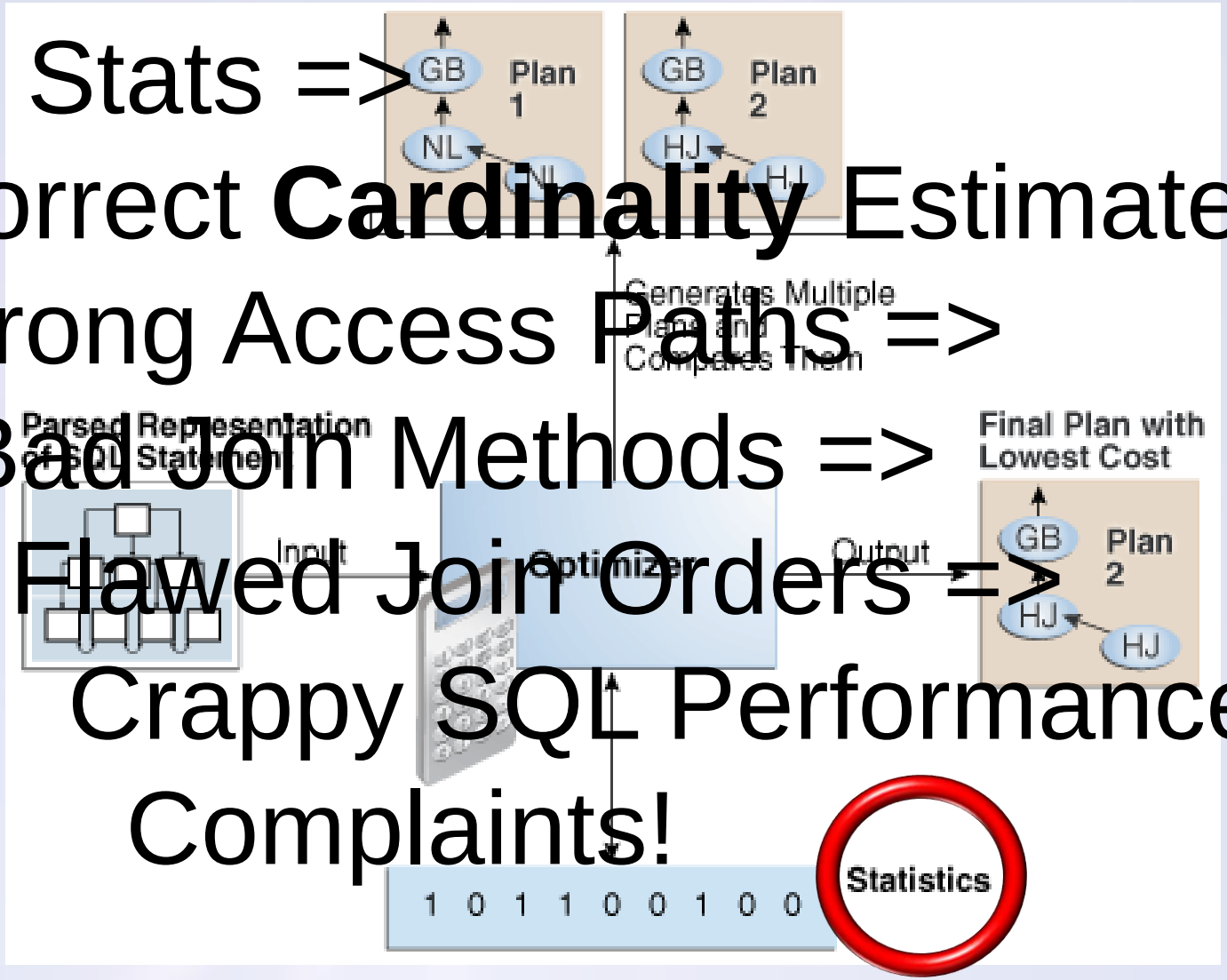
Wrong Access Paths =>

Bad Join Methods =>

Flawed Join Orders =>

Crappy SQL Performance =>

Complaints!



WHAT IS CARDINALITY?

The percentage of the table Oracle thinks you are going to return for a given predicate

Number of Rows in the table x DENSITY

The DENSITY where you do not have a Histogram =
1 / Number-of-Distinct-Values-in-the-column

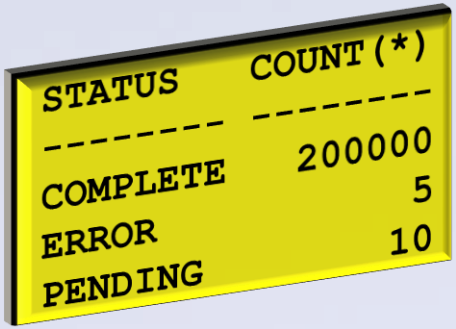
DBA_TAB_COL_STATISTICS.DENSITY
(or DBA_TAB_COLUMNS.DENSITY)

<u>Ignore this value if there is a Histogram on the column!</u>

HISTOGRAMS

- A type of *better* statistics
- Explains the data distributions in a column to improve **cardinality** estimates for a value
- But they may take a lot of effort to gather the extra stats
- May lead to a less efficient execution plan
- Can promote plan instability

HISTOGRAMS



STATUS	COUNT(*)
COMPLETE	200000
ERROR	5
PENDING	10

method_opt=>'FOR ALL COLUMNS SIZE 1'

NO HISTOGRAM
Cardinality =
num_rows * (1 / NDV) =
num_rows * density =
200015 * (1 / 3) = 66671.6

DBA_TAB_COLUMNS – No Histogram

TABLE_NAME	COLUMN_NAM	DATA_TYPE	NUM_DISTINCT	DENSITY	HISTOGRAM	NUM_BUCKETS	LOW_DECODE	HIGH_DECODE
TEST_BASIC	STATUS	VARCHAR2	3	0.333333333333	NONE	1	COMPLETE	PENDING

DBA_TAB_HISTOGRAMS

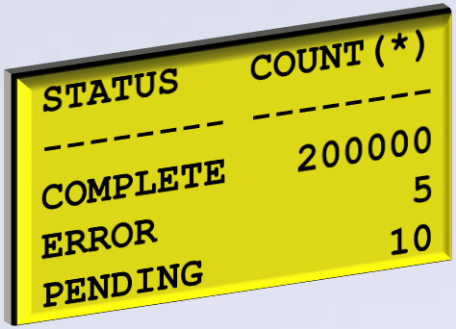
TABLE_NAME	COLUMN_NAM	ENDPOINT_NUMBER	ENDPOINT_VALUE	ENDPOINT_DECODE (6 chars)
TEST_BASIC	STATUS	0	34949232530004200000000000000000000000	COMPLE
TEST_BASIC	STATUS	1	41678943587571600000000000000000000000	PENDIN

DBMS_XPLAN – Execution Plan

```
select ID from test_basic where status = 'ERROR';
```

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	Buffers
0	SELECT STATEMENT		1			171 (100)	5	570
*1	TABLE ACCESS FULL	TEST_BASIC	1	66672	911K	171 (1)	5	570

HISTOGRAMS




```
method_opt=>'FOR ALL COLUMNS SIZE AUTO'
```

DBA_TAB_COLUMNS – Frequency Histogram

```
TABLE_NAME      COLUMN_NAM DATA_TYPE  NUM_DISTINCT        DENSITY HISTOGRAM  NUM_BUCKETS LOW_DECODE HIGH_DECODE
--------------- ---------- ---------- ------------ --------------- ---------- ----------- ---------- -----------
TEST_BASIC      STATUS     VARCHAR2              3  0.000002499813 FREQUENCY            3 COMPLETE   PENDING
```

200000-0 = 200000 rows with value of "COMPLETE"
200005-200000 = 5 rows with value of "ERROR"
200015-200005 = 10 rows with value of "PENDING"

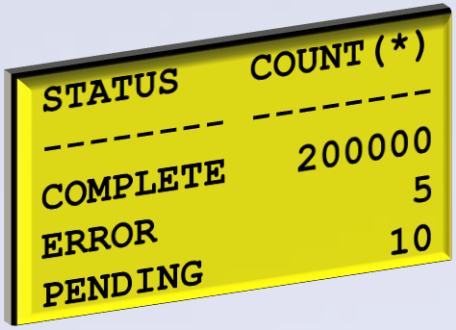
DBA_TAB_HISTOGRAMS

```
TABLE_NAME      COLUMN_NAM ENDPOINT_NUMBER                              ENDPOINT_VALUE ENDPOINT_DECODE (6 chars)
--------------- ---------- --------------- ---------------------------------------------- ------------------------
TEST_BASIC      STATUS                200000      34949232530004200000000000000000000000 COMPLE
TEST_BASIC      STATUS                200005      35993816208417600000000000000000000000 ERROQ
TEST_BASIC      STATUS                200015      41678943587571600000000000000000000000 PENDIN
```

DBMS_XPLAN – Execution Plan

```
select ID from test_basic where status = 'ERROR';
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Starts | E-Rows |E-Bytes|Cost (%CPU)|A-Rows |Buffers|
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |      1 |        |       |    4 (100)|     5 |      6 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TEST_BASIC |      1 |      5 |    70 |    4   (0)|     5 |      6 |
|*  2 |   INDEX RANGE SCAN                  | IDX_STATUS |      1 |      5 |       |    3   (0)|     5 |      4 |
-------------------------------------------------------------------------------------------------------------------
```

HISTOGRAMS : TIMING

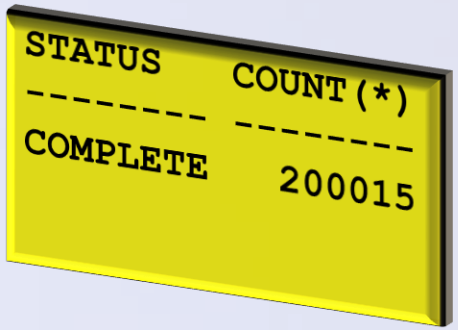


STATUS	COUNT(*)
COMPLETE	200000
ERROR	5
PENDING	10

The TIME OF DAY you gather stats is critical!

- 18:00 Processing Ends for the day
- 19:00 All PENDING records are processed and COMPLETE
- 20:00 All ERROR records are processed and COMPLETE
- 22:00 Gather Stats...

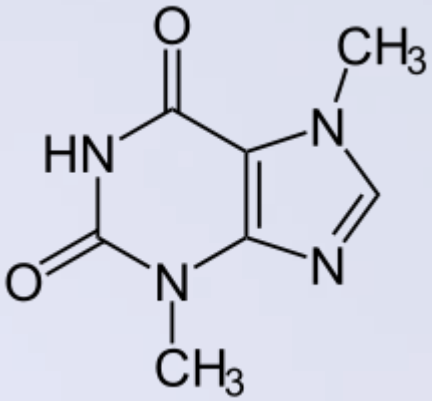
You only have status of COMPLETE



STATUS	COUNT(*)
COMPLETE	200015

CHOCOLATE IS EVIL?

CHOCOLATE CAN BE EVIL!



Chocolate contains Theobromine, an "alkaloid" (like Cocaine and Caffeine)
A lethal dose of chocolate for a human is about 22lb / 10kilos

HISTOGRAMS ARE EVIL?

Oracle
LOVES
histograms

5-25% of columns will get Frequency histograms, but I have seen as high as 60%

Seems to be higher in 12. Height Balanced are replaced Hybrid but you may also get a Frequency or Top-Frequency instead.

Maybe 2-3% of columns get Hybrid/Height Balanced Histograms – but usually on the BIG tables…

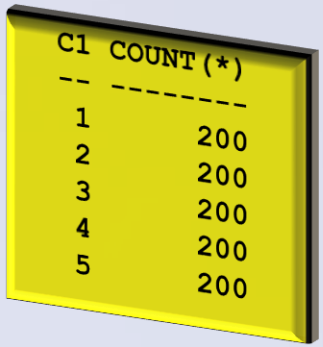
HISTOGRAM MYTH

HISTOGRAM MYTHS

You only get a histogram on skewed data

HISTOGRAM MYTHS : SKEW

```
create table test_uniform (c1 number not null);
Table created.

insert into test_uniform (c1) select mod(rownum,5)+1
from dba_objects where rownum < 1001;
1000 rows created.
```



Need a query before size "auto" kicks in to create a histogram

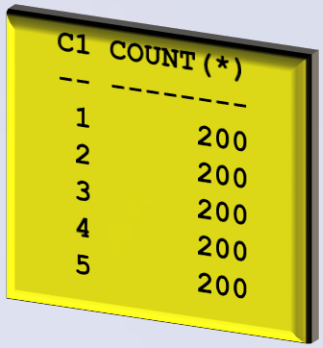
[We could use "skewonly" instead of auto to do this]

```
select count(*) from test_uniform where c1 > 4;
  COUNT(*)
----------
       200
```

```
SQL> select * from sys.col_usage$ where

TABLE_NAME   COL_NM EQUALITY EQUIJOIN NONEQUIJOIN RANGE LIKE NULL
------------ ------ -------- -------- ----------- ----- ---- ----
TEST_UNIFORM C1            0        0           0     1    0    0
```

(or you could use **DBMS_STATS.REPORT_COL_USAGE**)

HISTOGRAM MYTHS : SKEW



C1	COUNT(*)
1	200
2	200
3	200
4	200
5	200

```
dbms_stats.gather_table_stats(null,'TEST_UNIFORM',
          method_opt=>'FOR ALL COLUMNS SIZE AUTO');
```

TABLE_NAME	COL_NM	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	HISTOGRAM
TEST_UNIFORM	C1	5	0	5	**FREQUENCY**

USER_TAB_HISTOGRAMS

TABLE_NAME	COLUMN_NAM	ENDPOINT_NUMBER	ENDPOINT_VALUE
TEST_UNIFORM	C1	200	1
TEST_UNIFORM	C1	400	2
TEST_UNIFORM	C1	600	3
TEST_UNIFORM	C1	800	4
TEST_UNIFORM	C1	1000	5

HISTOGRAMS ARE EVIL?

Oracle
LOVES
histograms

5-25% of columns will get Frequency histograms, but I have seen as high as 60%

Seems to be higher in 12. Height Balanced are replaced Hybrid but you may also get a Frequency or Top-Frequency instead.

Maybe 2-3% of columns get Hybrid/Height Balanced Histograms – but usually on the BIG tables…

**Where you have a predicate,
you're probably getting a histogram…**

(except for equality predicates against unique not null columns)

HISTOGRAMS ARE EVIL?

Oracle
LOVES
histograms

**Where you have a predicate,
you're probably getting a histogram…**

- Histograms Consume Resources
  to create and maintain
  (esp. Hybrid and Height Balanced)
- Histograms Consume Resources to Use - they are resident in the dictionary cache
- Histograms Consume Resources to Store -
  SYSAUX can get big, especially with lots of partitions
- Histograms *may* make your plans worse
- Histograms *may* make your plans less stable

HISTOGRAMS TYPES

Pre-12C

Frequency

Where there are less distinct values than entries/buckets (up to 254 entries/buckets)

Height Balanced

Where there are more distinct values than entries/buckets

New for 12C

Top-Frequency

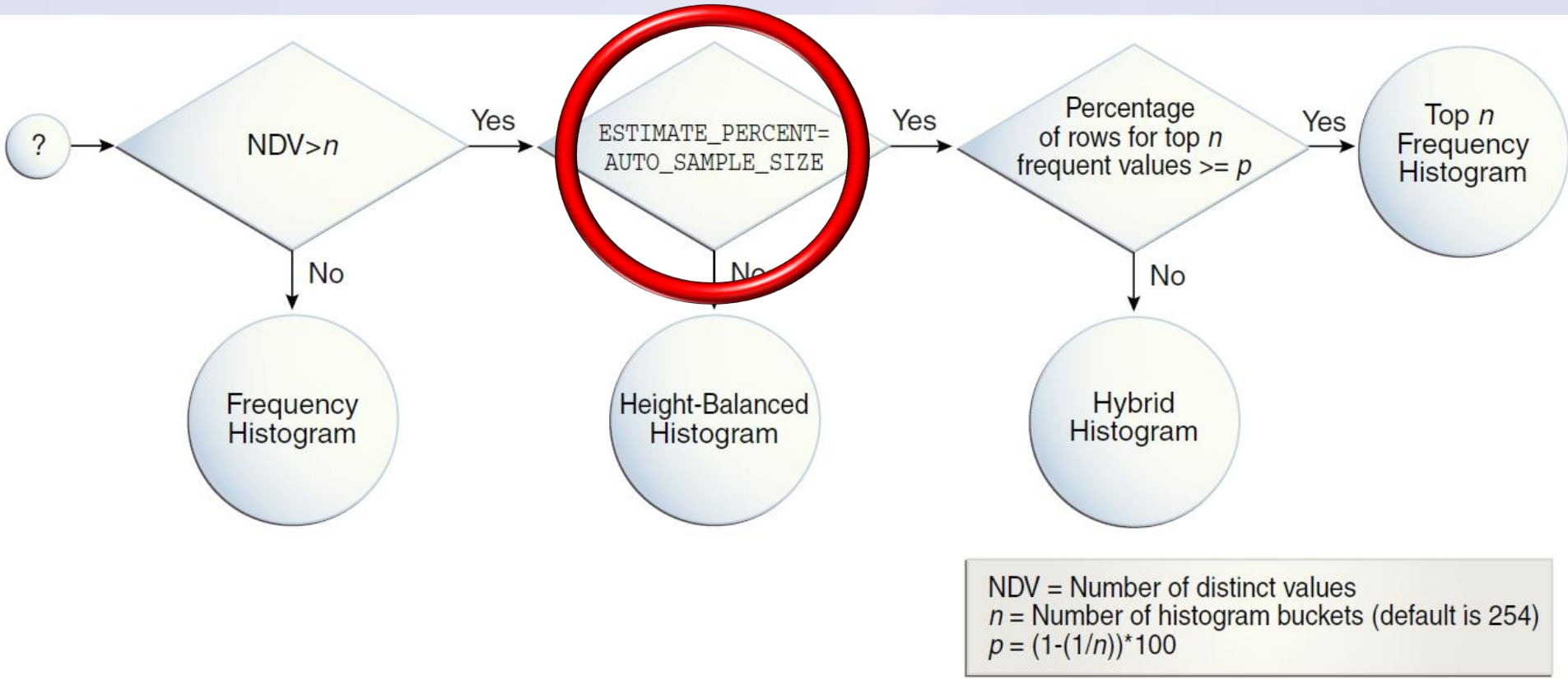
Where there are slightly more distinct values than entries

Hybrid

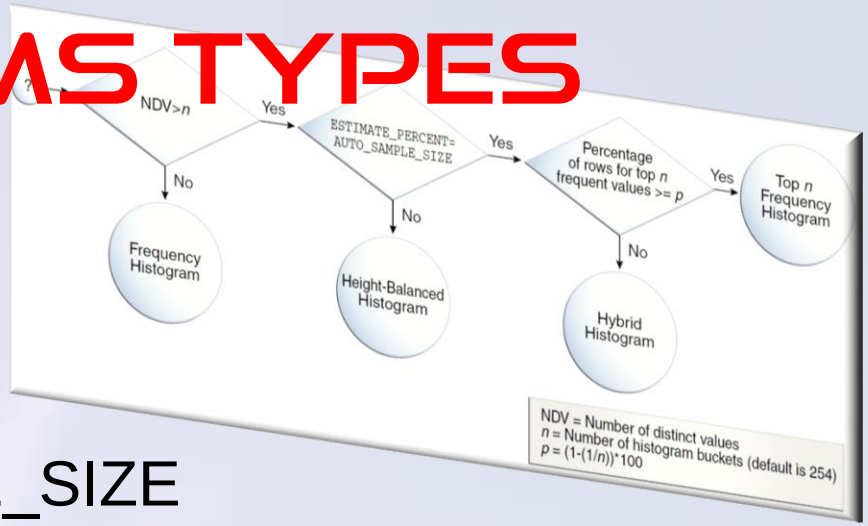
Replaces Height Balanced

Allowed to have 2048 entries/buckets (default still 254)

HISTOGRAMS TYPES



HISTOGRAMS TYPES



ESTIMATE_PERCENT

You only get the 2 new types of Histogram if set to AUTO_SAMPLE_SIZE

From 11G, if left to default to AUTO_SAMPLE_SIZE, the stats job performs FULL TABLE SCAN instead of Adaptive Sampling

Histogram Stats Processing has changed in 12:

10G - samples per column, re-sampled larger based upon perceived NDV correctness and number of NULLS

11G - typically one* sample for all histograms

12C - Frequency-type Histograms gathered in a single pass using new APPROXIMATE_NDV processing

*if column(s) have lots of NULLs, we may get a multiple samples

APPROXIMATE_NDV

12C is using ***awesome*** maths using "HyperLogLog" algorithm for near perfect cardinality calculations in a single table scan

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*}

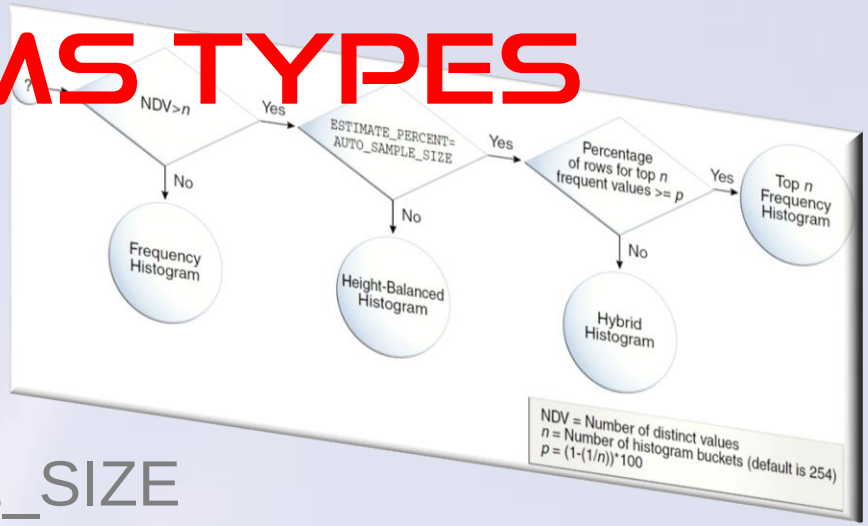
return $E := \alpha_m m^2 Z$ with α_m as given by Equation (3).

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

DBMS_STATS.APPROXIMATE_NDV_ALGORITHM
"ADAPTIVE SAMPLING" / "HYPERLOGLOG" / "REPEAT OR HYPERLOGLOG"

HISTOGRAMS TYPES

ESTIMATE_PERCENT



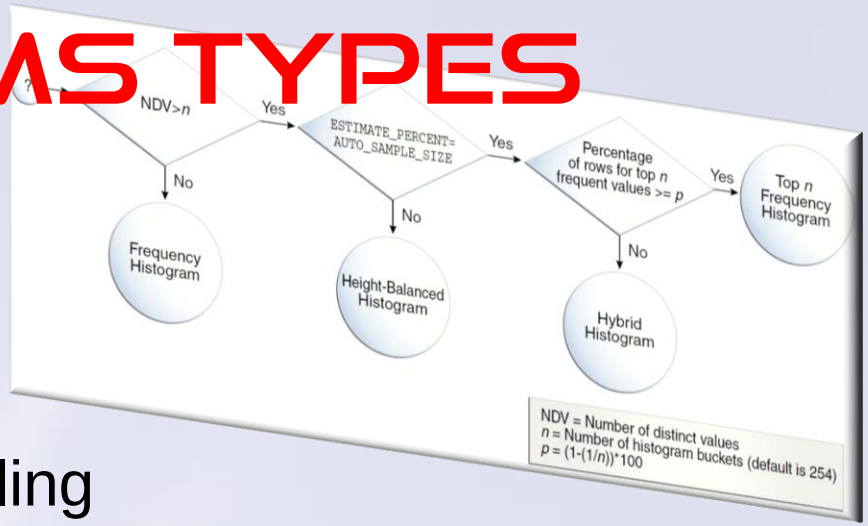
You only get the 2 new types of Histogram if set to AUTO_SAMPLE_SIZE

From 11G, if left to default to AUTO_SAMPLE_SIZE, the stats job performs FULL TABLE SCAN instead of Adaptive Sampling

This means that gathering Frequency and Top-Frequency Histograms is free* and very accurate

(*tiny bit of CPU)

HISTOGRAMS TYPES

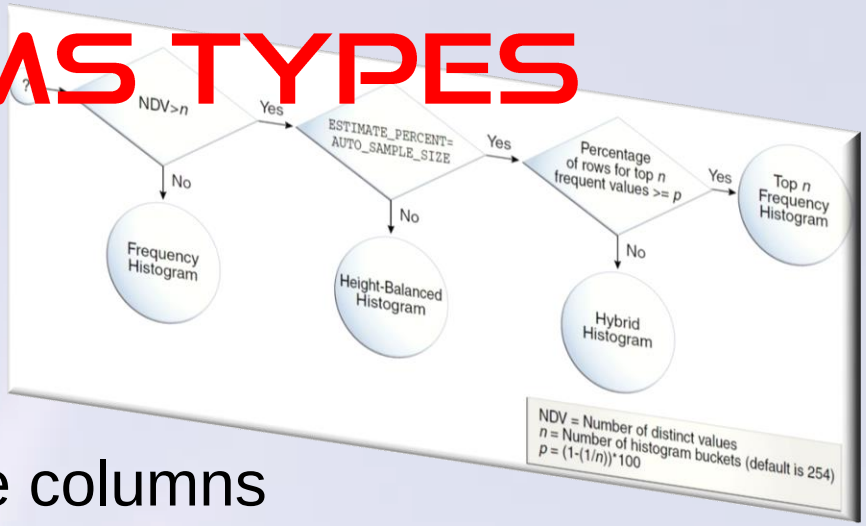


Adaptive Sampling

Height Balanced and **Hybrid** histograms still use Adaptive Sampling

```
method_opt=>'FOR ALL COLUMNS SIZE AUTO FOR COLUMNS SIZE 12 COL_HYBRID'

COLUMN_NAME      NUM_DISTINCT NULLS NUM_BUCKETS HISTOGRAM SAMPLE_SIZE
---------------- ------------ ----- ----------- --------- -----------
TST_ID                  53335     0           1 NONE            53335
COL_HYBRID                 29     0          12 HYBRID           5590
COL_FREQUENCY              10     1          10 FREQUENCY       53334
COL_TEXT                46520     1           1 NONE            53334
```

HISTOGRAMS TYPES



Adaptive Sampling

Adaptive Sample size is increased when there are NULLs in one of the columns

```
method_opt=>'FOR ALL COLUMNS SIZE AUTO
             FOR     COLUMNS SIZE 12 COL_HYBRID,COL_HYBRID_NULLS'
```

COLUMN_NAME	NUM_DISTINCT	NULLS	NUM_BUCKETS	HISTOGRAM	SAMPLE_SIZE
TST_ID	53335	0	1	NONE	53335
COL_HYBRID	**29**	**0**	**12**	**HYBRID**	**9276**
COL_HYBRID_NULLS	**19**	**21671**	**12**	**HYBRID**	**5503**
COL_FREQUENCY	10	1	10	FREQUENCY	53334
COL_TEXT	46912	1	1	NONE	53334

Sample has been increased to 9,276 rows so the column with NULLs has a large enough sample

HISTOGRAMS TYPES

SQL for stats gathering is FTS

```
SELECT /*+ lots-of-hints */
  TO_CHAR(COUNT("TST_ID")),
  substrb(dump(MIN("TST_ID"),16,0,64),1,240),
  substrb(dump(MAX("TST_ID"),16,0,64),1,240),
  TO_CHAR(COUNT("COL_HYBRID")),
  substrb(dump(MIN("COL_HYBRID"),16,0,64),1,240),
  substrb(dump(MAX("COL_HYBRID"),16,0,64),1,240),
  TO_CHAR(COUNT("COL_HYBRID_NULLS")),
  substrb(dump(MIN("COL_HYBRID_NULLS"),16, 0,64),1,240),
  substrb(dump(MAX("COL_HYBRID_NULLS"),16,0,64),1,240),
  TO_CHAR(COUNT("COL_FREQUENCY")),
  substrb(dump(MIN("COL_FREQUENCY"),16,0,64), 1,240),
  substrb(dump(MAX("COL_FREQUENCY"),16,0,64),1,240),
  TO_CHAR(COUNT("COL_TEXT")),
  substrb(dump(MIN("COL_TEXT"),16,0,64),1,240),
  substrb(dump(MAX("COL_TEXT"),16,0,64),1,240),
  COUNT(rowidtochar(rowid))
FROM "NEIL"."TEST_SAMPLING" t
  /* ACL,NIL,NIL,TOPN,NIL,NIL,TOPN,NIL,NIL,TOPN,NIL,
  NIL,NDV,NIL,NIL,RWID,U254,U12,U12,U254,U254U */

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=248 pr=0 pw=0 time=107302 us starts=1)
     53335      53335      53335   OPTIMIZER STATISTICS GATHERING  (cr=248 pr=0 pw=0 time=141938 us starts=1 cost=68 size=1013365 card=53335)
     53335      53335      53335    TABLE ACCESS FULL TEST_SAMPLING (cr=248 pr=0 pw=0 time=22027 us starts=1 cost=68 size=1013365 card=53335)
```

Some additional information for the Approximate NDV calculations

```
SELECT /*+ lots-of-hints */
  substrb(dump("COL_FREQUENCY",16,0,64),1,240) val,
  rowidtochar(rowid) rwid
FROM "NEIL"."TEST_SAMPLING" t
WHERE rowid IN (chartorowid('AAASLXAAOAAAACDAAA'),chartorowid('AAASLXAAOAAAACDAAB'),chartorowid('AAASLXAAOAAAACDAAC'),
chartorowid('AAASLXAAOAAAACDAAD'), chartorowid('AAASLXAAOAAAACDAAE'),chartorowid('AAASLXAAOAAAACDAAF'),chartorowid('AAASLXAAOAAAACDAAG'),
chartorowid('AAASLXAAOAAAACDAAI'),chartorowid('AAASLXAAOAAAACDAAJ'),chartorowid('AAASLXAAOAAAACDAAR')) ORDER BY "COL_FREQUENCY"

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
        10         10         10  SORT ORDER BY (cr=1 pr=0 pw=0 time=43 us starts=1 cost=2 size=19 card=1)
        10         10         10   INLIST ITERATOR  (cr=1 pr=0 pw=0 time=20 us starts=1)
        10         10         10    TABLE ACCESS BY USER ROWID TEST_SAMPLING (cr=1 pr=0 pw=0 time=10 us starts=10 cost=1 size=19 card=1)
```

HISTOGRAMS TYPES

Create GTT to hold sample data

```
CREATE global TEMPORARY TABLE sys.ora_temp_1_ds_560022 sharing=none
ON COMMIT preserve rows cache noparallel
AS
  SELECT
    /*+ lots-of-hints */
    "COL_HYBRID",
    "COL_HYBRID_NULLS",
    rowid SYS_DS_ALIAS_0
  FROM "NEIL"."TEST_SAMPLING" sample ( 17.3698837797) t
  WHERE 1 = 2

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         0          0          0  LOAD AS SELECT  ORA_TEMP_1_DS_560022 (cr=0 pr=0 pw=0 time=30 us starts=1)
         0          0          0   FILTER  (cr=0 pr=0 pw=0 time=2 us starts=1)
         0          0          0    TABLE ACCESS SAMPLE TEST_SAMPLING (cr=0 pr=0 pw=0 time=0 us starts=0 cost=68 size=176016 card=9264)
```

Grab the sample data - it knows there's NULLs from the earlier FTS

```
INSERT /*+ append */
INTO sys.ora_temp_1_ds_560022
SELECT
  /*+ lots-of-hints */
  "COL_HYBRID",
  "COL_HYBRID_NULLS",
  rowid SYS_DS_ALIAS_0
FROM "NEIL"."TEST_SAMPLING" sample ( 17.3698837797) t
UNION ALL
SELECT "COL_HYBRID",
  "COL_HYBRID_NULLS",
  SYS_DS_ALIAS_0
FROM sys.ora_temp_1_ds_560022
WHERE 1 = 0

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         0          0          0  LOAD AS SELECT  ORA_TEMP_1_DS_560022 (cr=248 pr=0 pw=28 time=51969 us starts=1)
      9276       9276       9276   UNION-ALL  (cr=248 pr=0 pw=0 time=6110 us starts=1)
      9276       9276       9276    TABLE ACCESS SAMPLE TEST_SAMPLING (cr=248 pr=0 pw=0 time=3440 us starts=1 cost=68 size=176016 card=9264)
         0          0          0    FILTER  (cr=0 pr=0 pw=0 time=1 us starts=1)
         0          0          0     TABLE ACCESS FULL ORA_TEMP_1_DS_560022 (cr=0 pr=0 pw=0 time=0 us starts=0 cost=29 size=310384 card=8168)
```

HISTOGRAMS TYPES

and some final bits of information

```
SELECT
  /*+  hints */
  COUNT(*)                                          AS nrw,
  COUNT(DISTINCT sys_op_lbid(74456,'L',t.rowid)) AS nlb,
  NULL                                              AS ndk,
  sys_op_countchg(substrb(t.rowid,1,15),1)       AS clf
FROM "NEIL"."TEST_SAMPLING" t
WHERE "TST_ID" IS NOT NULL
```

```
Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT GROUP BY (cr=107 pr=0 pw=0 time=35039 us starts=1)
     53335      53335      53335   INDEX FULL SCAN TEST_SAMPLING_PK
                                     (cr=107 pr=0 pw=0 time=7045 us starts=1 cost=107 size=640020 card=53335)(object id 74456)
```

SAMPLING THREAT

if you are SAMPLING data
(Hybrid / Height Balanced)
rare values will appear
and disappear, potentially
causing plan stability issues

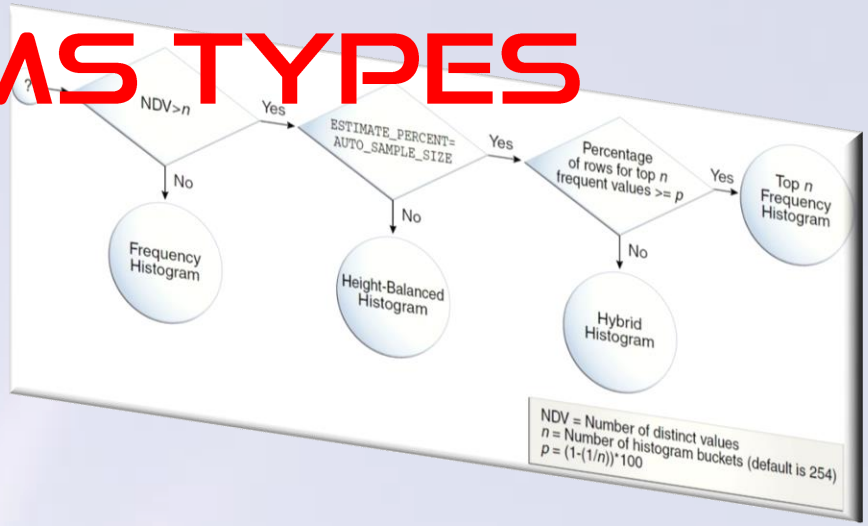
HISTOGRAM MYTHS

Histograms are only for indexed columns?

They help with Join Cardinality and therefore Join Methods too

If the optimizer *can* use stats to help, it *will* be using stats to help

HISTOGRAMS TYPES



Frequency

We have more "buckets" than NDV's

Top-Frequency

A few more NDV's than entries (buckets) available but we are allowed to throw away "insignificant" values which rarely occur.

Like ALL histograms, it must record Low/High Values.

n=254

p=(1-(1/254))*100=

> 99.6% of the data must fit into 254 entries. 0.4% is "insignificant"

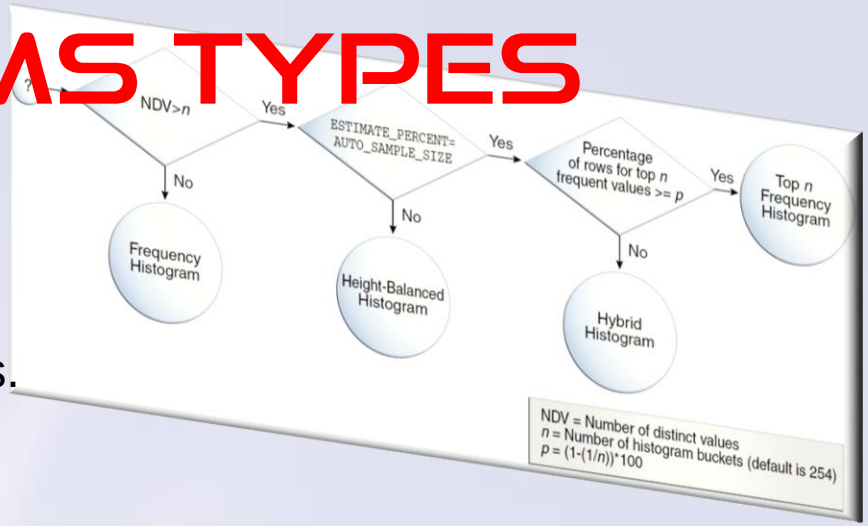
n=20

p=(1-(1/20))*100

> 95% of the data must fit into 20 entries. 5% is "insignificant"

HISTOGRAMS TYPES

Height-Balanced & Hybrid



Looking to identify and record "**popular**" values.
Must take lowest and highest values

Example:
rows=60
size (buckets)=12 *('FOR ALL COLUMNS SIZE 12')*

Oracle must sort the data set from the sample (*this is an expensive operation*)

We have 60 rows and 12 buckets. The Histogram will be build by capturing every (60/12) 5th value

TEAM	COUNT(*)	TEAM	COUNT(*)
Ajax	1	Juventus	4
Arsenal	3	Legia Warsaw	1
Athletic Bilbao	1	Liverpool	4
Athletico Madrid	1	Man City	3
Barcelona	6	Man Utd	9
BayernM	11	Maribor	1
Borussia Dortmund	2	Paris Saint Germain	1
Chelsea	4	Porto	1
Dinamo Zagreb	1	Real Madrid	1
Hertha Berlin	1	Sunderland	1
Internazionale	3		

HISTOGRAMS TYPES

Height-Balanced:

lowest, highest
and every 5th value

TEAM	COUNT(*)
Ajax	1
Arsenal	3
Athletic Bilbao	1
Athletico Madrid	1
Barcelona	6
BayernM	**11**
Borussia Dortmund	2
Chelsea	4
Dinamo Zagreb	1
Hertha Berlin	1
Internazionale	3

TEAM	COUNT(*)
Juventus	4
Legia Warsaw	1
Liverpool	4
Man City	3
Man Utd	**9**
Maribor	1
Paris Saint Germain	1
Porto	1
Real Madrid	1
Sunderland	1

0	**Ajax**			
1	Ajax,	Arsenal,	Arsenal,	Arsenal,	**Athletic Bilbao**,
2	Athletico Madrid,	Barcelona,	Barcelona,	Barcelona,	**Barcelona**,
3	Barcelona,	Barcelona,	BayernM,	BayernM,	BayernM,
4	BayernM,	BayernM,	BayernM,	BayernM,	**BayernM**,
5	BayernM,	BayernM,	BayernM,	Borussia Dortmund,	**Borussia Dortmund**,
6	***Chelsea***,	***Chelsea***,	***Chelsea***,	***Chelsea***,	**Dinamo Zagreb**,
7	Hertha Berlin,	Internazionale,	Internazionale,	Internazionale,	**Juventus**,
8	Juventus,	Juventus,	Juventus,	Legia Warsaw,	**Liverpool**,
9	Liverpool,	Liverpool,	Liverpool,	Man City,	**Man City**,
10	Man City,	Man Utd,	Man Utd,	Man Utd,	Man Utd,
11	Man Utd,	Man Utd,	Man Utd,	Man Utd,	**Man Utd**,
12	Maribor,	Paris Saint Germain,	Porto,	Real Madrid,	**Sunderland**

COLUMN_NAM	ENDPOINT_NUMBER	ENDPOINT_VALUE	ENDPOINT_ACTUAL_VALUE	
TEAM	0	339656953492716000000000000000000000	Ajax	
TEAM	1	339860328594721000000000000000000000	Athletic Bilbao	
TEAM	2	344668049167009000000000000000000000	Barcelona	
TEAM	**4**	**344668604398841000000000000000000000**	**BayernM**	**<= popular (> 1 entry needed)**
TEAM	5	344952008489148000000000000000000000	Borussia Dortmund	
TEAM	6	355214584638972000000000000000000000	Dinamo Zagreb	
TEAM	7	386612389769899000000000000000000000	Juventus	
TEAM	8	396753594576542000000000000000000000	Liverpool	
TEAM	9	401782997701836000000000000000000000	Man City	
TEAM	**11**	**401782997701836000000000000000000000**	**Man Utd**	**<= popular**
TEAM	12	433342427351037000000000000000000000	Sunderland	

HISTOGRAMS TYPES

TEAM	COUNT(*)
Ajax	1
Arsenal	3
Athletic Bilbao	1
Athletico Madrid	1
Barcelona	6
BayernM	**11**
Borussia Dortmund	2
Chelsea	4
Dinamo Zagreb	1
Hertha Berlin	1
Internazionale	3

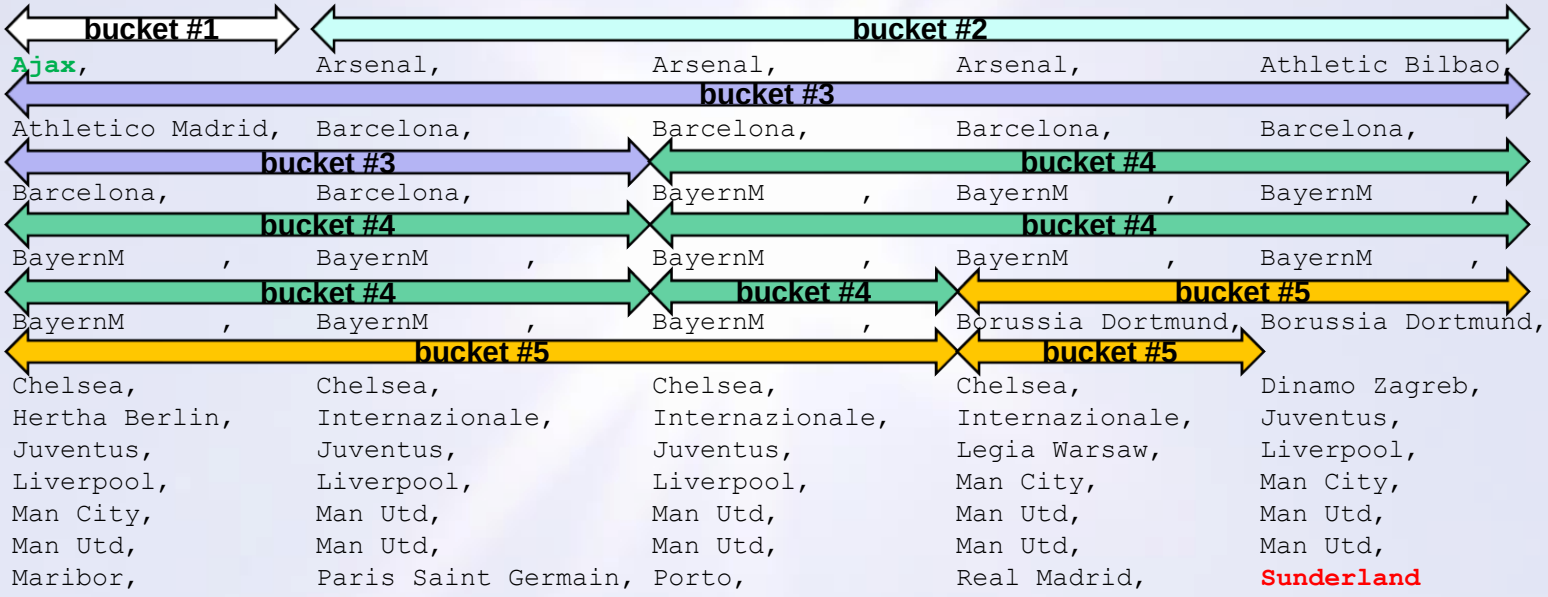
TEAM	COUNT(*)
Juventus	4
Legia Warsaw	1
Liverpool	4
Man City	3
Man Utd	**9**
Maribor	1
Paris Saint Germain	1
Porto	1
Real Madrid	1
Sunderland	1

Hybrid:

lowest, highest

and every 5th value BUT

- *no value can occupy more than 1 bucket*
- *duplicate values are moved into the same bucket*
- *buckets can vary in size*



HISTOGRAMS TYPES

Hybrid:

TEAM	COUNT(*)
Ajax	1
Arsenal	3
Athletic Bilbao	1
Athletico Madrid	1
Barcelona	1
BayernM	**6**
Borussia Dortmund	**11**
Chelsea	2
Dinamo Zagreb	4
Hertha Berlin	1
Internazionale	1
Juventus	4
Legia Warsaw	1
Liverpool	4
Man City	**Man Utd**
Maribor	3
Paris Saint Germain	**9**
Porto	1
Real Madrid	1
Sunderland	1

```
   (bucket size)
 1 ( 1) Ajax
 5 ( 4) Arsenal, Arsenal, Arsenal, Athletic Bilbao,
12 ( 7) Athletico Madrid, Barcelona, Barcelona, Barcelona, Barcelona, Barcelona, Barcelona,
23 (11) BayernM, BayernM, BayernM, BayernM, BayernM, BayernM, BayernM, BayernM, BayernM, BayernM, BayernM,
29 ( 6) Borussia Dortmund, Borussia Dortmund, Chelsea,  Chelsea,  Chelsea, Chelsea,
34 ( 5) Dinamo Zagreb, Hertha Berlin, Internazionale, Internazionale, Internazionale,
38 ( 4) Juventus, Juventus, Juventus, Juventus,
43 ( 5) Legia Warsaw,  Liverpool, Liverpool, Liverpool, Liverpool,
55 (12) Man City, Man City, Man City, Man Utd, Man Utd, Man Utd, Man Utd, Man Utd, Man Utd, Man Utd, Man Utd, Man Utd
57 ( 2) Maribor,  Paris Saint Germain
58 ( 2) Porto,
60 ( 1) Real Madrid, Sunderland
```

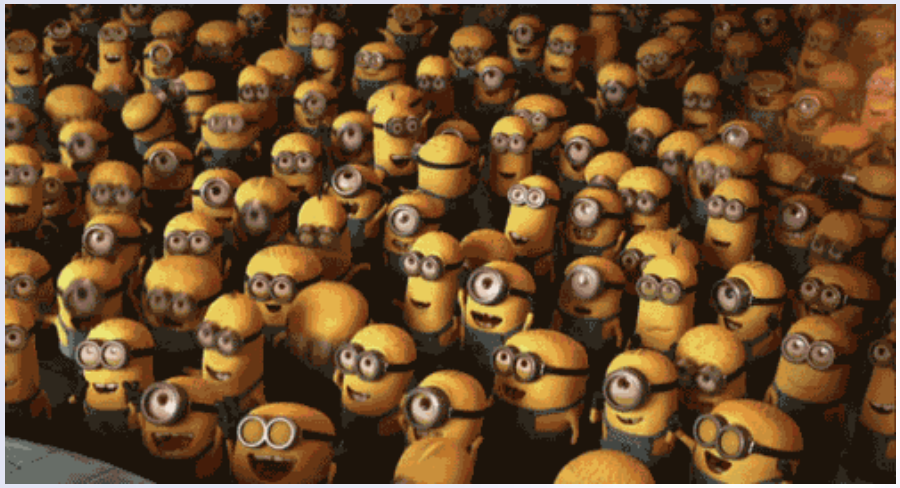
COLUMN_NAM	ENDPOINT	ENDPOINT_VALUE	DECODE	ENDPOINT_REPEAT_COUNT	
TEAM	1	33965695349271600000000000000000000000	Ajax	1	
TEAM	5	33986030932283700000000000000000000000	Athlet	1	
TEAM	12	34466804916700900000000000000000000000	Barcel	6	<= Popular (> orig BktSize 5)
TEAM	23	34466860439884100000000000000000000000	BayernM	11	<= Popular
TEAM	29	35000129572891300000000000000000000000	Chelse	4	
TEAM	34	38127795759262100000000000000000000000	Intern	3	
TEAM	38	38661238976989900000000000000000000000	Juvent	4	
TEAM	43	39675359457654200000000000000000000000	Liverp	4	
TEAM	55	40178299770183600000000000000000000000	Man Ut	9	<= Popular
TEAM	57	41736020705997200000000000000000000000	Paris	1	
TEAM	58	41764416419377000000000000000000000000	Portn	1	
TEAM	60	43334242735103700000000000000000000000	Sunder	1	

ENDPOINT_REPEAT_COUNT=How many values are there associated with the endpoint?

Hybrid Histogram is Basically a Frequency for **Popular** Values

HISTOGRAMS TYPES

Hybrid and Height Balanced Histograms



It's all about being *POPULAR*!

CARDINALITY CALCS

Cardinality Calculation (12.2C) are undocumented, complex & different for each Histogram type.

They are subject to change with no notice.

Frequency

If the Predicate Value is in the Histogram = Num Rows** stored in the Histogram
If the Predicate Value is NOT in the Histogram = Least Popular Value from Histogram / 2

Top Frequency

Value in Histogram = Num Rows stored in the Histogram
Value not in Histogram = Least Popular Value from Histogram
(I don't think the "/2" is applied in 12.2 but is applied in 12.1)

***Actually: Num Rows In The Histogram* x *(Num Rows in Table / Sample Size)*

CARDINALITY CALCS

Height-Balanced and Hybrid are all about Popular Values:

Height Balanced

NewDensity= ((Bucket# - PopularBucket#) / Bucket#) / (NDV/PopValueCnt)

Popular = Num Rows * (Num Buckets for Value / Total Num Buckets)
Non-Popular with Endpoint = Num Rows * *NewDensity*
Non-Popular without Endpoint = Num Rows * *NewDensity / 2 (not sure about the "/ 2" in 12.2)*

NOTE: Because is takes 2 buckets to identify a Popular value, your precision is at best ½ the number of buckets

Hybrid

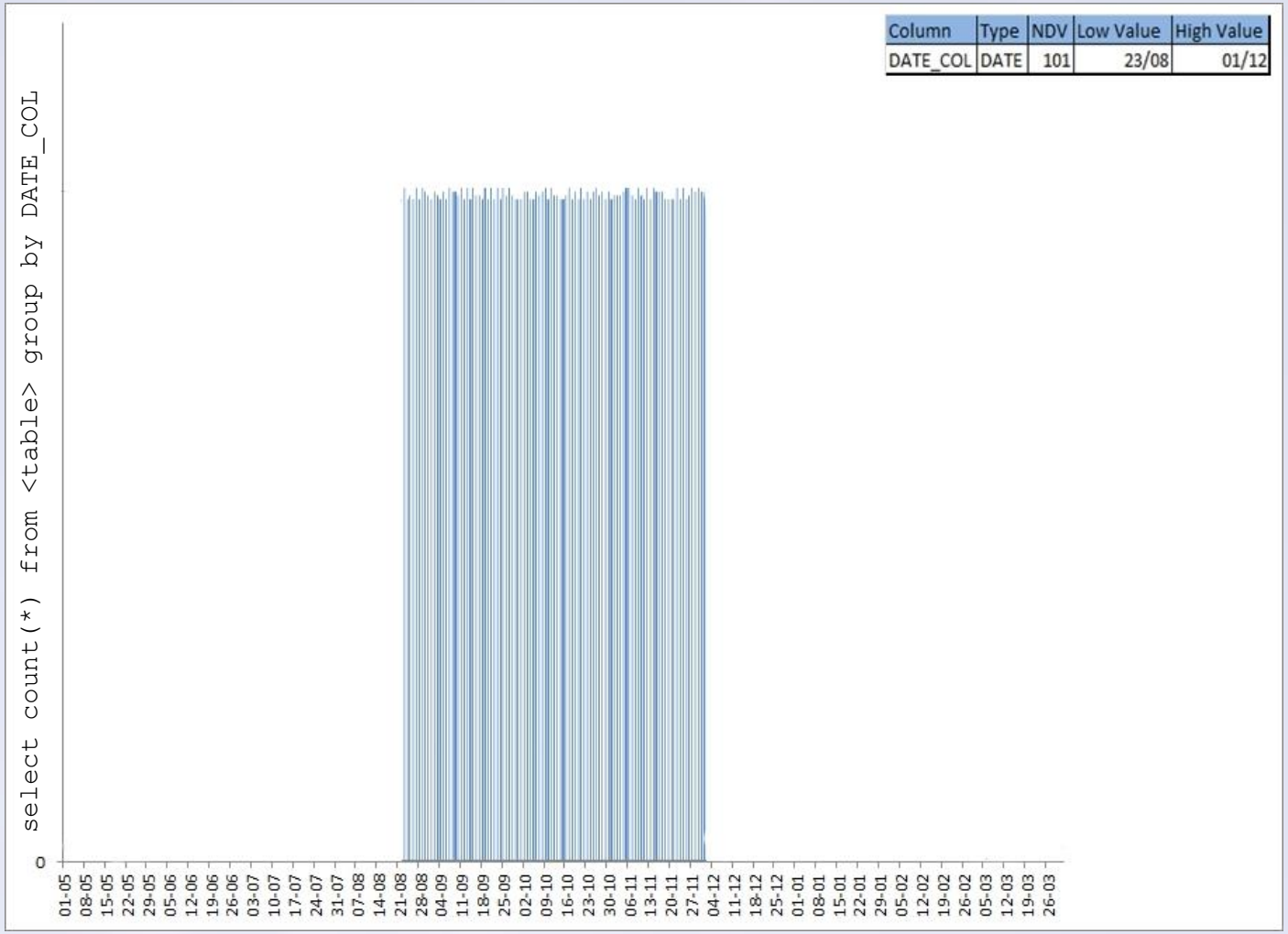
Popular = ENDPOINT_REPEAT_COUNT * (NUM_ROWS / SAMPLE_SIZE)
Non-Popular with Endpoint = Num Rows * the largest of
[*NewDensity* or
ENDPOINT_REPEAT_COUNT/SAMPLE_SIZE]
Non-Popular without Endpoint = Num Rows * *NewDensity*

HISTOGRAM MYTHS

Your plan is always better with a histogram

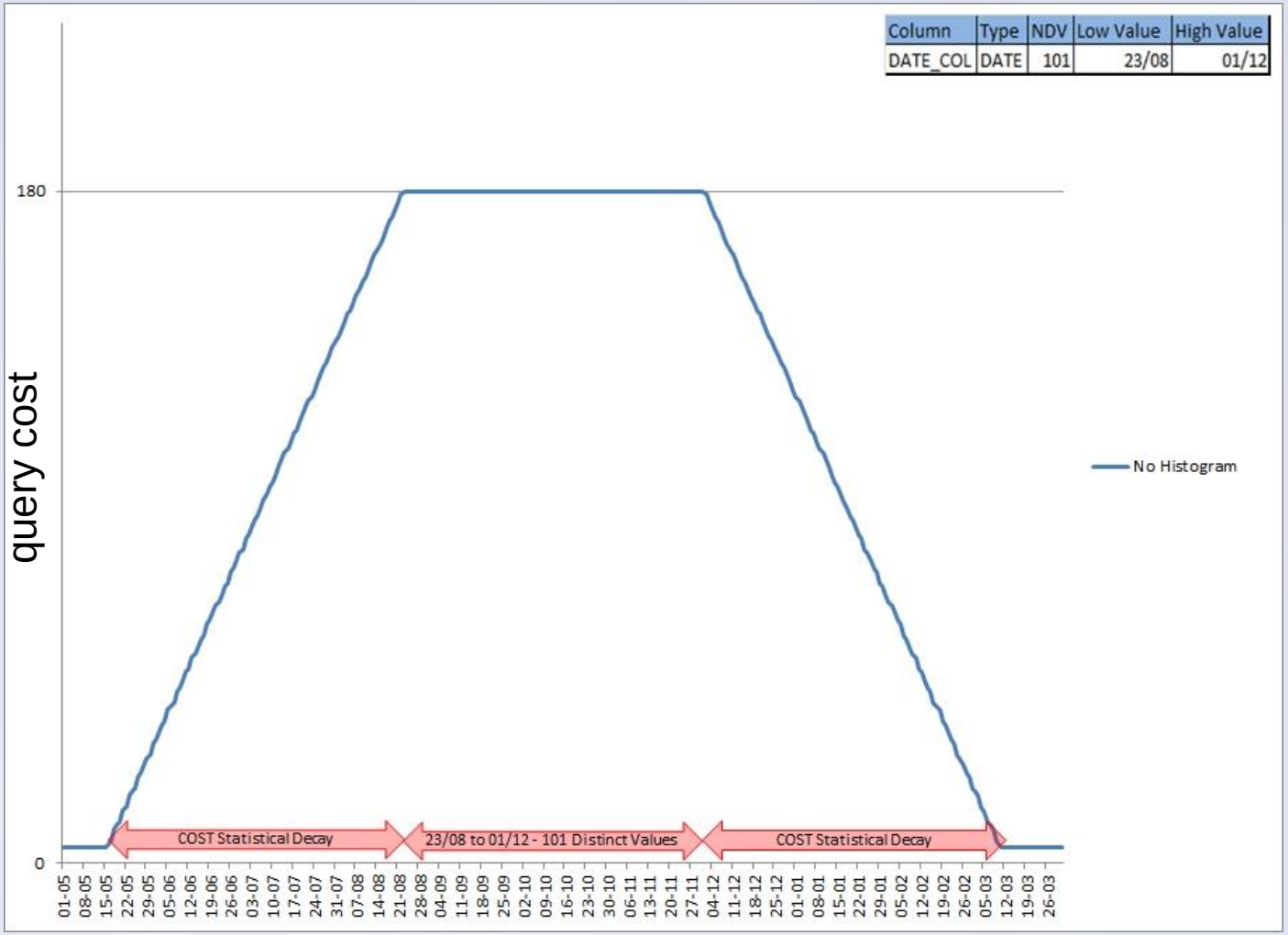
HISTOGRAM MYTHS

Well, here's a potential threat...



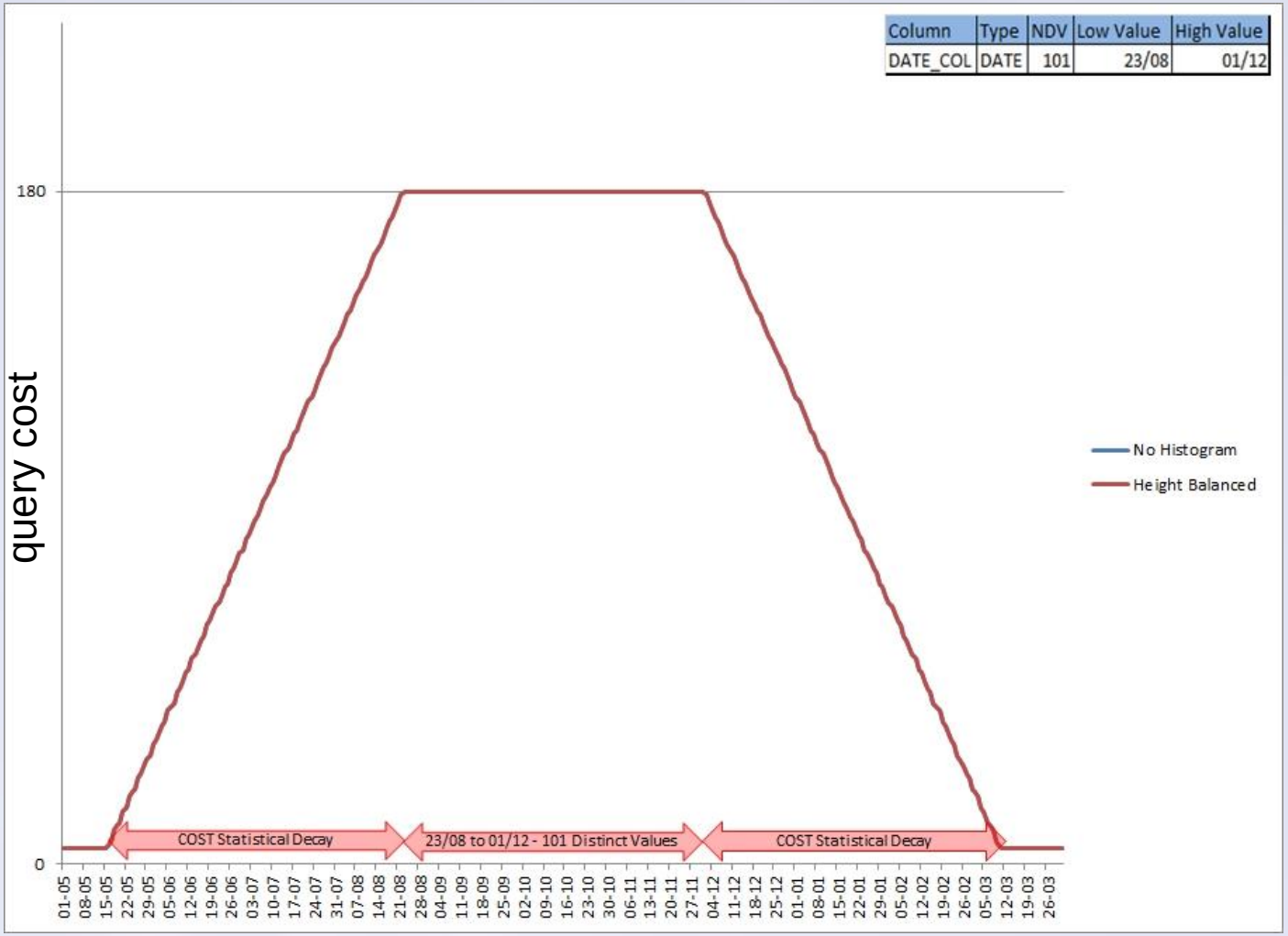
HISTOGRAM MYTHS

Well, here's a potential threat... Statistical Decay



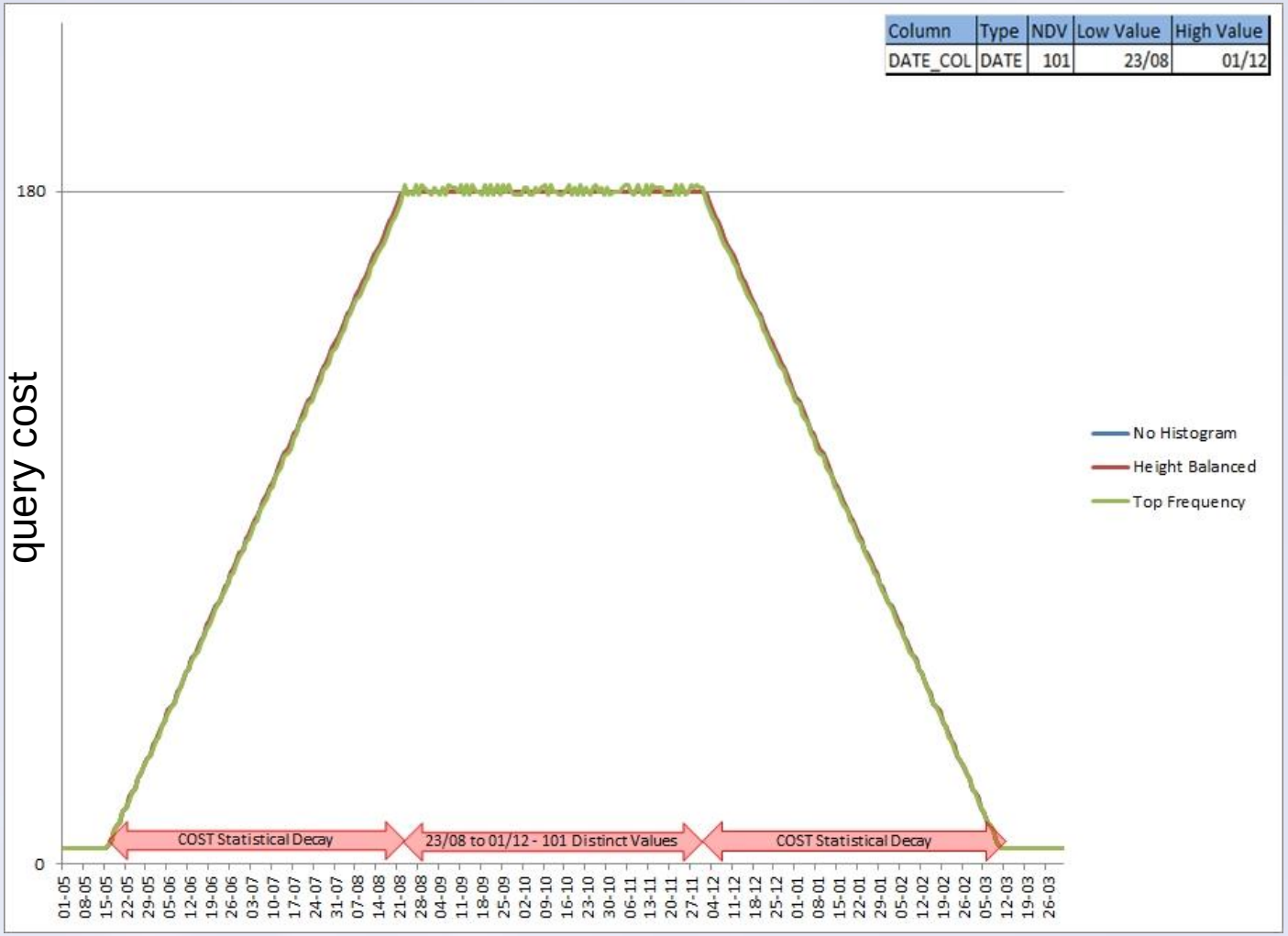
HISTOGRAM MYTHS

Well, here's a potential threat...



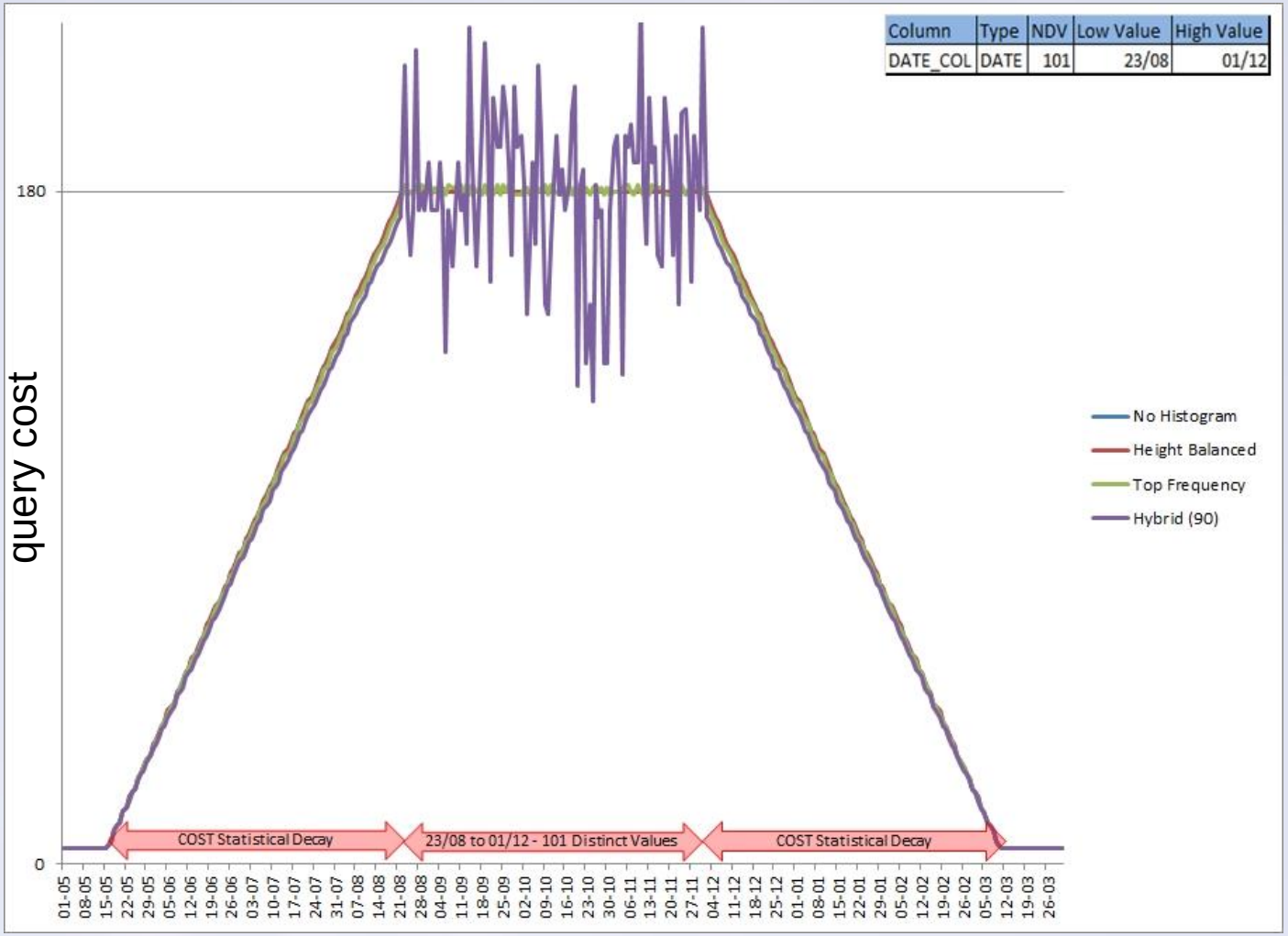
HISTOGRAM MYTHS

Well, here's a potential threat...



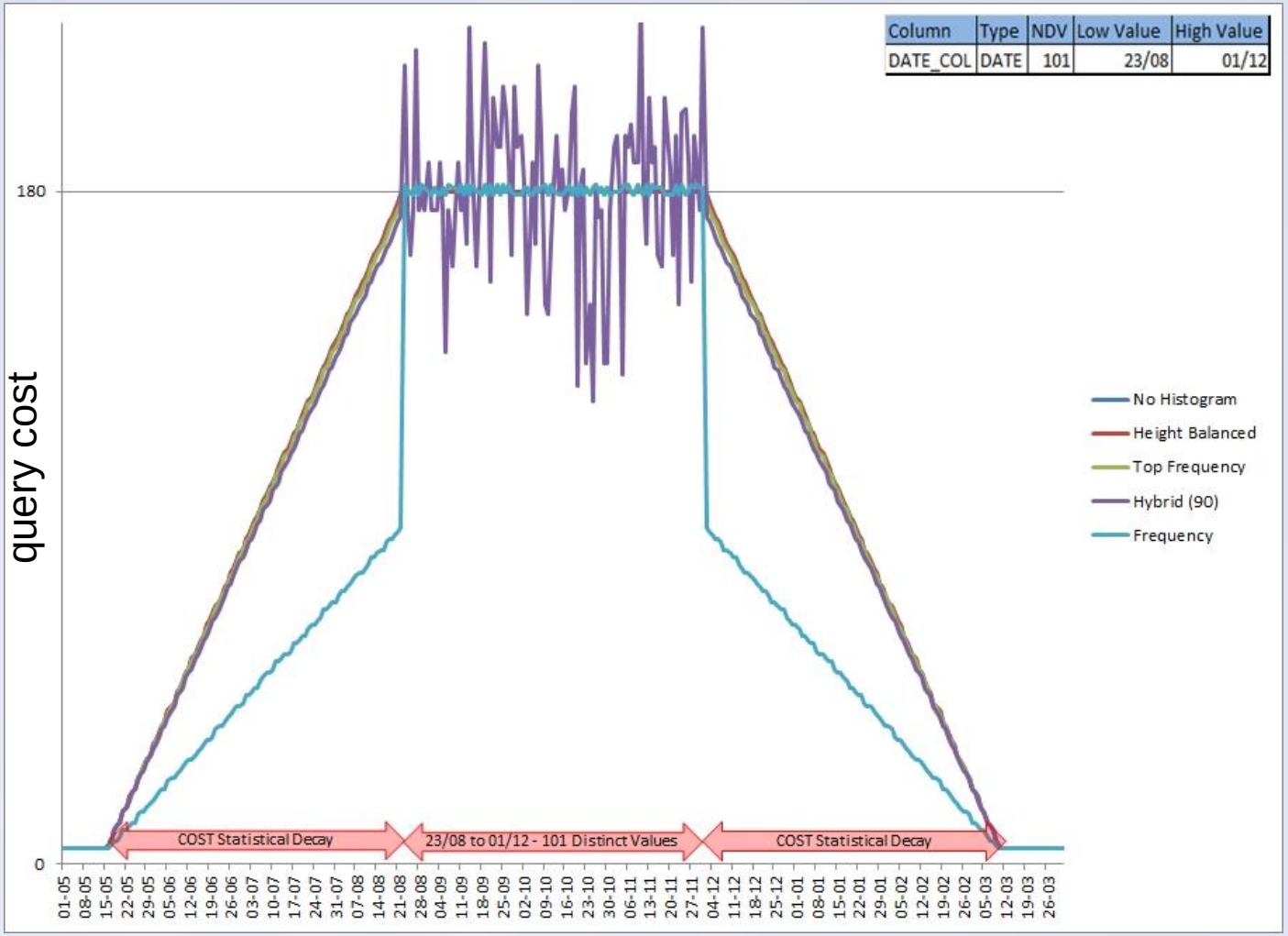
HISTOGRAM MYTHS

Well, here's a potential threat...



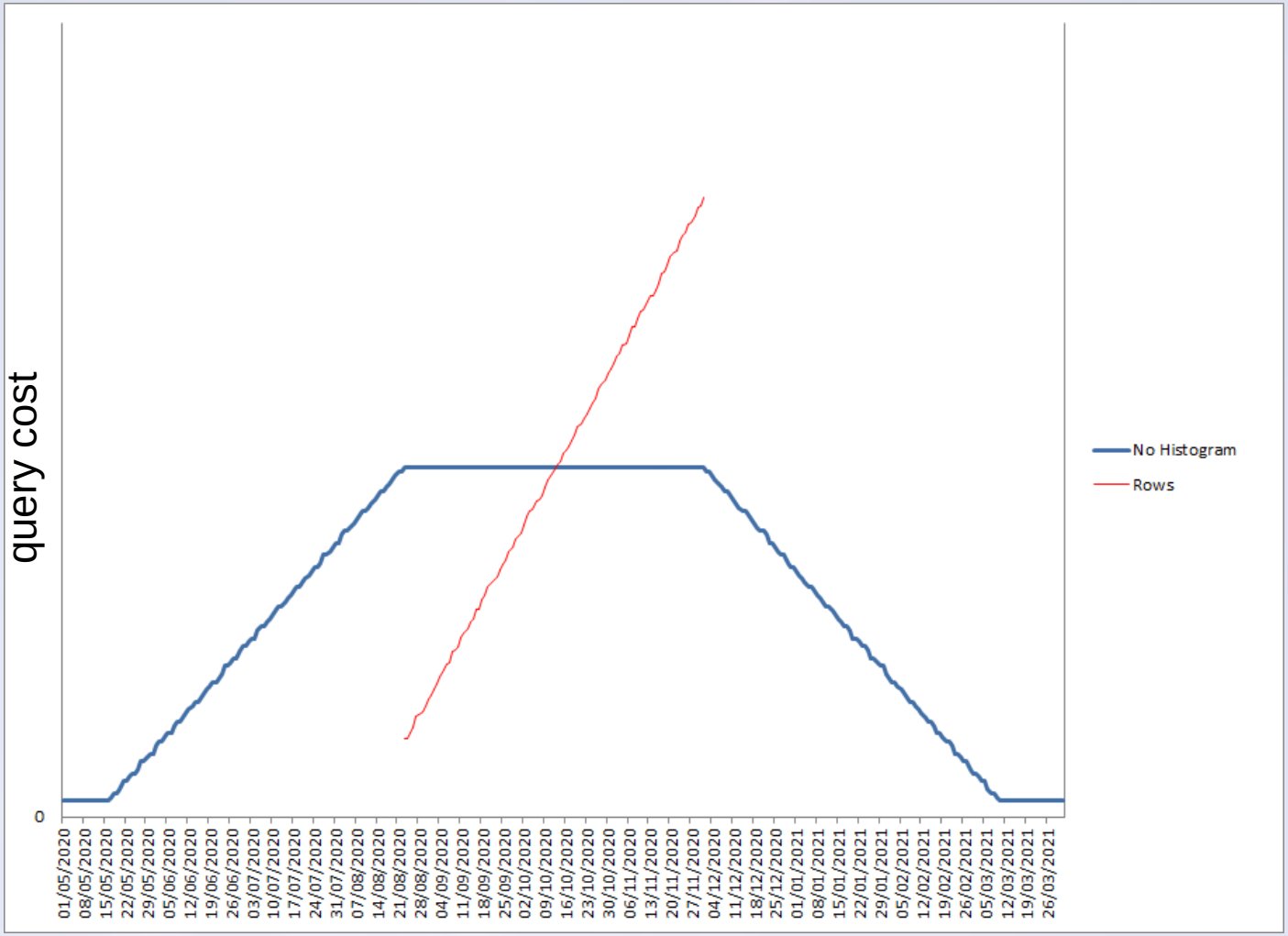
HISTOGRAM MYTHS

Well, here's a potential threat...



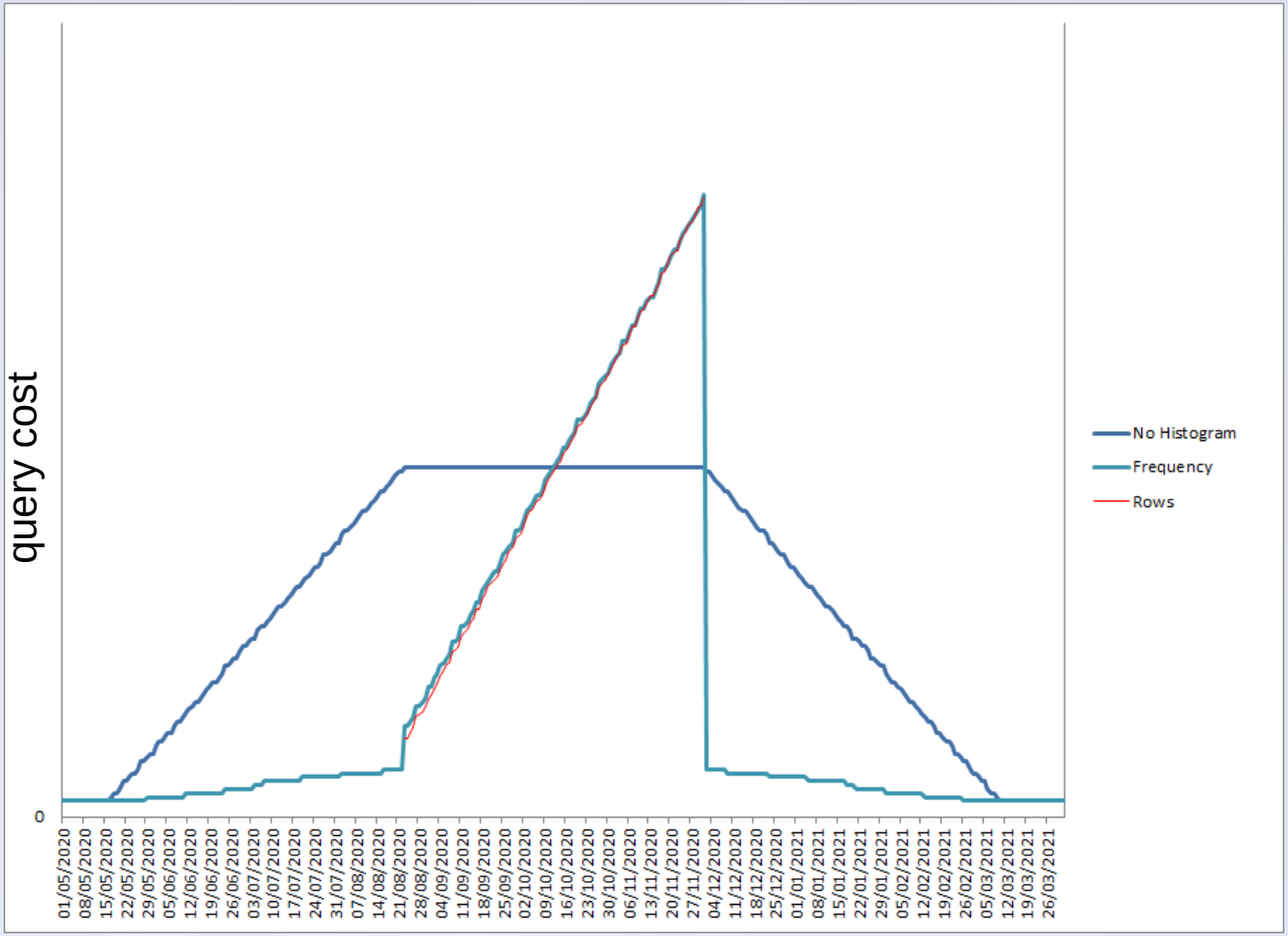
HISTOGRAM MYTHS

But with a different data set...



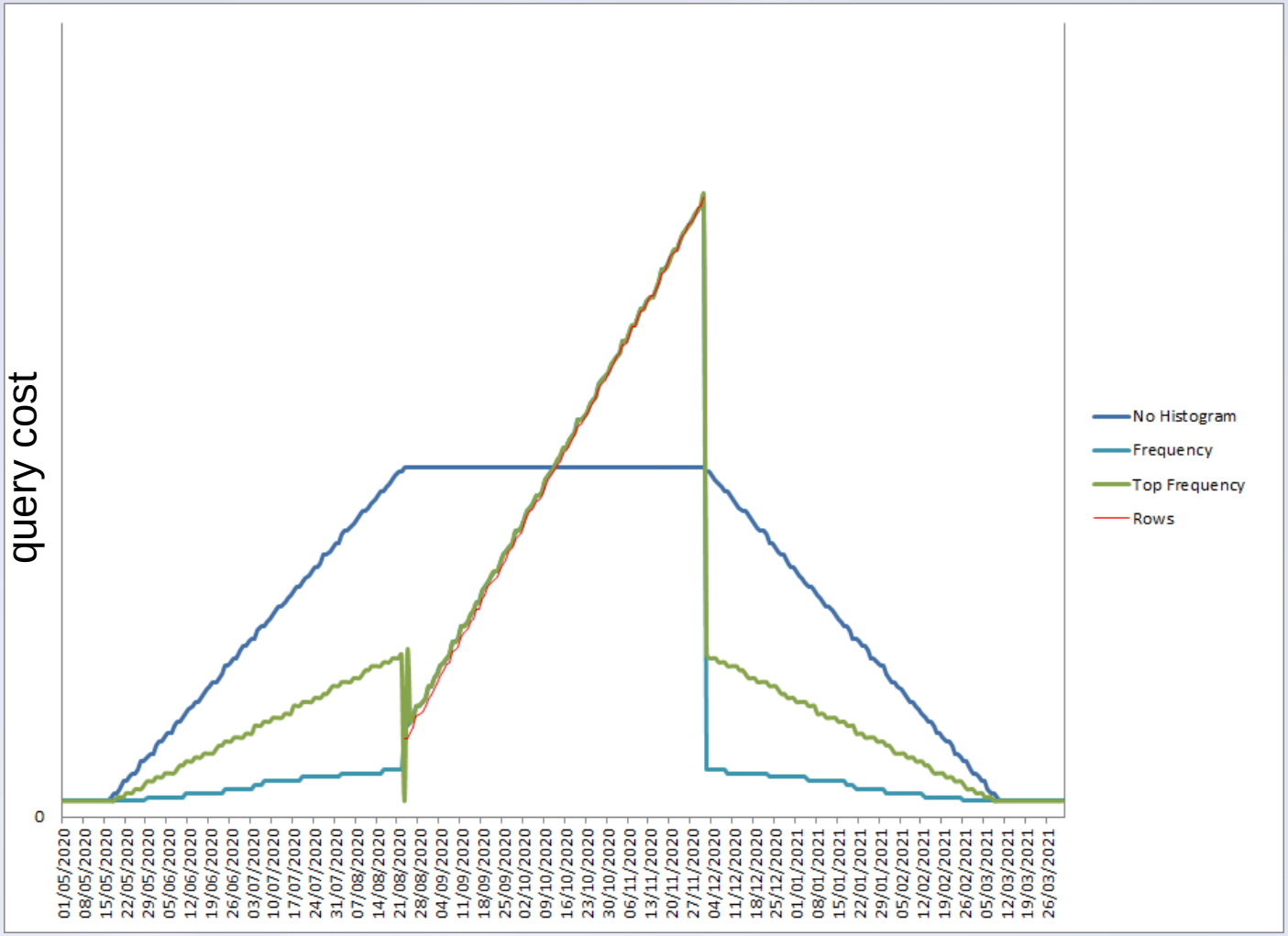
HISTOGRAM MYTHS

But with a different data set...



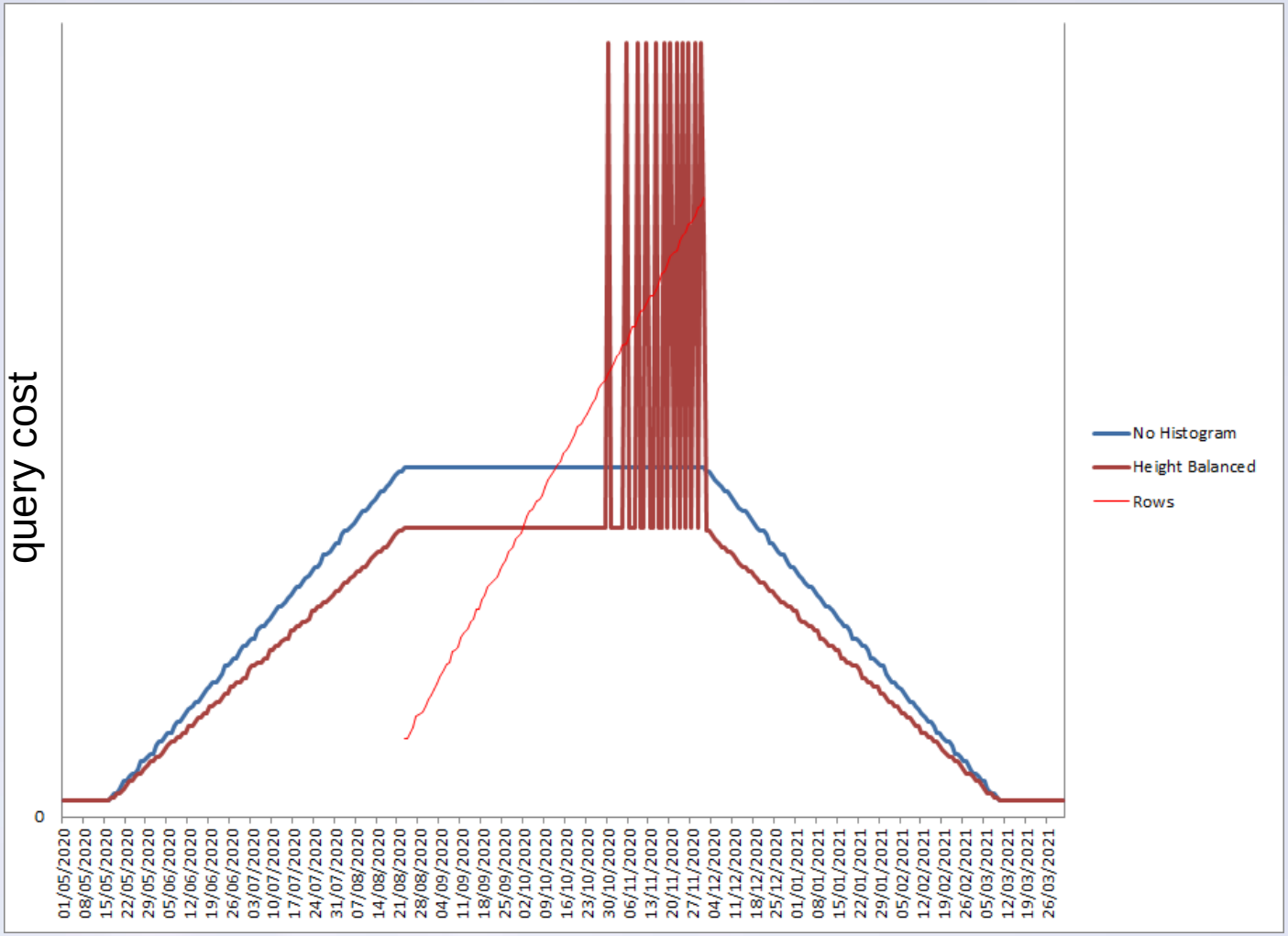
HISTOGRAM MYTHS

But with a different data set...



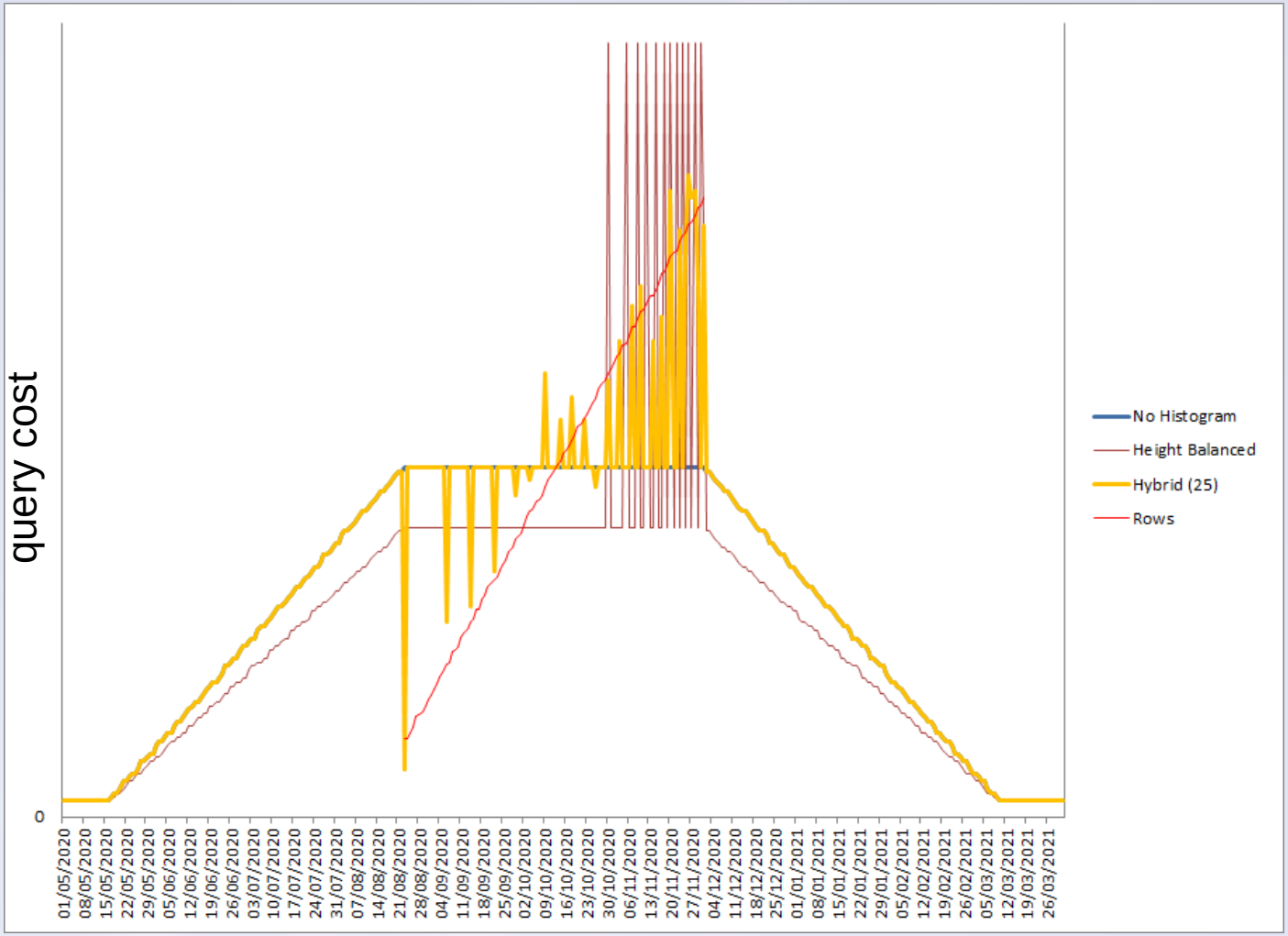
HISTOGRAM MYTHS

But with a different data set...



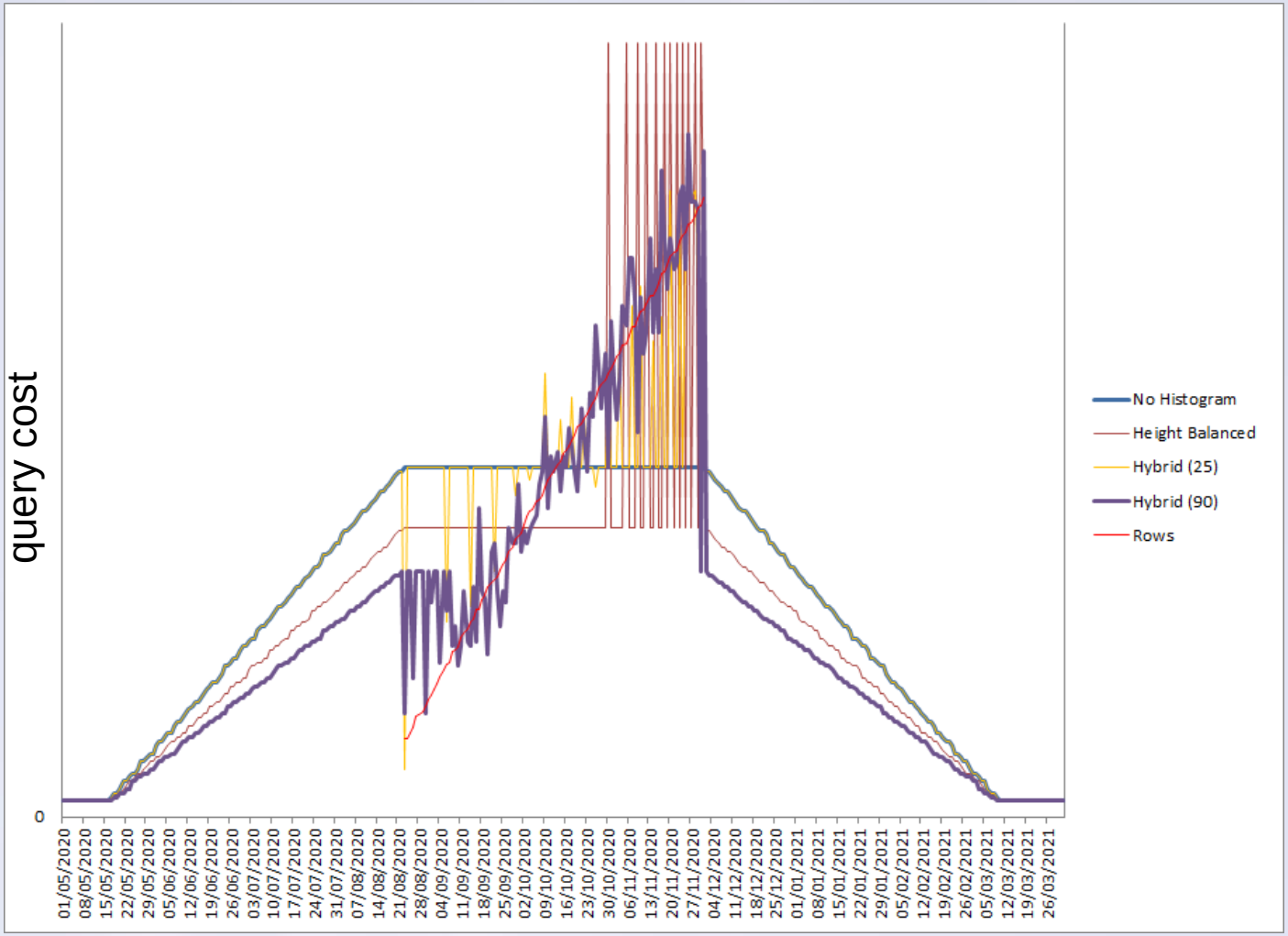
HISTOGRAM MYTHS

But with a different data set... aren't (new) histograms good!



HISTOGRAM MYTHS

But with a different data set...



WHAT TO DO?

so what should we do to minimise the threats?

SIZE REPEAT

minimise the number of histograms?

FOR ALL COLUMNS SIZE REPEAT

SIZE REPEAT

```
exec dbms_stats.set_global_prefs('method_opt',
'for all columns size repeat');
```

This was a **great** idea in Oracle 11.2 and earlier.
It stopped more Histograms appearing, but maintained the ones currently in place.

Oracle have changed the algorithm in 12C

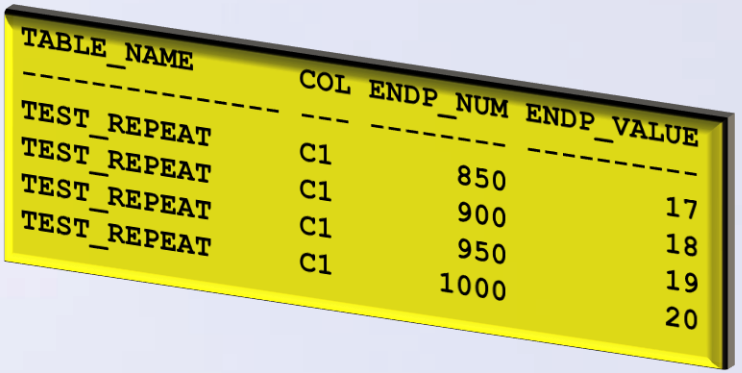
It now uses the CURRENT number of buckets as a MAXIMUM, with **disastrous** consequences

This is an undocumented change, except on Maria Colgan's Blog from April 2013 (does not reference versions):
https://blogs.oracle.com/optimizer/how-does-the-methodopt-parameter-work

> **REPEAT** ensures a histogram will only be created for any column that already has one. If the table is a partitioned table, repeat ensures a histogram will be created for a column that already has one on the global level. However, this is not a recommended setting, as the number of buckets currently in each histogram will limit the maximum number of buckets used for the newly created histograms. Lets assume there are 5 buckets currently in a histogram. When the histogram is re-gathered with SIZE REPEAT, the newly created histogram will use at most 5 buckets and may not been of good quality.

SIZE REPEAT

```
create table TEST_REPEAT (c1 number not null);
insert into  TEST_REPEAT (c1) select mod (rownum,20)+1 from dba_objects where rownum < 1001;


exec dbms_stats.gather_table_stats
    (null,'TEST_REPEAT',method_opt=>'FOR ALL COLUMNS SIZE AUTO'); [after selects, to create the histogram]

exec dbms_stats.gather_table_stats
    (null,'TEST_REPEAT',method_opt=>'FOR ALL COLUMNS SIZE REPEAT');

select table_name,column_name,num_distinct,num_buckets,histogram from dba_tab_columns
 where table_name = 'TEST_REPEAT';


TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ---------- ---------------
TEST_REPEAT     C1                   20         20 FREQUENCY


select TABLE_NAME,COLUMN_NAME,ENDPOINT_NUMBER,ENDPOINT_VALUE from user_tab_histograms
 where endpoint_value > 16 order by TABLE_NAME,ENDPOINT_NUMBER;

TABLE_NAME      COLUMN_NAM ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- ---------- --------------- --------------
TEST_REPEAT     C1                    850             17
TEST_REPEAT     C1                    900             18
TEST_REPEAT     C1                    950             19
TEST_REPEAT     C1                   1000             20
```



SIZE REPEAT

11.2

```
TABLE_NAME       COL ENDP_NUM ENDP_VALUE
---------------- --- ------- ---------
TEST_REPEAT      C1      850        17
TEST_REPEAT      C1      900        18
TEST_REPEAT      C1      950        19
TEST_REPEAT      C1     1000        20
```

```
delete from TEST_REPEAT where c1 = 19;

exec dbms_stats.gather_table_stats (null,'TEST_REPEAT',
     method_opt=>'FOR ALL COLUMNS SIZE REPEAT');

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  19          19 FREQUENCY

insert into TEST_REPEAT (c1) values (19);

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  20          20 FREQUENCY

insert into TEST_REPEAT (c1) values (21);

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  21          21 FREQUENCY
```

SIZE REPEAT

12.2

```
TABLE_NAME       COL ENDP_NUM ENDP_VALUE
---------------- --- ------- ---------
TEST_REPEAT      C1      850
TEST_REPEAT      C1      900        17
TEST_REPEAT      C1      950        18
TEST_REPEAT      C1     1000        19
                                    20
```

```
delete from TEST_REPEAT where c1 = 19;

exec dbms_stats.gather_table_stats (null,'TEST_REPEAT',
     method_opt=>'FOR ALL COLUMNS SIZE REPEAT');

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  19          19 FREQUENCY

insert into TEST_REPEAT (c1) values (19);

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  20          19 TOP-FREQUENCY

insert into TEST_REPEAT (c1) values (21);

TABLE_NAME      COLUMN_NAM NUM_DISTINCT NUM_BUCKETS HISTOGRAM
--------------- ---------- ------------ ----------- ---------------
TEST_REPEAT     C1                  21          19 HYBRID
```

DO NOT USE: SIZE REPEAT

```
method_opt: for all columns size repeat
```

should NOT be used in ORACLE 12C

so what should we do?

CONCLUSION : BA

for
Data Warehouse Systems
Business Intelligence Systems
Data Analytics Systems
where
unique large and complex queries are common

embrace your histograms

*May be w*orth specific intervention on the
BIG FACT TABLES
to optimize resource usage

CONCLUSION : OLTP

for larger high volume OLTP systems
probably using bind variables
where performance *consistency* is critical
there is a reasonable argument to switch off histograms
globally and use **SET_TABLE_PREFS**
to control histograms for individual columns

```
exec dbms_stats.set_global_prefs(
    'method_opt',
    'for all columns size 1');

exec dbms_stats.set_table_prefs(
     ownname=>'NEIL',
     tabname=>'TEST_REPEAT',
     pname  =>'method_opt',
     pvalue =>'for all columns size 1
               for columns size auto col1,col2,col3');
```

USER_TAB_STAT_PREFS

TABLE_NAME	PREFERENCE_NAME	PREFERENCE_VALUE
TEST_REPEAT	METHOD_OPT	FOR ALL COLUMNS SIZE 1 FOR COLUMNS SIZE AUTO col1,col2,col3

THANKS

Thanks to the knowledge, research, blogs and white papers of:

- Maria Colgan
- The rest of the core Ask ToM team (Chris and Connor)
- Nigel Bayliss
- Jonathan Lewis
- Amit Poddar
- Christian Antognini
- Mohamed Houri
- Wolfgang Breitling
- Probably Martin Widlake too
- anyone else who has blogged or presented about Histograms, Approximate NDV or any other related subject I've ingested over the years

and

- The Oracle Manuals

BLOG: http://chandlerdba.com
EMAIL: neil@chandler.uk.com
TWITTER: @chandlerDBA

ANY QUESTIONS

BLOG: http://chandlerdba.com
EMAIL: neil@chandler.uk.com
TWITTER: @chandlerDBA