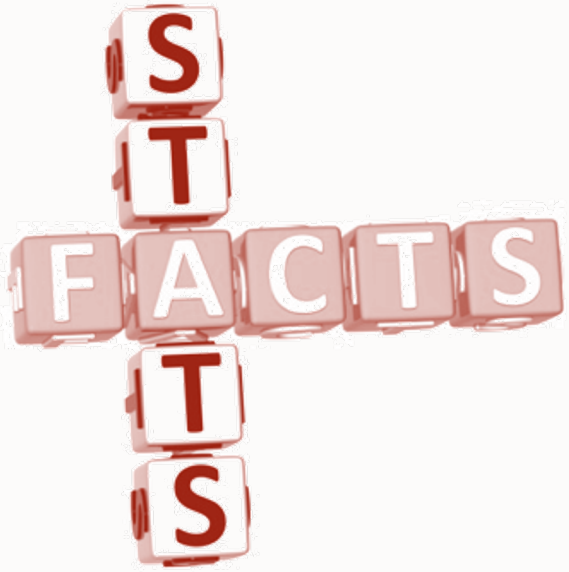
ORACLE

Best Practices For Managing Optimizer Statistics

Part 2



Maria Colgan
Master Product Manager
Oracle Database
February 2020
@SQLMaria

SQL Maria

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Common Statistics Questions and Fears

- What statistics do I actual need?
- How should I gather statistics?
- What sample size should I use?
- When should I gather statistics?
- Should I use the auto stats job?
- What statistics don’t I need?
- How do I speed up statistics gathering?

Warning on Best Practices

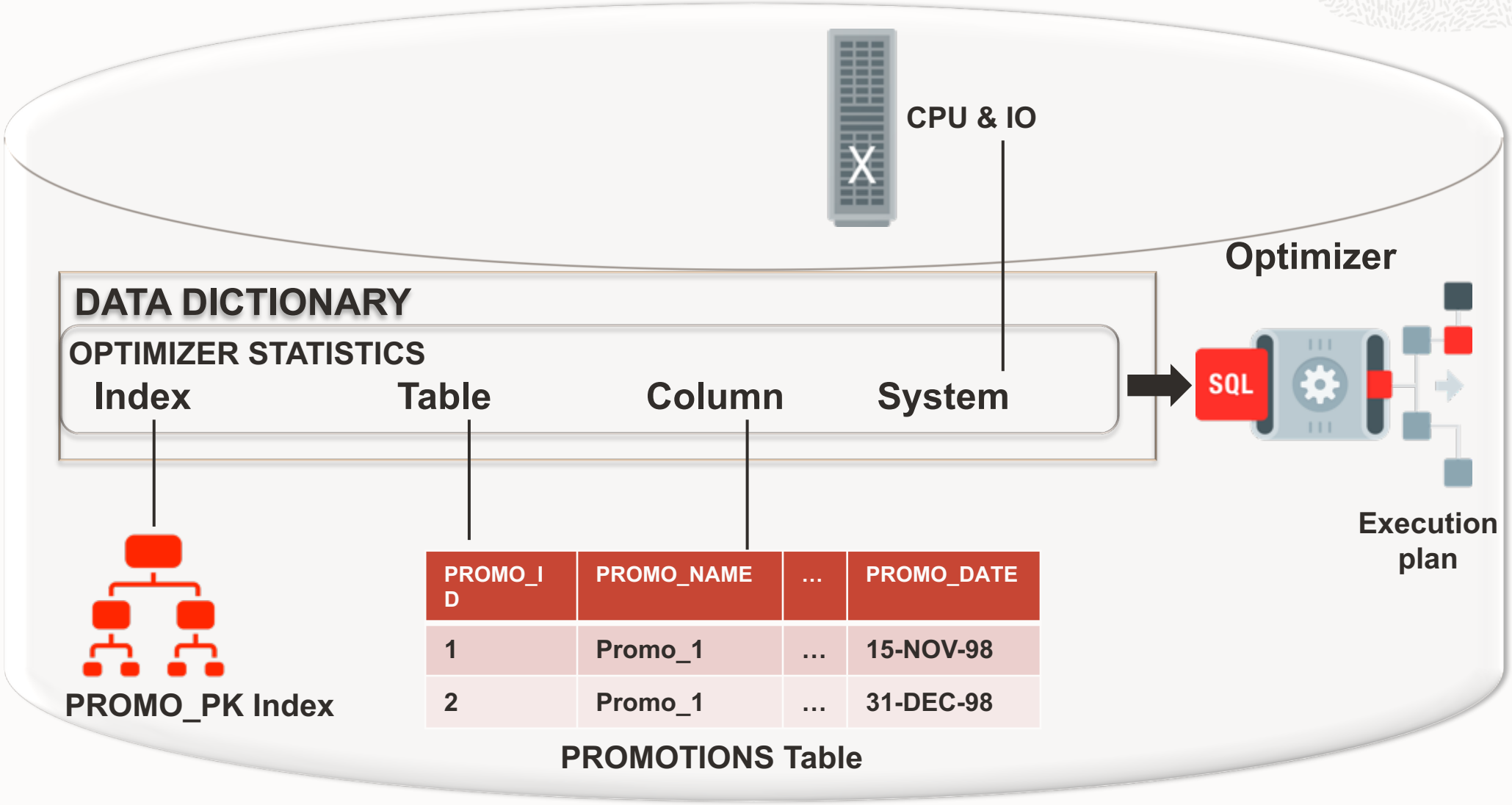
- Best practice recommendations work for ~80% of systems 90% of the time
- There are **always** exceptions
- Your goal should be to provide the optimizer a **representative set** of stats

- To help you navigate this presentation:

- Indicates a recommendation

- Indicates a warning of potential size effects

Optimizer Statistics



AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

How to gather statistics

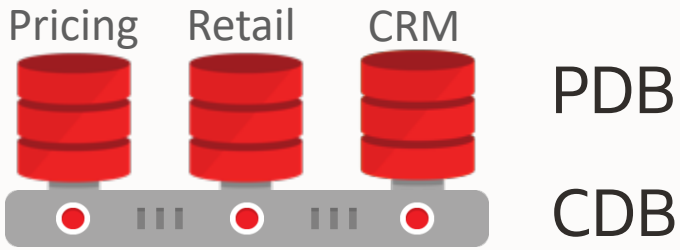
Automatic Statistics Gathering

- Oracle automatically collect statistics for all database objects, which are missing statistics or have stale statistics
- The stats gathering AutoTask runs during a predefined maintenance window
- Internally prioritizes the database objects
  - Both user schema and dictionary tables and fixed object stats from 12c onwards
  - Objects that need updated statistics most are processed first
- Controlled by `DBMS_AUTO_TASK_ADMIN` package or Enterprise Manager
- Monitor using `DBA_AUTOTASK_*` views

How to gather statistics

Automatic Statistics Gathering in a PDB environment

- Starting in 12c there is a new Multitenant architecture
- Each PDB has its own stats gathering autotask & its own maintenance window (default all same time)



- From 12.2 onwards use `AUTOTASK_MAX_ACTIVE_PDBS` to control how many PDBs can run autotasks concurrently (Default 2)
- Use `ENABLE_AUTOMATIC_MAINTENANCE_PDB` to disable the maintenance window at the PDB level (Default `TRUE`)

NOTE: From 20c onward Multitenant architecture is the only supported architecture

How to gather statistics

Use DBMS_STATS Package

`ANALYZE` command is deprecated

- DO NOT USE
- Only good for row chaining

The `GATHER_*_STATS` procedures take 13 parameters

- Ideally you should only set the first 2-3 parameters
  - `SCHEMA NAME`
  - `TABLE NAME`
  - `PARTITION NAME`

How to gather statistics

Use DBMS_STATS Package

Your gather statistics commands should be this simple

```
BEGIN
dbms_stats.Gather_database_stats();
END; /

BEGIN
dbms_stats.Gather_schema_stats('SH');
END; /

BEGIN
dbms_stats.Gather_table_stats('SH', 'SALES');
END; /
```

How to gather statistics

Changing default parameter values for gathering statistics

Occasionally default parameter values may need to change

For example - features not automatically on by default

- Incremental Statistics
  - Ability to accurate generate global statistics from partition level statistics
  - Controlled by the parameter `INCREMENTAL` (default is `FALSE`)
- Concurrent Statistics Gathering
  - Ability to gather statistics on multiple objects concurrently under a `GATHER_SCHEMA_STATS` command
  - Controlled by the parameter `CONCURRENT` (default is `FALSE`)

How to gather statistics

Changing default parameter values for gathering statistics

Can change the default value at the global level

- DBMS_STATS.SET_GLOBAL_PREFS
- This changes the value for all existing objects and any new objects

```
BEGIN
  dbms_stats.Set_global_prefs('INCREMENTAL', 'TRUE');
END;
/
```

Can change the default value at the table level

- DBMS_STATS.SET_TABLE_PREFS

How to gather statistics

Changing default parameter values for gathering statistics

Can change the default value at the schema level

- `DBMS_STATS.SET_SCHEMA_PREFS`
- Current objects in the schema only
- New objects pick up global preferences

Can change the default value at the database level

- `DBMS_STATS.SET_DATABASE_PREFS`
- Current objects in the Database only
- New objects pick up global preferences

How to gather statistics

Changing default parameter values for gathering statistics

The following parameter defaults can be changed:

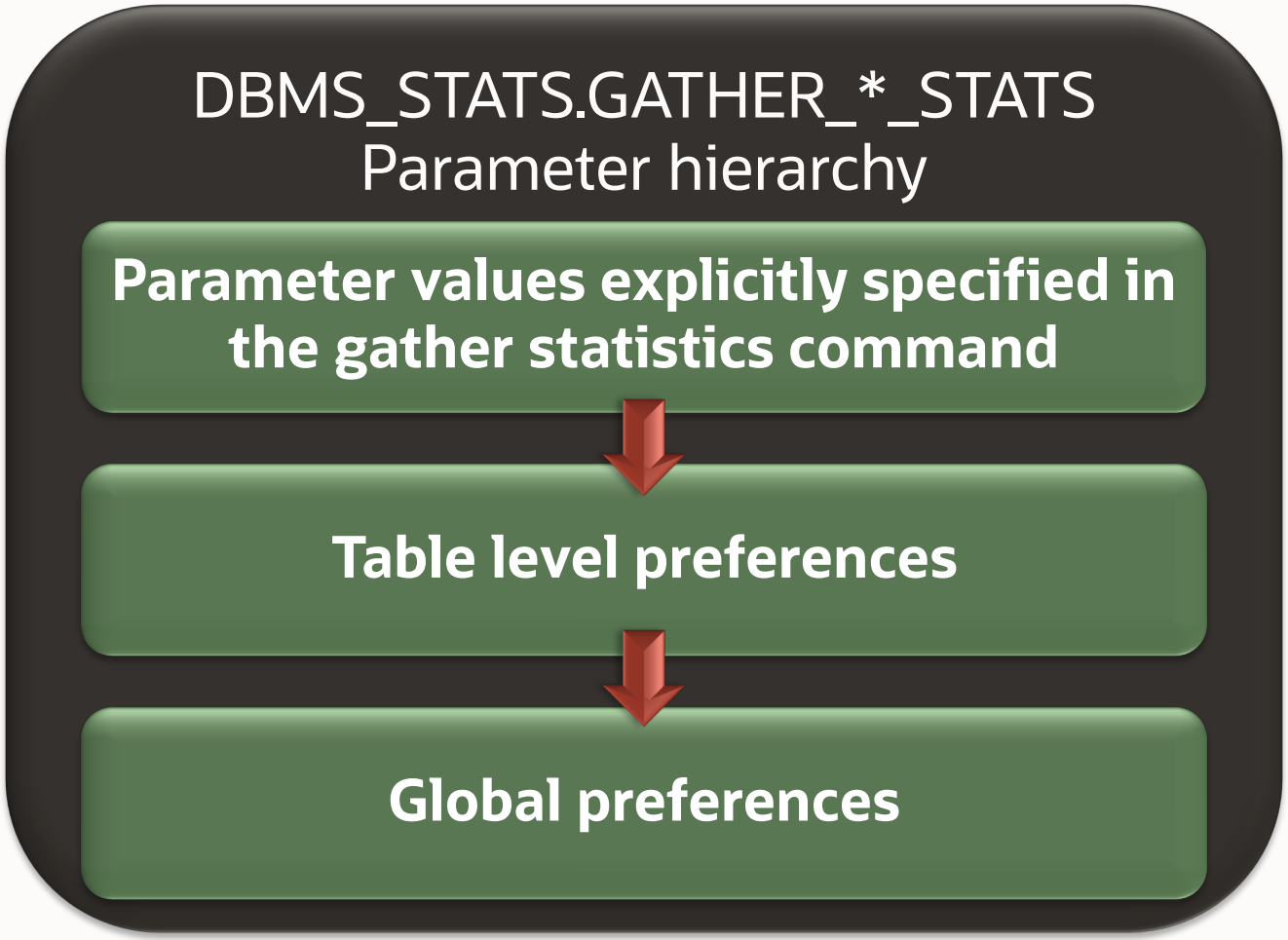
- APPROXIMATE_NDV_ALGORITHM
- AUTO_STATS_EXTENSIONS
- AUTOSTATS_TARGET*
- CASCADE
- CONCURRENT*
- DEGREE
- ESTIMATE_PERCENT
- GLOBAL_TEMP_TABLE_STATS*
- GRANULARITY
- INCREMENTAL
- INCREMENTAL_LEVEL**
- INCREMENTAL_STALENESS
- METHOD_OPT
- NO_INVALIDATE
- OPTIONS
- PERFERENCE_OVERRIDES_PARAMETER
- PUBLISH
- STALE_PERCENT
- STAT_CATEGORY*
- TABLE_CACHED_BLOCKS
- WAIT_TIME_TO_UPDATE_STATS*

* Parameters that can only be set at a Global level

**Parameter that can only be set at a table level

How to gather statistics

Hierarchy for parameter values



How to gather statistics

Sample size

1 most commonly asked question

"What sample size should I use?"

Controlled by `ESTIMATE_PRECENT` parameter

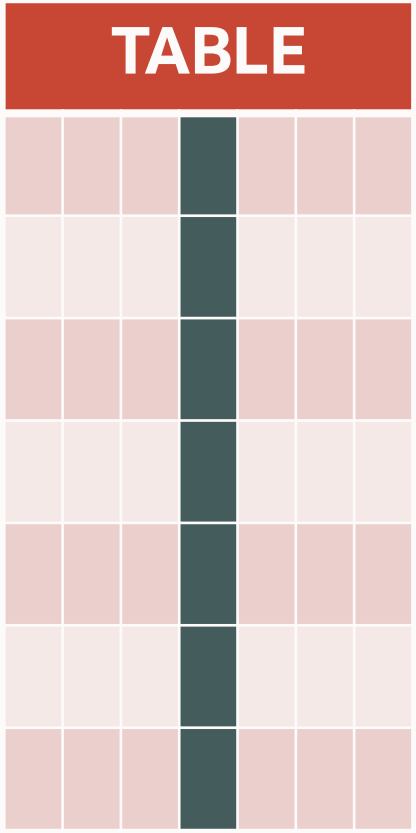
From 11g onwards use default value **`AUTO_SAMPLE_SIZE`**

- New hash-based algorithm
- Speed of a 10% sample
- Accuracy of 100% sample

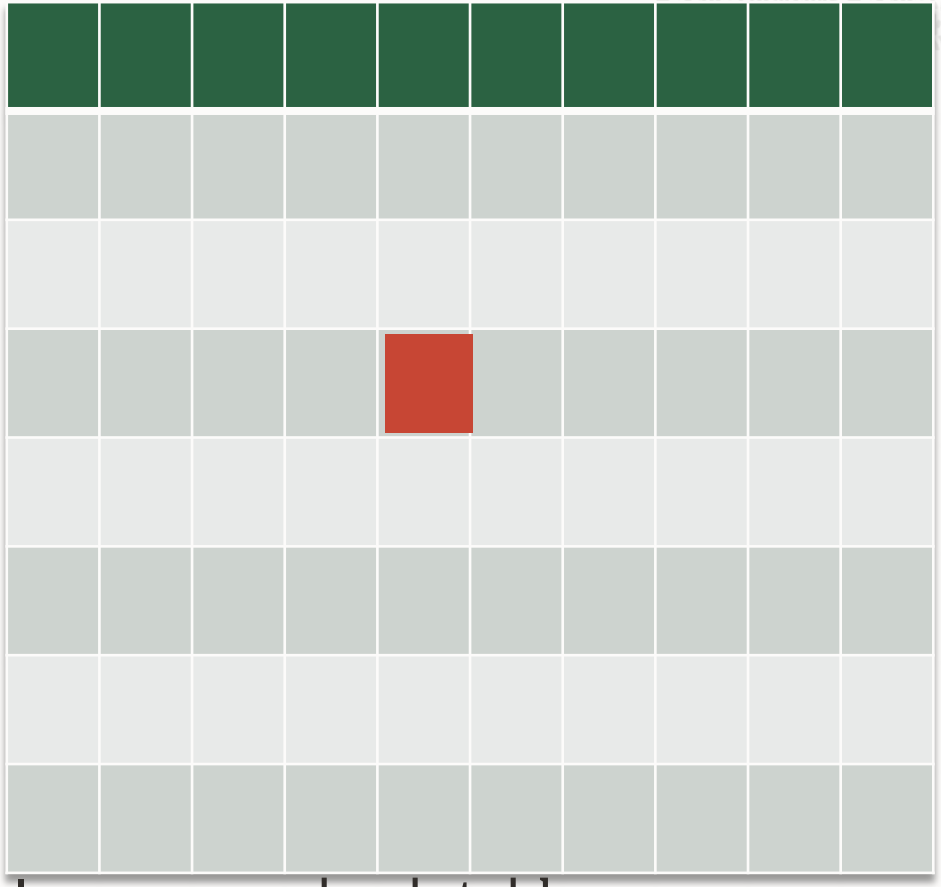
More info in the following paper http://dl.acm.org/citation.cfm?id=1376721

How to gather statistics

How AUTO_SAMPLE_SIZE works



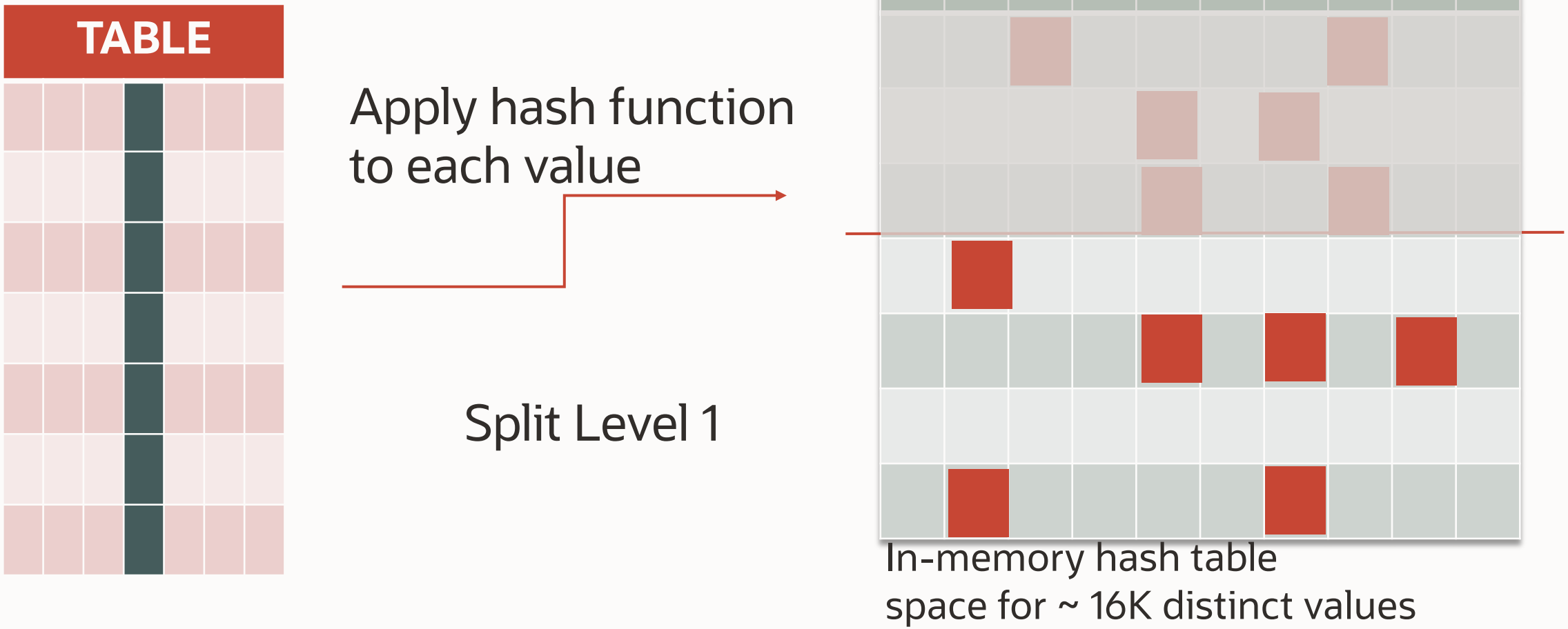
Apply hash function
to each value



In-memory hash table
space for ~ 16K distinct values

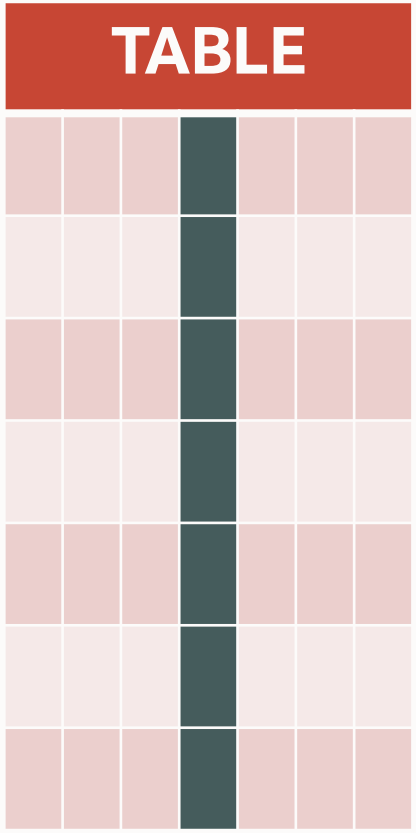
How to gather statistics

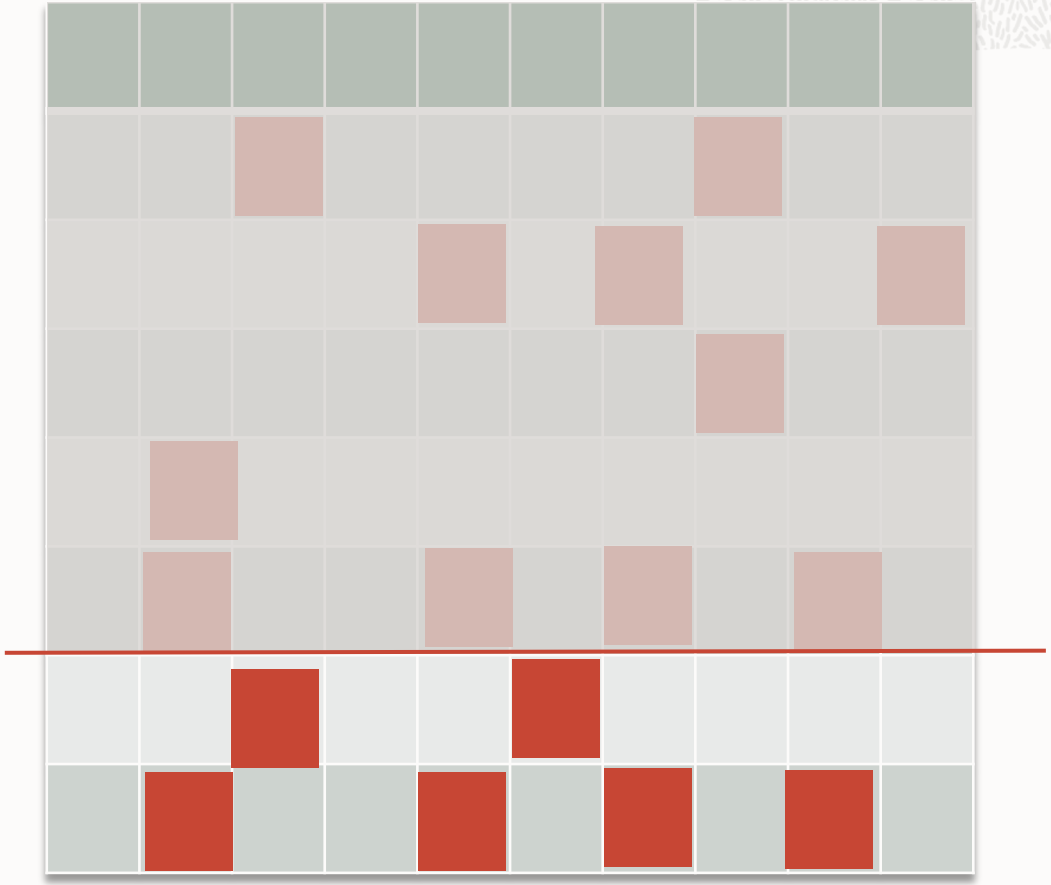
How AUTO_SAMPLE_SIZE works



How to gather statistics

How AUTO_SAMPLE_SIZE works





Split Level 2

NDV = 2^split_level

How to gather statistics

How AUTO_SAMPLE_SIZE performs

Speed of a 10% sample

Run Num	AUTO_SAMPLE_SIZE	10% SAMPLE	100% SAMPLE
1	00:02:21.86	00:02:31.56	00:08:24.10
2	00:02:38.11	00:02:49.49	00:07:38.25
3	00:02:39.31	00:02:38.55	00:07:37.83

Accuracy of 100% sample

Column Name	NDV with AUTO_SAMPLE_SIZE	NDV with 10% SAMPLE	NDV with 100% SAMPLE
C1	59852	31464	60351
C2	1270912	608544	1289760
C3	768384	359424	777942

AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

What basic statistics to collect

Table Statistics

By default the following basic table & column statistic are collected either during a CTAS or IAS command* or via a `DBMS_STATS.GATHER_*_STATS` command :

- Number of Rows++
- Number of blocks ++
- Average row length
- Min / Max values of a column ++
- Number of distinct values
- Number of nulls in column

*Online statistics gather possible from Oracle Database 12cR1
++ Real-time statistics gather possible from 19c onwards on Exadata & Cloud

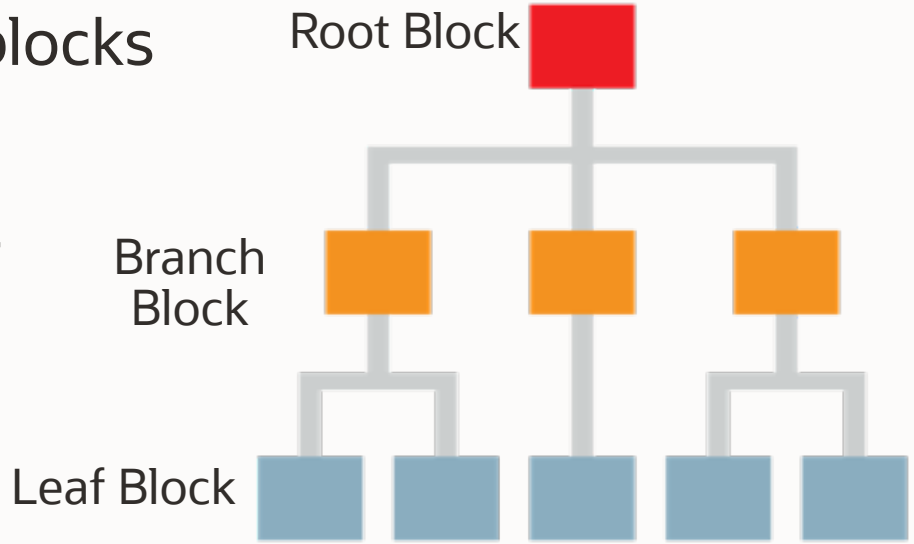
Basic index statistics collected by Oracle

Index Statistics

Index statistics are automatically gathered during creation and maintained by `DBMS_STATS.GATHER_*_STATS` commands

Index statistics include:

- Number of leaf blocks
- Branch Levels
- Clustering factor



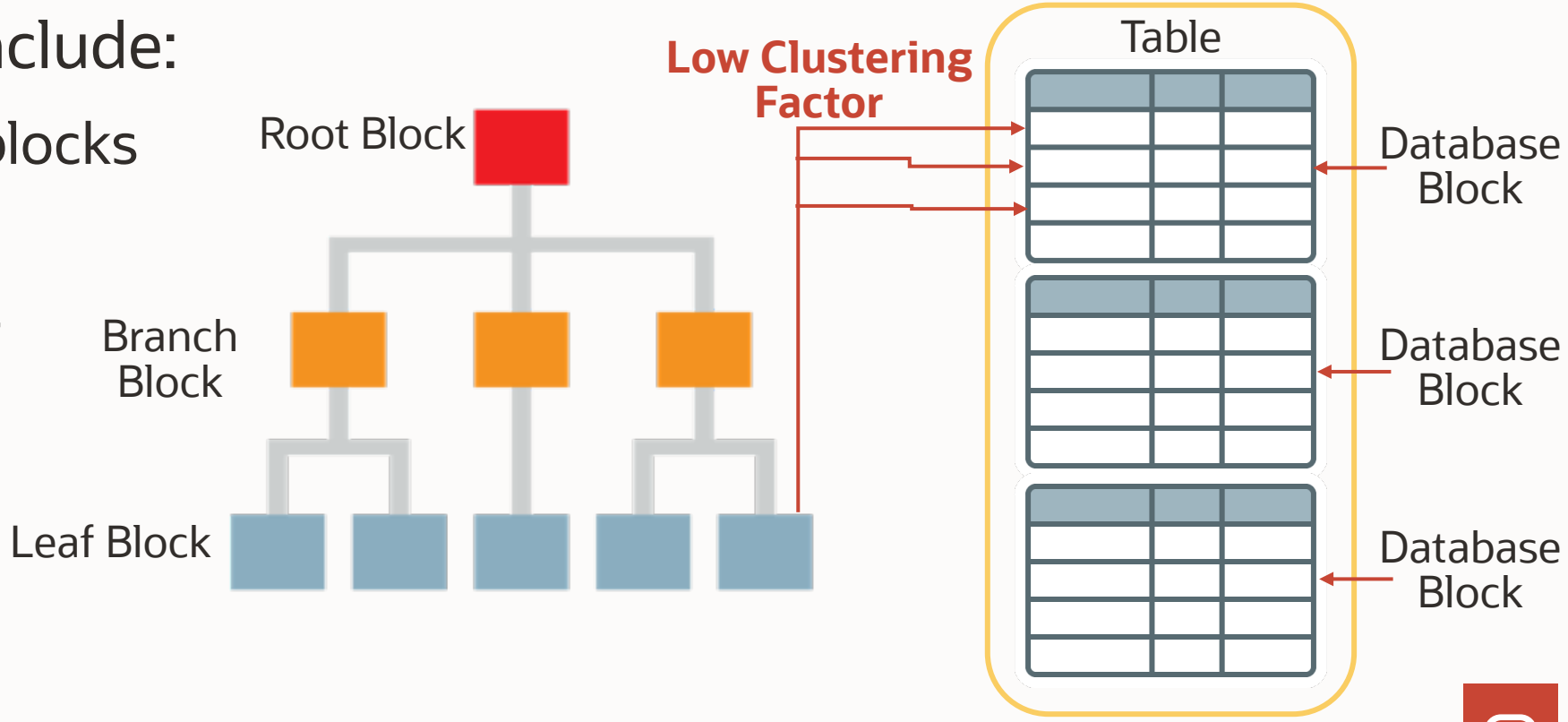
Basic index statistics collected by Oracle

Index Statistics

Index statistics are automatically gathered during creation and maintained by `DBMS_STATS.GATHER_*_STATS` commands

Index statistics include:

- Number of leaf blocks
- Branch Levels
- Clustering factor



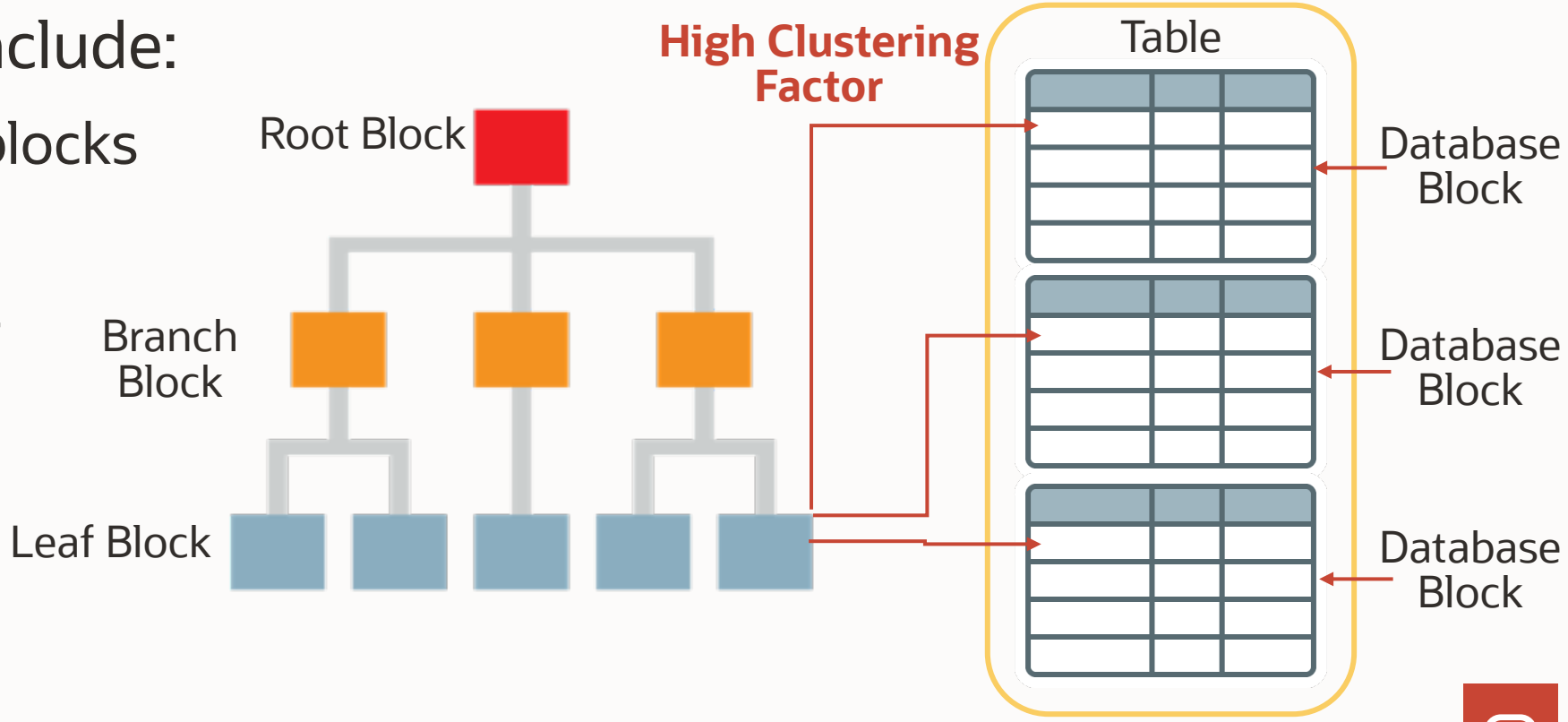
Basic index statistics collected by Oracle

Index Statistics

Index statistics are automatically gathered during creation and maintained by `DBMS_STATS.GATHER_*_STATS` commands

Index statistics include:

- Number of leaf blocks
- Branch Levels
- Clustering factor



What basic statistics to collect

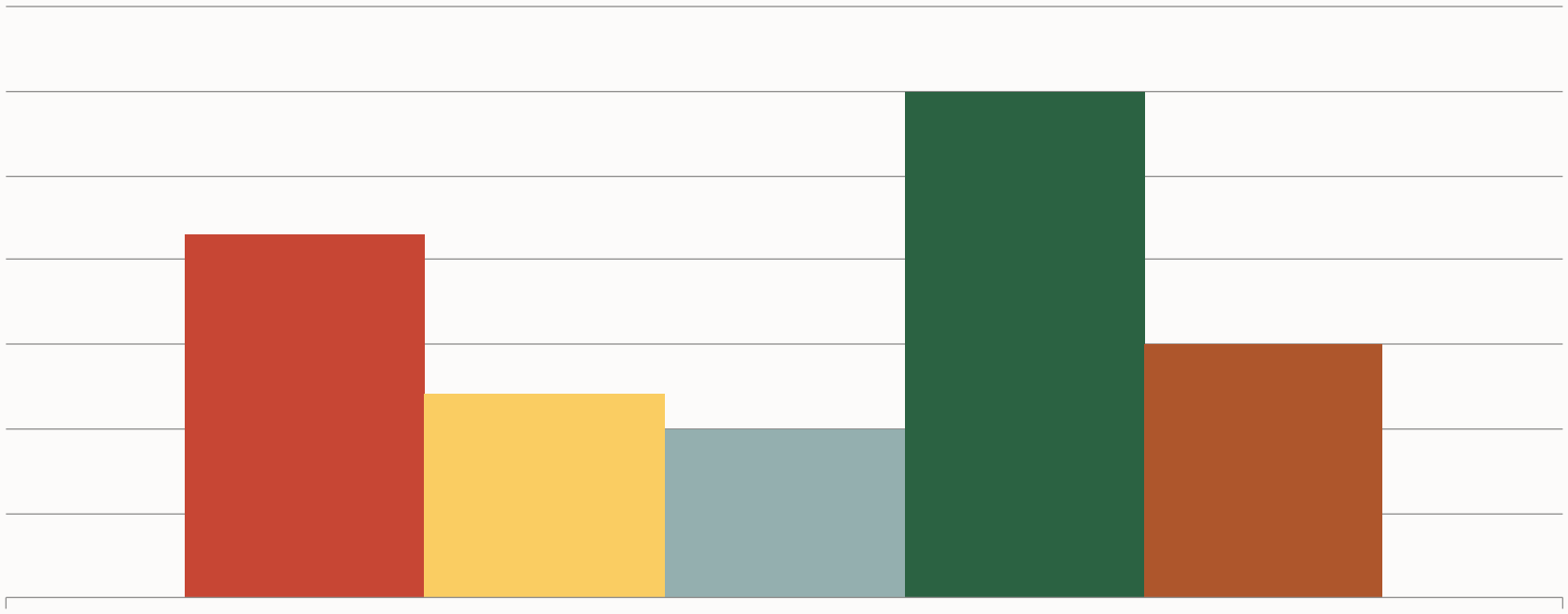
Histograms

- Histograms tell the Optimizer about the data distribution in a Column
- Creation controlled by `METHOD_OPT` parameter
- Default create histogram on any column that has been used in the `WHERE` clause or `GROUP BY` of a statement **and** has a data skew
- Relies on column usage information gathered at compilation time and stored in `SYS.COL_USAGE$`
- Four types of histograms
  - Frequency
  - Top-Frequency
  - Height balanced
  - Hybrid

Histograms

Frequency histograms (FREQUENCY)

- A frequency histogram is only created if the number of distinct values in a column (NDV) is less than 254 values
- Each distinct value will have it own bucket



Frequency histogram

Creating a frequency histogram

Step 1: SELECT job_id FROM employees ORDER BY job_id;

```
   JOB_ID
==========
AC_ACCOUNT
AC_MGR
AD_ASST
AD_PRES
AD_VP
AD_VP
FI_ACCOUNT
FI_ACCOUNT
      :
```

Creating a frequency histogram

Step2 : Allocate a histogram buckets for each pf the job_ids listed

JOB_ID	
AC_ACCOUNT	Bucket 1 has end point AC_ACCOUNT
AC_MGR	Bucket 2 has end point AC_MGR
AD_ASST	Bucket 3 has end point AD_ASST
AD_PRES	Bucket 4 has end point AD_PRES
AD_VP	Bucket 5 has end point AD_VP
AD_VP	Bucket 6 has end point AD _VP
FI_ACCOUNT	Bucket 7 has end point FI_ACCOUNT
FI_ACCOUNT	Bucket 8 has end point FI_ACCOUNT
:	

Take Note of the Bucket # that have duplicated end point values

Creating a frequency histogram

Step3 : Compress duplicate buckets to reduce the bucket count to 254 or less

JOB_ID

==========

AC_ACCOUNT — Bucket 1 has end point AC_ACCOUNT

AC_MGR — Bucket 2 has end point AC_MGR

AD_ASST — Bucket 3 has end point AD_ASST

AD_PRES — Bucket 4 has end point AD_PRES

AD_VP — Bucket 6 has end point AD _VP

FI_ACCOUNT — Bucket 11 has end point FI_ACCOUNT

:

```
SQL>  Select endpoint_number bucket_number, endpoint_value
  2  From   user_histograms
  3  Where  table_name='EMPLOYEES'
  4  And    column_name='JOB_ID';

BUCKET_NUMBER ENDPOINT_VALUE
------------- --------------
            1     3.3887E+35
            2     3.3887E+35
            3     3.3889E+35
            4     3.3889E+35
            6     3.3889E+35
           11     3.6495E+35
           12     3.6495E+35
           13     3.7552E+35
           18     3.8075E+35
           19     4.0134E+35
           20     4.0134E+35
           21     4.1705E+35
           26     4.1712E+35
           27     4.1712E+35
           32     4.3229E+35
           62     4.3229E+35
           82     4.3243E+35
          102     4.3267E+35
          107     4.3267E+35

19 rows selected.
```

NOTE: the column named endpoint_number in USER_HISTOGRAMS is the bucket number

Histograms

How Optimizer uses frequency histograms

```
SELECT *
FROM   employee
WHERE  job_id = 'AD_VP';
```

Cardinality estimate = #rows X # buckets value is in / total # of buckets

= 107 X 2 / 107 = **2**

```
-----------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES  |      1 |      2 |
|*  2 |   INDEX RANGE SCAN           | EMP_JOB_IX |      1 |      2 |
-----------------------------------------------------------------------
```

Employee Table

LAST_NAME	EMP_ID	JOB_ID
GEITZ	206	AC_ACCT
HIGGINS	205	AC__MGR
WHALEN	200	AD_ASST
KING	100	AD_PRES
KOCHHAR	101	AD_VP
DE HAAN	102	AD_VP

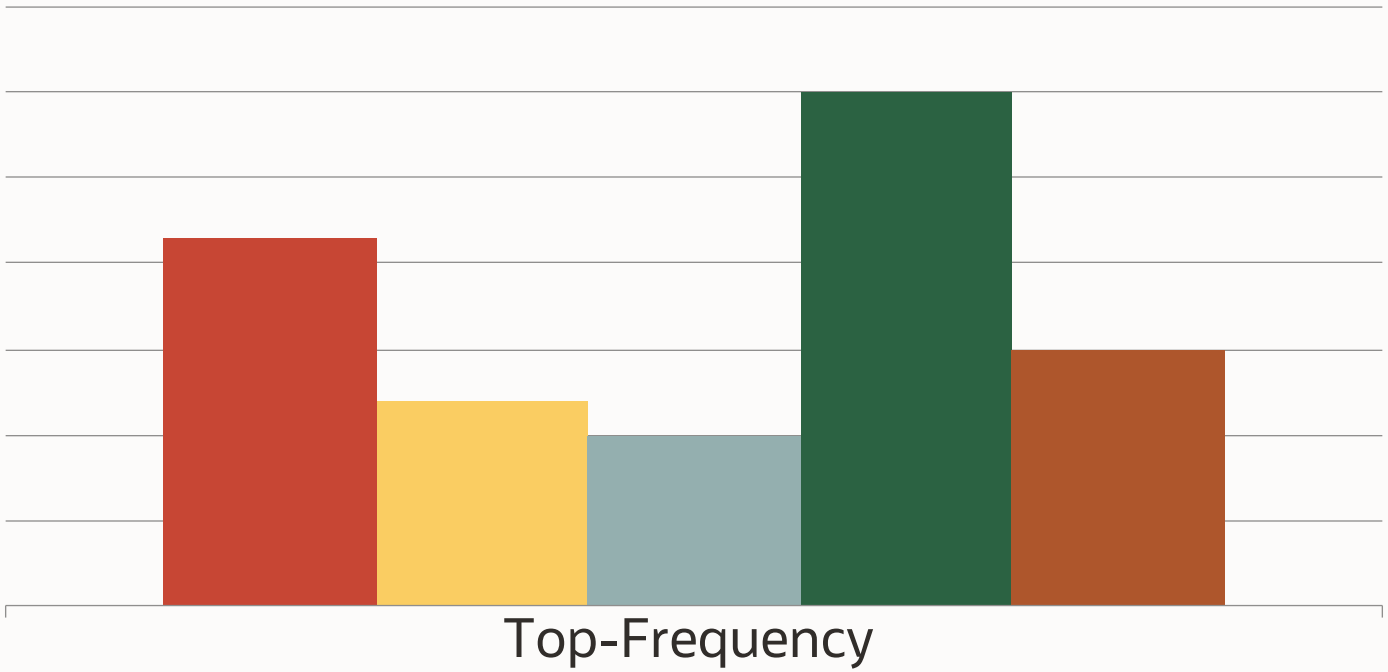
Number of buckets calculated by finding the bucket number for value and subtracting the previous bucket number (6-4 = 2)

Histograms

NEW IN 12.1

Top-Frequency Histograms - Starting in 12c

A Top-Frequency histogram created if less than 254 distinct values account for 99.9% of the entries in a column & `AUTO_SAMPLE_SIZE` is used



Histograms

Top Frequency (TOP-FREQUENCY)

- Traditionally a frequency histogram is only created if NDV < 254
- But if a small number of values occupies most of the rows (>99% rows)
- Creating a frequency histograms on that small set of values is very useful even though NDV is greater than 254
- Ignores the unpopular values to create a better-quality histogram for popular values
- Built using the same technique used for frequency histograms
- Only created with **`AUTO_SAMPLE_SIZE`**

Top Frequency Histogram Example

```
SQL>  Select  time_id, count(*)
  2  From     product_sales
  3  Group by time_id Order by 2;

TIME_ID     COUNT(*)
--------- ----------
06-JAN-98          1
28-JAN-98          1
06-FEB-98          1
07-FEB-98          1
08-FEB-98          1
16-FEB-98          1
10-APR-98          1
05-MAY-98          1
    .
    .
02-DEC-11        512
09-DEC-11        768
24-DEC-12       1024
21-DEC-11       2048
13-DEC-11       8192
21-DEC-12      10240
05-DEC-12      20480
22-DEC-12      32768
08-DEC-12      65536
17-DEC-11     327680
04-DEC-12    1310720

620 rows selected.
```

- Imagine we have store that sells Christmas decorations
- Majority of our sales take place from Nov – Jan
- But some small number of sales are made throughout the year

```
SQL> Select table_name, column_name, histogram
  2  From   user_tab_col_statistics
  3  Where  table_name='PRODUCT_SALES'
  4  And    column_name='TIME_ID';

TABLE_NAME           COLUMN_NAME                    HISTOGRAM
-------------------- ------------------------------ --------------
PRODUCT_SALES        TIME_ID                        TOP-FREQUENCY
```

`TIME_ID` column perfect candidate for top-frequency histogram

Histograms

Height balanced histograms (HEIGHT BALANCED)

- A height balanced histogram is created if the number of distinct values in a column (NDV) is greater than 254 values
- Goal is to put an even number of rows in each of the 254 buckets

Height balanced histogram

Creating a height-balance histogram

Step 1: SELECT cust_city_id FROM customers ORDER BY cust_city_id;

Row count	CUST_CITY_ID
1	**51040**
2	**51040**
:	:
219	**51043**
:	:
5256	**51166**
:	:
5475	**51166**
:	:
55500	**52531**

Creating a height-balance histogram

Step 2: Assign an equal number of rows per bucket

Row count	CUST_CITY_ID	
1	**51040**	
2	**51040**	
:	**:**	
219	**51043**	Bucket 1 has end point 51043
:	**:**	
5256	**51166**	Bucket 24 has end point 51166
:	**:**	
5475	**51166**	Bucket 25 has end point 51166
:	**:**	
55500	**52531**	Bucket 254 has end point 52531

Number of rows per bucket:
55,500 / 254 = 218.5

Creating a height-balance histogram

Step 3: If endpoint of 1st bucket is not min value add 0 bucket

Row count	CUST_CITY_ID	
1	**51040**	Bucket 0 has end point 51040
2	**51040**	
:	**:**	
219	**51043**	Bucket 1 has end point 51043
:	**:**	
5256	**51166**	Bucket 24 has end point 51166
:	**:**	
5475	**51166**	Bucket 25 has end point 51166
:	**:**	
55500	**52531**	Bucket 254 has end point 52531

But now we have 255 buckets instead of 254

How do we reduce it?

Creating a height-balance histogram

Step 4: Compress duplicate buckets

Row count	CUST_CITY_ID	
1	**51040**	Bucket 0 has end point 51040
2	**51040**	
:	**:**	
219	**51043**	Bucket 1 has end point 51043
:	**:**	
:	**:**	
5475	**51166**	Bucket 25 has end point 51166
:	**:**	
55500	**52531**	Bucket 254 has end point 52531

Monitoring histograms

Information on histograms found in USER_HISTOGRAMS

```
SQL> SELECT endpoint_number bucket_number, endpoint_value
  2  FROM   user_histograms
  3  WHERE  table_name='CUSTOMERS'
  4  AND    column_name='CUST_CITY_ID';

BUCKET_NUMBER ENDPOINT_VALUE
------------- --------------
            0          51040
            1          51043
            2          51044
            3          51046
            4          51049
            5          51053
            6          51055
            7          51057
            8          51059
            9          51061
           10          51062
           11          51067
           14          51069
           15          51073
           17          51075
            ⋮
          250          52520
          251          52526
          252          52527
          253          52529
          254          52531

212 rows selected.
```

Bucket 16 is missing because buckets 16 & 17 had the same endpoint value

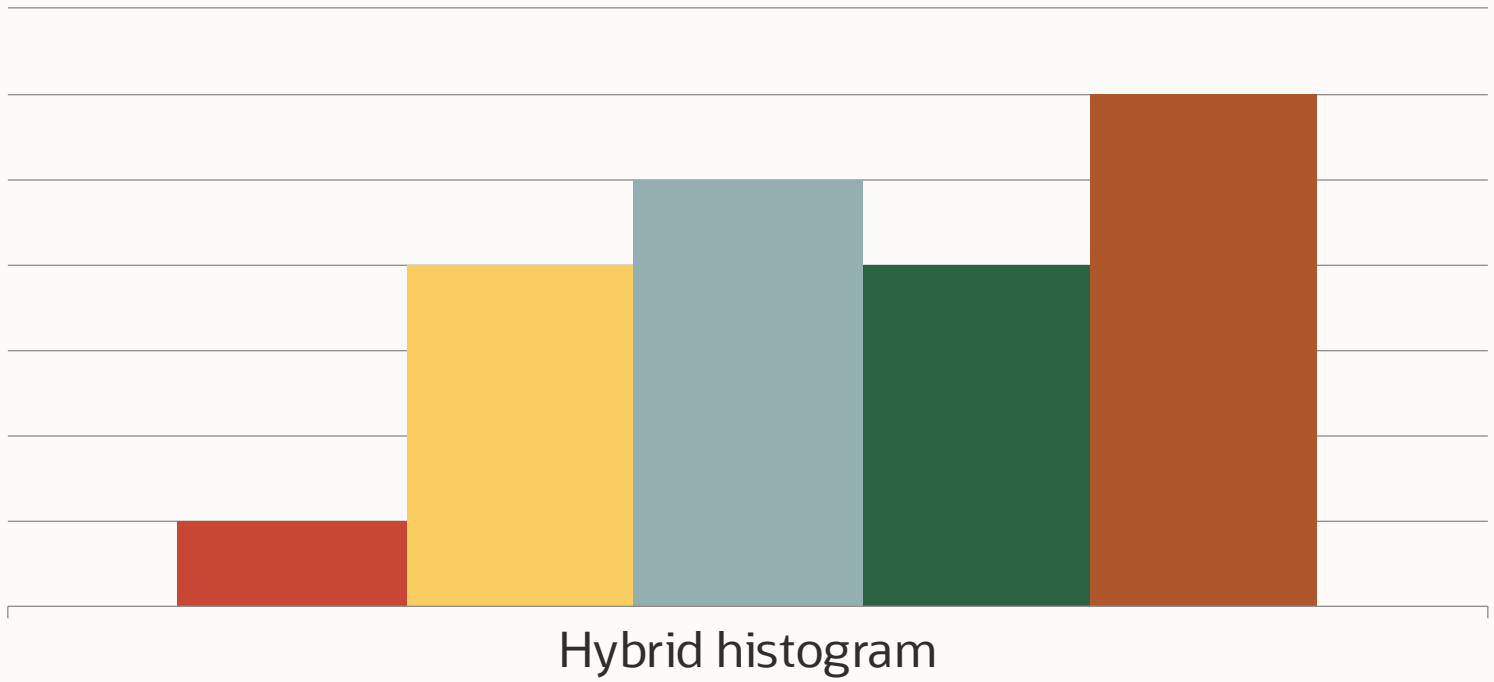
Not all 254 buckets used due to compression step

Histograms

Hybrid Histograms (HYBRID) - Starting in 12c

NEW IN 12.1

A hybrid balanced histogram is created if the number of distinct values in a column (NDV) is greater than 254 values & `AUTO_SAMPLE_SIZE` is used



Histograms

Hybrid Histograms (HYBRID) - Starting in 12c

- Like height balanced histogram hybrids are created if the NDV >254
- Store the actual frequencies of bucket endpoints in histograms
- No value allowed to spill over multiple buckets
- More endpoint values can be squeezed in a histogram
- Achieves the same effect as increasing the # of buckets
- Only created with `AUTO_SAMPLE_SIZE`

Hybrid Histogram Example

Step 1: SELECT row_num, time_id FROM sales ORDER BY 2;

ROWNUM	TIME_ID
1	02-JAN-98
2	03-JAN-98
3	05-JAN-98
4	05-JAN-98
5	06-JAN-98
6	09-JAN-98
7	09-JAN-98
8	09-JAN-98
9	10-JAN-98
10	10-JAN-98
:	:

There are 960 rows in the sales table
Automatically created histograms have 254 buckets
That is 3.77 rows per bucket

Hybrid Histogram Example

Step 2: Assign roughly an equal number of rows per bucket

ROWNUM	TIME_ID	BUCKET INFORMATION	FREQUENCY OF ENDPOINT
1	02-JAN-98	Bucket 1 has end point 02-JAN-98	1
2	03-JAN-98		
3	05-JAN-98		
4	05-JAN-98		
5	06-JAN-98	Bucket 5 has end point 06-JAN-98	1
6	09-JAN-98		
7	09-JAN-98		
8	09-JAN-98		
9	10-JAN-98		
10	10-JAN-98	Bucket 10 has end point 10-JAN-98	2
:	:		

Hybrid Histogram Example

Monitoring

- No two buckets have the same endpoint
- Frequency of endpoint values recorded in new column called `Endpoint Repeat Count`

```
SQL> Select endpoint_number, endpoint_value, endpoint_repeat_count
  2  From   user_tab_histograms
  3  Where  table_name='SALES'
  4  And    column_name='TIME_ID'
  5  Order by endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE ENDPOINT_REPEAT_COUNT
--------------- -------------- ---------------------
              1        2450816                     1
              5        2450820                     1
             10        2450824                     2
             14        2450830                     1
             17        2450837                     1
             21        2450846                     2
             25        2450850                     1
             30        2450854                     2
             33        2450859                     2
                           :
            956        2451874                     1
            960        2451878                     3

254 rows selected.
```

Number of occurrences of the endpoint value in the sample

How the Optimizer uses Height-balanced & Hybrid histograms

Optimizer used two different formulas depend on the popularity of the value

- Popular values in height-balance are the endpoint for two or more buckets
- Popular values in hybrid are the endpoint of a bucket
- Formulas used are:

Height-balanced: $\frac{\text{Number of endpoint buckets}}{\text{Total number of buckets}} \times \text{\#rows in table}$

Hybrid: $\frac{\text{Endpoint Frequency}}{\text{Max buckets}} \times \text{\#rows in table}$

How the Optimizer uses Height-balanced & Hybrid histograms

Optimizer used two different formulas depend on the popularity of the value

- Non-popular value means values that are the endpoint for only one bucket on a height-balance histogram or are not an endpoint at all
- Formula used is:

DENSITY X number of rows in the table

NOTE: Density from 10.2.0.4 is calculated on the fly based on histogram information and is not the value show in `USER_HISTOGRAMS`

How the Optimizer uses histograms

Popular values use Height-Balance histogram information

```
SELECT Count(*)
FROM   sales
WHERE  cust_id = 22300;
```

```
COUNT(*)
--------------
       5
```

Cardinality estimate:

$$\frac{\text{Number of endpoint buckets}}{\text{total number of buckets}} \times \text{number of rows in table}$$

$$\frac{2}{249} \times 960 = 7.71 = 8$$

```
-----------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |       |       |     5 (100)|          |
|   1 |  SORT AGGREGATE      |       |     1 |     5 |            |          |
|   2 |   PARTITION RANGE ALL|       |     8 |    40 |     5   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL | SALES |     8 |    40 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

How the Optimizer uses histograms

Popular values use Hybrid histogram information

```
SELECT Count(*)
FROM   sales
WHERE  cust_id = 22810;
```

```
COUNT(*)
-------------
        5
```

Cardinality estimate:

$$\frac{\text{Endpoint Frequency}}{\text{Max buckets}} \times \text{number of rows in the table}$$

$$\frac{5}{960} \times 960 = 5$$

```
-----------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |       |       |     5 (100)|          |
|   1 |  SORT AGGREGATE      |       |     1 |     5 |            |          |
|   2 |   PARTITION RANGE ALL|       |     5 |    25 |     5   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL | SALES |     5 |    25 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

How the Optimizer uses histograms

Non-popular values use density

```
SELECT Count(*)
FROM   sales
WHERE  cust_id = 200;
```

```
COUNT(CUST_ID)
--------------
            2
```

Cardinality estimate:

DENSITY X number of rows

0.00104 X 960

```
-----------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |       |       |     5 (100)|          |
|   1 |  SORT AGGREGATE      |       |     1 |     5 |            |          |
|   2 |   PARTITION RANGE ALL|       |     1 |     5 |     5   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL | SALES |     1 |     5 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

How to gather statistics

Why people hate histograms

Histograms have a bad reputation

WHY?

They can cause instability
They can increase the time to gather statistics

How to gather statistics

Why people hate histograms

Two main cause of instability with Histograms

1. Bind peeking interacts with histograms
2. Nearly popular values

Adaptive Cursor Sharing introduced in 11gR2 helps relieve a lot of the problems caused by bind peeking and histograms

Solutions for bind peeking and histograms

Adaptive Cursor Sharing

- Share the plan when binds values are “equivalent”
  - Plans are marked with selectivity range
  - If current bind values fall within range, they use the same plan
- Create a new plan if binds are not equivalent
  - Generating a new plan with a different selectivity range
- Controlled by init.ora parameter *`_optim_peek_user_binds`*
- Monitoring - V$SQL has 2 new columns
  - `IS_BIND_SENSITIVE` - Optimizer believes the plan may depend on the value of bind
  - `IS_BIND_AWARE` - Multiple execution plans exist for this statement
  - Can force bind awareness via the BIND_AWARE hint

Solutions for bind peeking and histograms

Prior to 11gR2

Applications that only have statements with binds

- Drop histogram using `DBMS_STATS.DELETE_COL_STATS`
- Use `DBMS_STATS.SET_PARM` to change default setting for method_opt parameter to prevent histogram from being created
- Re-gather statistics on the table without histogram

Applications that have statements with bind and literals

- Switch off bind peeking *_optim_peek_user_binds = false*

How to gather statistics

Why people hate histograms

Two main cause of instability with Histograms

1. Bind peeking interacts with histograms
2. Nearly popular values

Nearly popular values are classified as non-popular values that means their cardinality estimates is based on a density calculation, which is not always accurate enough for them

- To get an accurate cardinality estimate use dynamic sampling
- OR …

Nearly popular values

Cardinality estimates is based on a density calculation

Nearly popular value means the value is classified as non-popular but the density calculation is not accurate for them

```
SQL> SELECT count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =52114;

COUNT(CUST_ID)
--------------
           227
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE              |           |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |    66 |   330 |   405   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

Same estimate used as for non-popular. Here density is not good enough to get accurate cardinality estimate

Nearly popular values

Solution 1 dynamic sampling

To get an accurate cardinality estimate for nearly popular values use dynamic sampling

```
SQL> SELECT /*+ dynamic_sampling(customers 2) */ count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =52114;

COUNT(CUST_ID)
--------------
           227

-------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE            |           |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |   248 |  1240 |   405   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Nearly popular values

Solution 2 Drop the Histogram

If hints aren't possible and stability is a priority drop the histogram

```
BEGIN
    dbms_stats.Delete_column_stats(ownname=>'SH', tabname=>'CUSTOMERS', -
    colname=>'CUST_CITY_ID', col_stat_type=>'HISTOGRAM');
END;
/

BEGIN
    dbms_stats.Set_table_prefs('SH', 'CUSTOMERS', 'METHOD_OPT', -
    'FOR ALL COLUMNS SIZE AUTO, FOR COLUMNS SIZE 1 CUST_CITY_ID');
END;
/
```

AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

Additional statistics

When table and column statistics are not enough

Two types of Extended Statistics

- Column groups statistics
  - Useful when multiple column from the same table are used in where clause predicates
- Expression statistics
  - Useful when a column is used as part of a complex expression in where clause predicate

Can be manually or automatically created

Automatically maintained when statistics are gathered on the table

Extended Statistics – column group statistics

```
SELECT *
FROM   vehicles
WHERE  model = '530xi'
AND    color = 'RED';

MAKE    MODEL  COLOR
------ ------ ------
 BMW    530xi  RED
```

Cardinality = #ROWS * $\frac{1}{\text{NDV c1}}$ * $\frac{1}{\text{NDV c2}}$ = 12 * $\frac{1}{4}$ * $\frac{1}{3}$ = 1

```
-----------------------------------------------------------------------
| Id  | Operation         | Name     | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |      1 |        |      1 |
|*  1 |  TABLE ACCESS FULL| VEHICLES |      1 |      1 |      1 |
-----------------------------------------------------------------------
```

Vehicles Table (12 ROWS)		
MAKE	MODEL	COLOR
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
TESLA	P85D	GRAY
:	:	:
PORSCHE	911	RED

Extended Statistics – column group statistics

```
SELECT *
FROM   vehicles
WHERE  model = '530xi'
AND    make = 'BMW';

MAKE    MODEL  COLOR
------ ------ ------
 BMW    530xi  RED
```

Cardinality $= \#ROWS * \frac{1}{NDV\ c1} * \frac{1}{NDV\ c2} = 12 * \frac{1}{4} * \frac{1}{3} = 1$

```
-----------------------------------------------------------------------
| Id  | Operation         | Name     | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |      1 |        |      3 |
|*  1 |  TABLE ACCESS FULL| VEHICLES |      1 |      1 |      3 |
-----------------------------------------------------------------------
```

Vehicles Table (12 ROWS)		
MAKE	MODEL	COLOR
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
TESLA	P85D	GRAY
:	:	:
PORSCHE	911	RED

Extended Statistics – column group statistics

Create extended statistics on the `MODEL & MAKE` columns using the `CREATE_EXTENDED_STATS` function in the `DBMS_STATS` package

```
SQL> SELECT    dbms_stats.create_extended_stats(Null, 'VEHICLES', '(MODEL,MAKE)')
  2  FROM      dual;

DBMS_STATS.CREATE_EXTENDED_STATS(NULL,'VEHICLES','(MODEL,MAKE)')
----------------------------------------------------------------------------------
SYS_STUJK04CGHOMR70#X#4QHNIFAZ

SQL>
SQL> BEGIN
  2  dbms_stats.gather_table_stats( Null,  'VEHICLES');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT column_name, num_distinct, histogram
  2  FROM   user_tab_col_statistics
  3  WHERE  table_name='VEHICLES';

COLUMN_NAME                    NUM_DISTINCT HISTOGRAM
------------------------------ ------------ ---------------
MAKE                                      3 NONE
MODEL                                     4 NONE
COLOR                                     5 NONE
SYS_STUJK04CGHOMR70#X#4QHNIFAZ            4 NONE
```

New Column with system generated name

Extended Statistics – column group statistics

```
SELECT *
FROM   vehicles
WHERE  model = '530xi'
AND    make = 'BMW';

MAKE    MODEL  COLOR
------ ------ ------
 BMW    530xi  RED
```

Cardinality calculated using column group statistics

```
-----------------------------------------------------------------------
| Id  | Operation         | Name     | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |      1 |        |      3 |
|*  1 |  TABLE ACCESS FULL| VEHICLES |      1 |      3 |      3 |
-----------------------------------------------------------------------
```

Vehicles Table (12 ROWS)		
MAKE	MODEL	COLOR
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
TESLA	P85D	GRAY
:	:	:
PORSCHE	911	RED

Extended Statistics – expression statistics solution

```
SELECT *
FROM   Customers
WHERE  UPPER(CUST_NAME) = 'SMITH';
```

- Optimizer doesn't know how function affects values in the column
- Optimizer guesses the cardinality to be 1% of rows

Customers Table (55,500 ROWS)		
CUST_ID	NAME	EMAIL
00001	Smith	Smith@...
00002	Jones	Jones@....
00003	Colgan	Colgan@.. .
00004	Lewis	Lewis@...
:	:	:
55500	Baer	Baer@...

```
-----------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      1 |
|   1 |  SORT AGGREGATE              |           |      1 |      1 |      1 |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |      1 |    555 |     79 |
-----------------------------------------------------------------------------
```

Cardinality estimate is 1% of the rows

Extended Statistics – expression statistics solution

```
SQL> BEGIN
  2  dbms_stats.gather_table_stats(null,'customers',method_opt =>'for all columns size skewonly for columns (UPPER(CUST_LAST_NAME))');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT column_name, num_distinct, histogram
  2  FROM   user_tab_col_statistics
  3  WHERE  table_name = 'CUSTOMERS';

COLUMN_NAME                    NUM_DISTINCT HISTOGRAM
------------------------------ ------------ ---------------
SYS_STUSKCCJE8MV8IIBWT5PA5A41V          908 HEIGHT BALANCED
CUST_ID                               55500 HEIGHT BALANCED
CUST_FIRST_NAME                        1300 HEIGHT BALANCED
CUST_LAST_NAME                          908 HEIGHT BALANCED
CUST_GENDER                               2 FREQUENCY
CUST_YEAR_OF_BIRTH                       75 FREQUENCY
CUST_MARITAL_STATUS                      11 FREQUENCY
CUST_STREET_ADDRESS                   49900 HEIGHT BALANCED
CUST_POSTAL_CODE                        623 HEIGHT BALANCED
```

New Column with system generated name

Extended Statistics

Automatic column group detection

1. Start column group usage capture

```
BEGIN
    dbms_stats.Seed_col_usage(NULL, NULL, 300);
END;
/
```

Switches on monitoring for 300 seconds or the next 5 minutes. Any statement executed will be monitored for columns used in the where and group by clauses

Extended Statistics

Automatic Creation via SQL Plan Directives in 12c

Directives are additional information used during optimization to generate a plan

- For example, when table T1 is joined to T2 use dynamic statistics to get accurate cardinality estimate
- Directives are collected on query expressions not at a statement level
  - Allows for directives to be used for multiple statements
- Persisted on disk in the `SYSAUX` tablespace
- Managed using the new package `DBMS_SPD`
- Column group statistics can be automatically created via directives

Extended Statistics

Automatic Creation via SQL Plan Directives in 12c and Beyond

By default in 12.1 directives were always created & used

- Use of directive and the automatic creation of column groups is controlled by the parameter `OPTIMIZER_ADAPTIVE_FEATURES` (default value is `TRUE`)

From 12.2 onward directives are always created but not always used

- Use of SQL Plan Directives by the Optimizer is controlled by the parameter `OPTIMIZER_ADAPTIVE_STATISTICS` (default value is `FALSE`)
- To have column group statistics automatically created based on directives set the `DBMS_STATS` preference `AUTO_STAT_EXTENSIONS` to `ON` (default value is `OFF`)

AGENDA

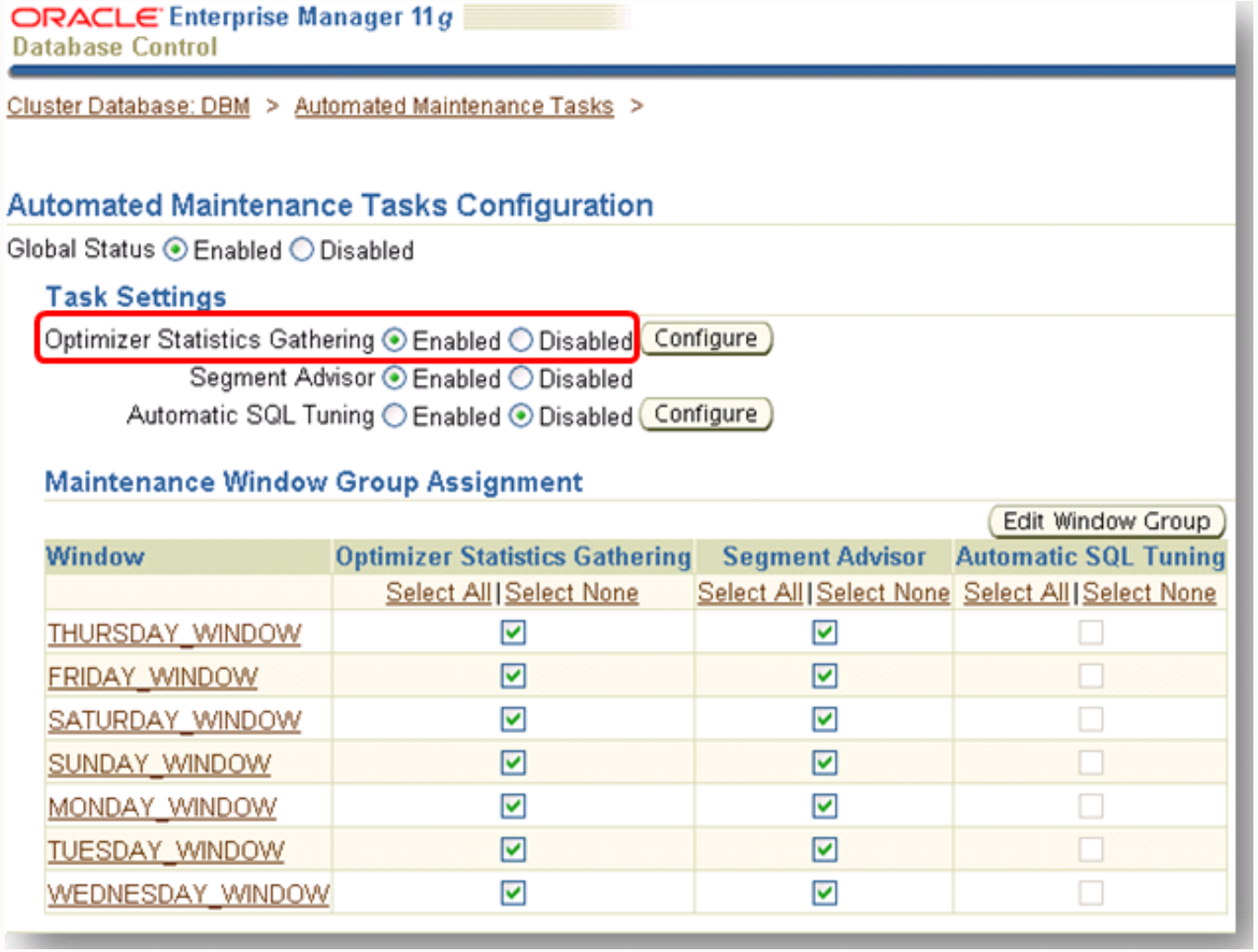
1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

When to gather statistics

Automatic Statistics Gathering

- Oracle automatically collect statistics for all database objects, which are missing statistics or have stale statistics
- AutoTask run during a predefined maintenance window
- Internally prioritizes the database objects
  - Both user schema and dictionary tables and fixed objects from 12c onwards
  - Objects that need updated statistics most are processed first
- Controlled by `DBMS_AUTO_TASK_ADMIN` package or via Enterprise Manager

Automatic Statistics Gathering



Automatic Statistics Gathering

If you want to disable auto job for application schema leaving it on for Oracle dictionary tables

The scope of the auto job is controlled by the global preference `AUTOSTATS_TARGET`

Possible values are

- AUTO Oracle decides what tables need statistics (Default)
- All Statistics gathered for all tables in the system
- ORACLE Statistics gathered for only the dictionary tables

Let Oracle take care of itself

- Remember use `ENABLE_AUTOMATIC_MAINTENANCE_PDB` to disable the maintenance window at the PDB level

Online Statistics Gathering

NEW IN 12.2

- Starting in 12c statistics gathered as part of the direct path load operations
  - Create Table As Select or Insert As Select commands
- Most essential statistics available directly after load
- No additional table scan required to gather statistics
- All internal maintenance operations that use CTAS benefit from this
- Note only occurs on IAS if table is empty
- Remaining statistics gathering is deferred until Auto Stats Task

Note: Histograms are not created as they require an additional scan of the data
Use `_optimizer_gather_stats_on_load_hist` to enable histogram collection

Real-Time Statistics

NEW IN 19c

- Gather statistics as part of conventional DML (Insert/update/merge)
- Statistics gathered during DML needs to be fast with negligible overhead
- Only most essential stats are gathered to avoid catastrophic SQL execution plan performance regressions (e.g. avoiding out-of-range conditions)
  - Min, Max, num_rows, etc.
- Gathering of remaining statistics is deferred
- Automatic statistics gathering job still kicks in as needed

When to gather statistics

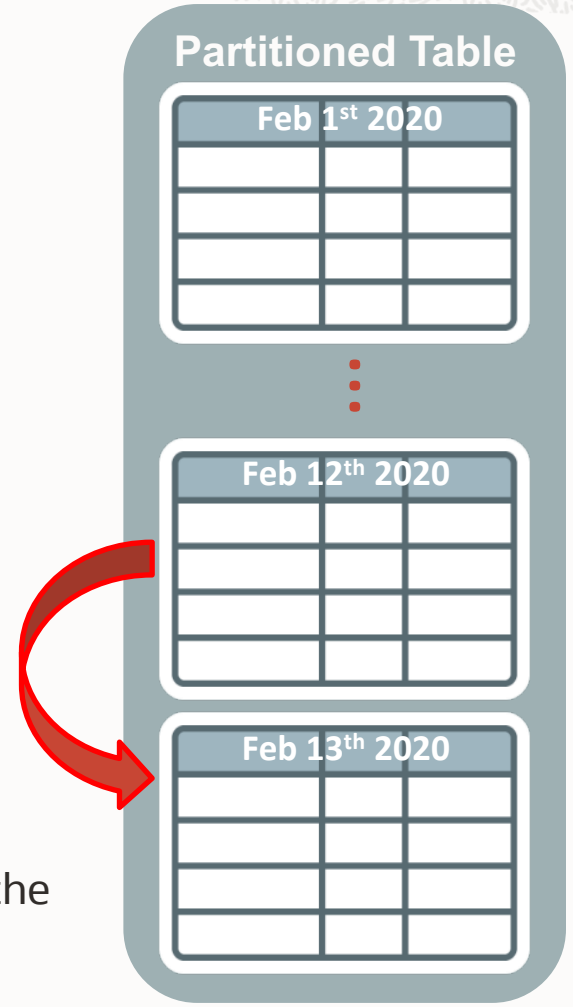
If the Auto Statistics Gather Job is not suitable

After a large data load

- Include gathering statistics as part of the ETL or ELT process

If trickle loading into a partition table

- Used `dbms_stats.Copy_table_stats()`
  - Copies stats from source partition
  - Adjust min & max values for partition column
    - Both partition & global statistics*
  - Copies statistics of the dependent objects
    - Columns, local (partitioned) indexes* etc.



Note*: By default global statistics are not adjusted. You must set the `FLAGS` parameter of the `DBMS_STATS.COPY_TABLE_STATS` to 8 for global stats to be adjusted.
Global indexes statistics are never adjusted

AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

Statistics gathering performance

How to speed up statistics gathering

Four options to speed up statistics gathering

- Intra object using parallel execution
- Inter object using concurrency
- The combination of Inter and Intra object
- Incremental statistics gathering for partitioned tables

Statistics gathering performance

Intra object using parallel execution

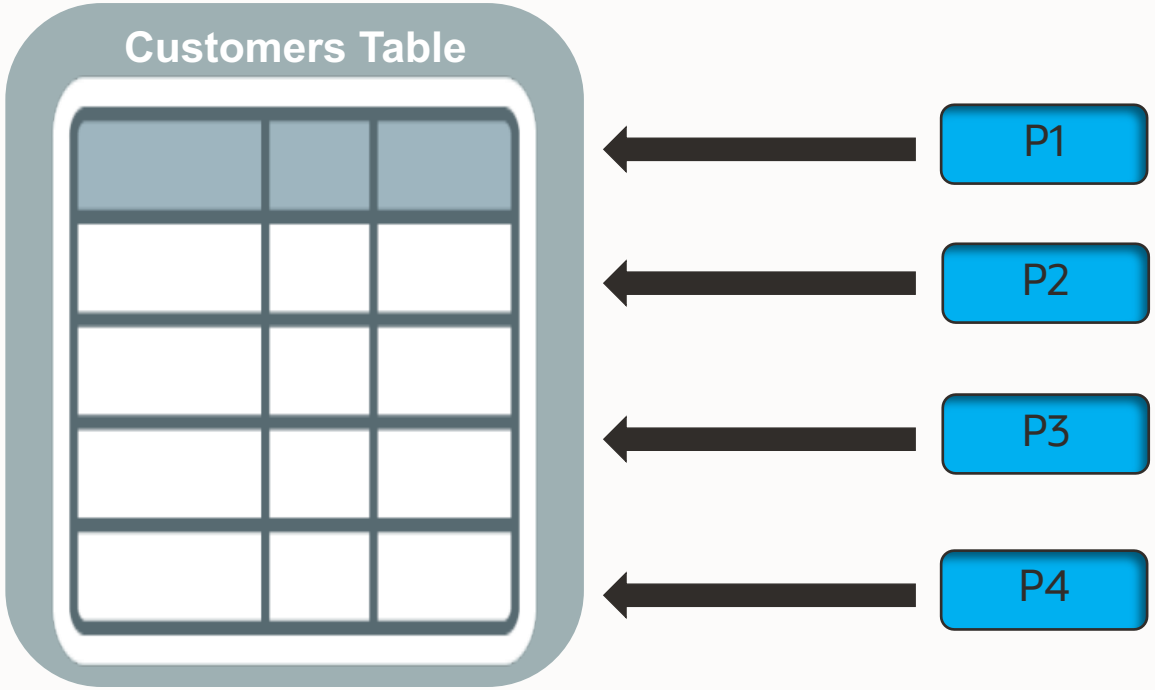
- Controlled by `GATHER_*_STATS` parameter `DEGREE`
- Default is to use parallel degree specified on object
- If set to `AUTO` Oracle decide parallel degree used
- Works on one object at a time

Statistics gathering performance

Intra object using parallel execution

Customers table has a degree of parallelism of 4

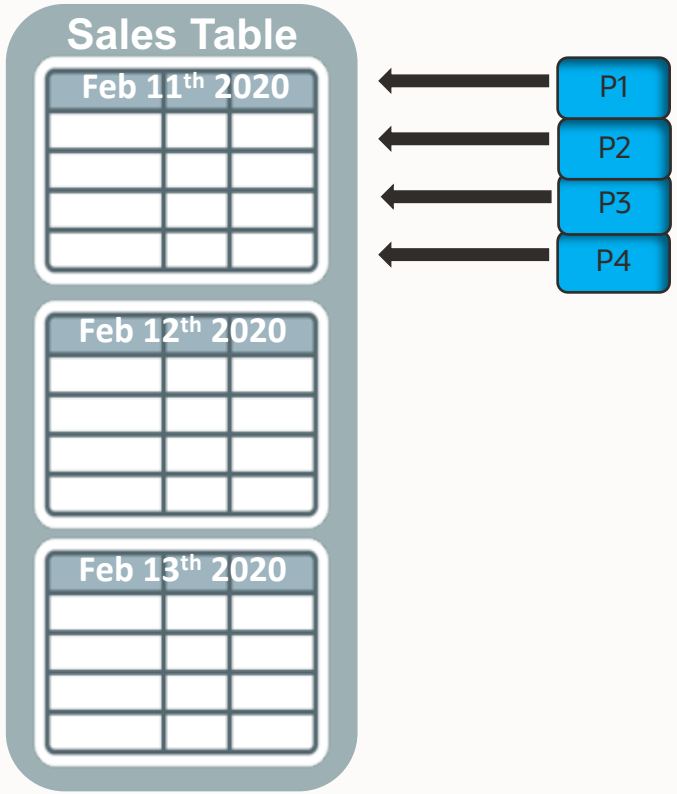
4 parallel server processes will be used to gather stats



Statistics gathering performance

Intra object using parallel execution

```
dbms_stats.Gather_table_stats('SH', 'SALES');
```



Each individual partition will have statistics gathered one after the other

The statistics gather procedure on each individual partition operates in parallel BUT the statistics gathering procedures won't happen concurrently

Statistics gathering performance

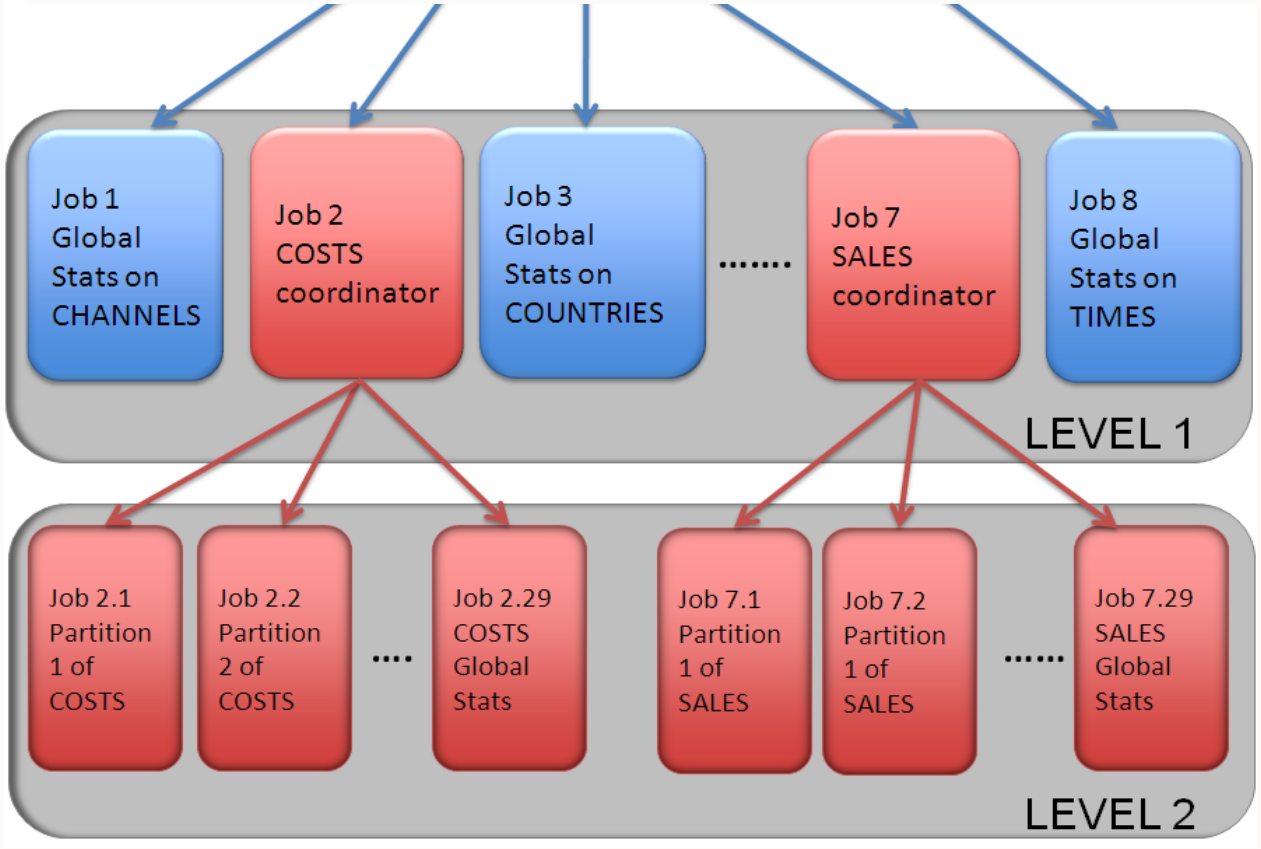
Inter object

- Gather statistics on multiple objects at the same time
- Controlled by `DBMS_STATS` preference, `CONCURRENT`
- Uses Database Scheduler and Advanced Queuing
- Number of concurrent gather operations controlled by `JOB_QUEUE_PROCESSES` parameter
- Each gather operation can still operate in parallel

Statistics gathering performance

Intra object statistics gathering for SH schema

```
dbms_stats.Gather_schema_stats('SH');
```



A separate statistics gathering job is created for each table and each partition in the schema

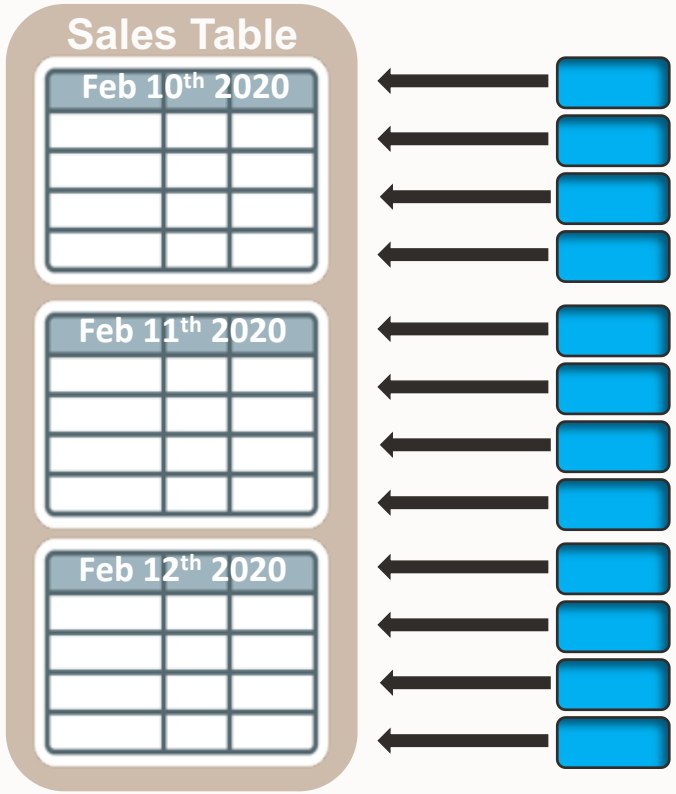
Level 1 contain statistics gathering jobs for all non-partitioned tables and a coordinating job for each partitioned table

Level 2 contain statistics gathering jobs for each partition in the partitioned tables

Statistics gathering performance

Inter and intra working together for partitioned objects

```
dbms_stats.Gather_schema_stats('SH');
```



The number of concurrent gathers is controlled by the parameter `JOB_QUEUE_PROCESSES`

In this example it is set to 3

Remember each concurrent gather operates in parallel

In this example the parallel degree is 4

Statistics gathering performance

Incremental statistics gathering for partitioned tables

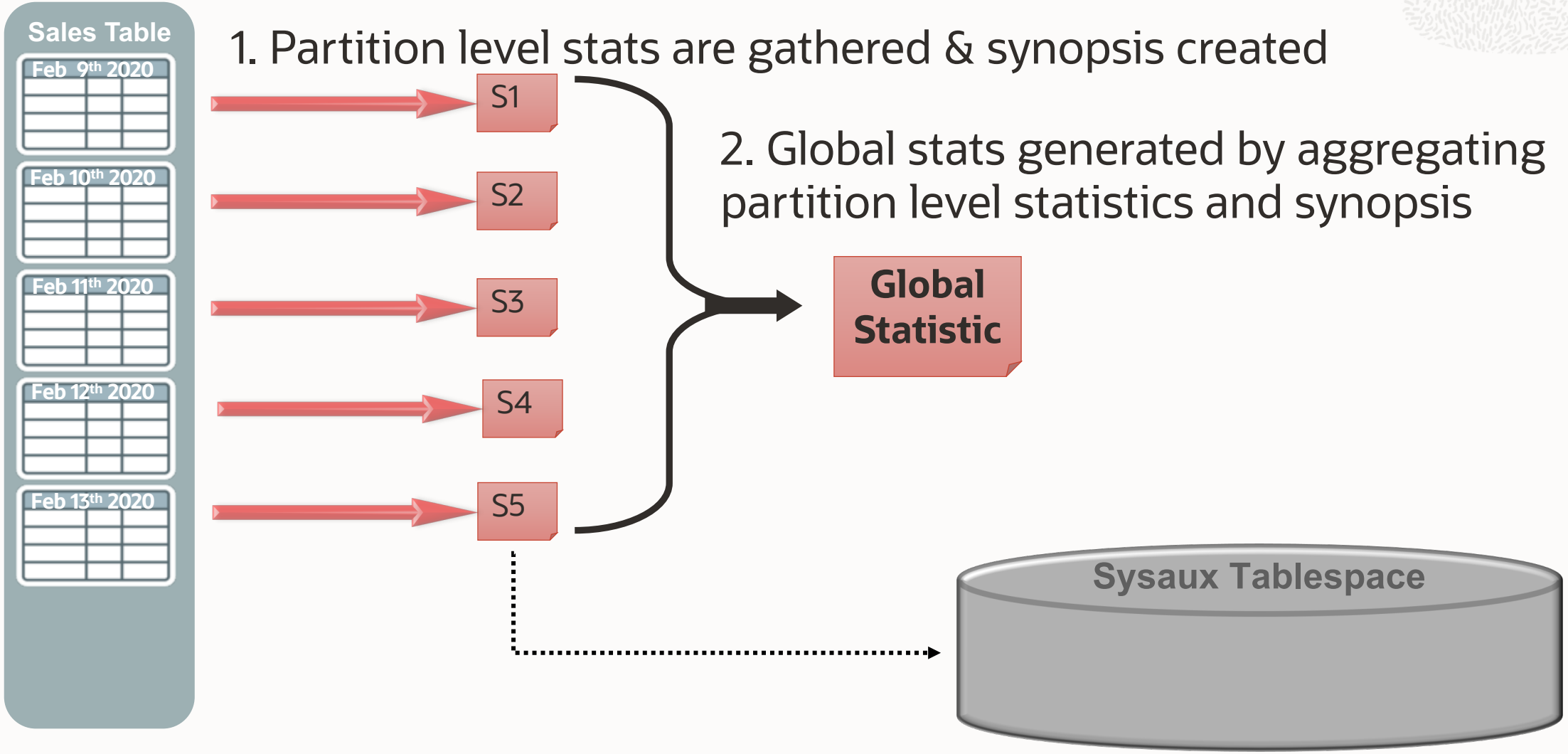
Typically gathering statistics after a bulk load into one partition would causes a full scan of all partitions to gather global table statistics

- Extremely time consuming

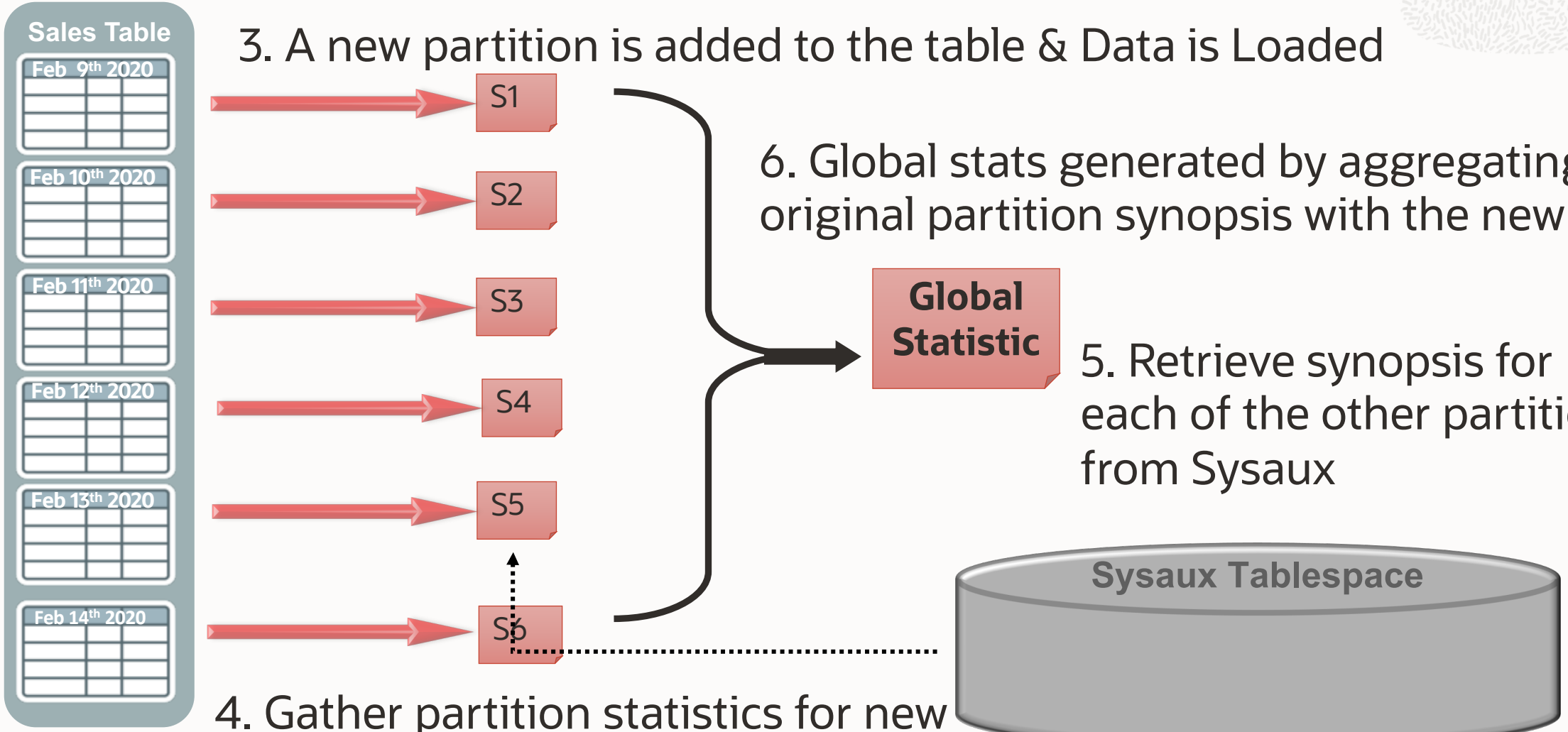
With Incremental Statistic gather statistics for touched partition(s) ONLY

- Table (global) statistics are accurately built from partition statistics
- Reduce statistics gathering time considerably
- Controlled by INCREMENTAL preference

Incremental Statistics Gathering



Incremental Statistics Gathering



Incremental Statistics Gathering

Why some people hate incremental statistics

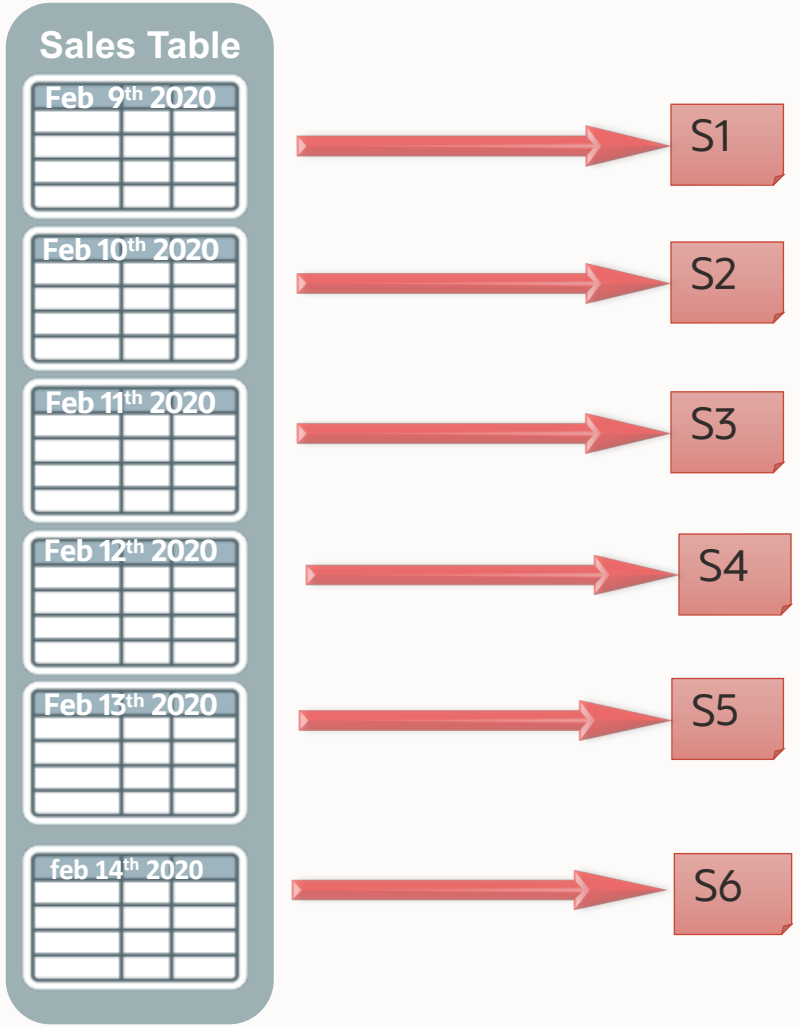
Incremental statistics have a bad reputation

WHY?

Originally, they were bloated and slow

Incremental Statistics Gathering

How expensive is the creation of a synopsis?



Synopsis can now be created in two ways

- Adaptive Sampling
  - Database uses an adaptive sampling algorithm which provide very accurate results, but the resulting synopsis can be large
- Hyperloglog
  - Database uses the hyperloglog algorithm which offers similar accuracy of adaptive sampling but requires significantly less memory as the synopsis size is greatly reduce
- It is possible to have both types on one table

Control by the new `DBMS_STATS` preference `APPROXIMATE_NDV_ALGORITHM`

Incremental Statistics Gathering

How expensive is the creation Adaptive Statistics synopsis?

- Number of rows is a factor of the # of columns, partitions & NDV per column

```
SQL> select count(*) from sys.WRI$_OPTSTAT_SYNOPSIS$
  2  where bo# = (select object_id from user_objects where object_name = 'T1' and object_type = 'TABLE');

  COUNT(*)
----------
   4887844
```

- One entry for each column, for each partition

```
SQL> select count(*) from sys.WRI$_OPTSTAT_SYNOPSIS_HEAD$
  2  where bo# = (select object_id from user_objects where object_name = 'T1' and object_type = 'TABLE');

  COUNT(*)
----------
      2560
```

Incremental Statistics Gathering

How expensive is the creation Hyperloglog synopsis?

- No rows are store in the WRI$_OPTSTAT_SYNOPSIS$

```
SQL> select count(*) from sys.WRI$_OPTSTAT_SYNOPSIS$
  2  where bo# = (select object_id from user_objects where object_name = 'T1' and object_type = 'TABLE');

  COUNT(*)
----------
         0
```

- One entry for each column, for each partition

```
SQL> select count(*) from sys.WRI$_OPTSTAT_SYNOPSIS_HEAD$
  2  where bo# = (select object_id from user_objects where object_name = 'T1' and object_type = 'TABLE');

  COUNT(*)
----------
      2560
```

Incremental Statistics Gathering

How expensive is the creation Hyperloglog synopsis?

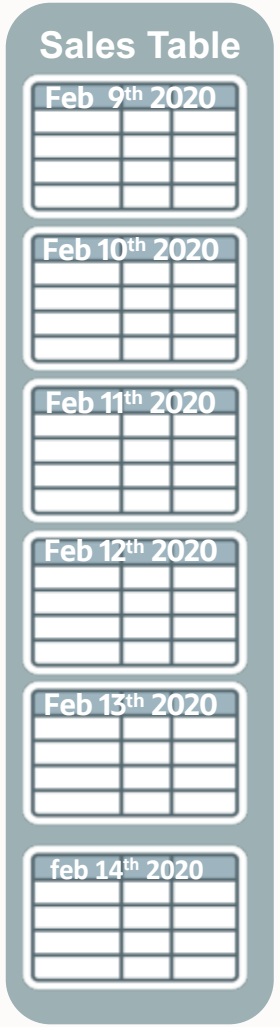
```
SQL> select spare1,spare2 from sys.WRI$_OPTSTAT_SYNOPSIS_HEAD$
  2  where  bo# = (select object_id from user_objects where object_name = 'T1' and object_type = 'TABLE')
  3  and rownum<11;

    SPARE1 SPARE2
---------- ------------------------------------------------------------------------------------------
         1 0D0C00000FCB0000000000000440A0431010460820010C31850841C10C40C21410C20820820C2088084082144100083 0
         1 0D0C08E9000100000000000000010000000000000000000000000000000000000000000000000000000000000000000
         1 0D0C0006007C000000000000007D0000000000C000000001000000000000000000800020000C000000000000000000
         1 0D0C001C0246000000000000027300000000000000000000000000000000000000000400000000000000000000000
         1 0D0C0586000500000000000000050000000000000000000000000000000000000000000000000000000000000000000
         1 0D0C00000FC900000000000000463B0C2081102084043101 0C41010851040820C3083103185142142083103102042043 1
         1 0D0C00E9000100000000000000010000000000000000000000000000000000000000000000000000000000000000000
         1 0D0C0012007B000000000000007C000000000000000000000000000800000000000000000000000000000000400000
         1 0D0C0005024C000000000000027A000000001000000000000400000000000000001000000000C000300000000800000
         1 0D0C0586000500000000000000050000000000000000000000000000000000000000000000000000000000000000000
```

Only binary representation of NDV stored for each column in each partition

Incremental Statistics Gathering

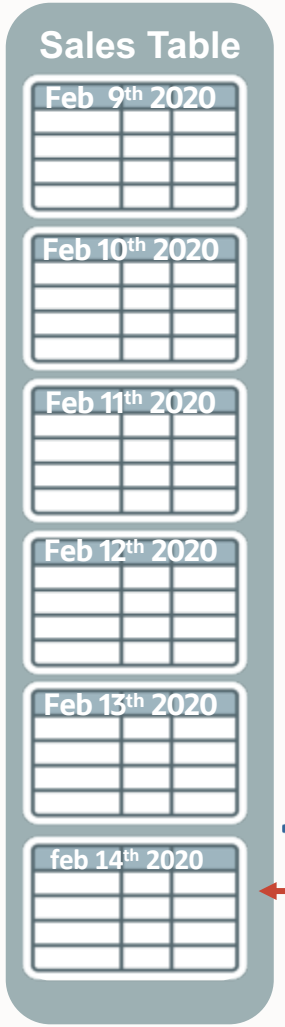
What happens if data changes in one of the existing partitions?



- During data load some rows often going into the older partitions
- In 11g any DML on older partitions triggered partition statistics to be re-gathered
- New `DBMS_STATS` preference `INCREMENTAL_STALENESS`
- When set to `USE_STALE_PERCENT`, DML on less than 10% of rows in older partitions will not trigger re-gather
- When set to `USE_LOCKED_STATS`, locked partitions stats are never considered stale, regardless of DML changes
- When set to `ALLOW_MIXED_FORMAT`, adaptive statistics & hyperloglog synopsis can exist on the same table

Incremental Statistics Gathering

What happens if you do a partition exchange load?



1. Create external table for flat files

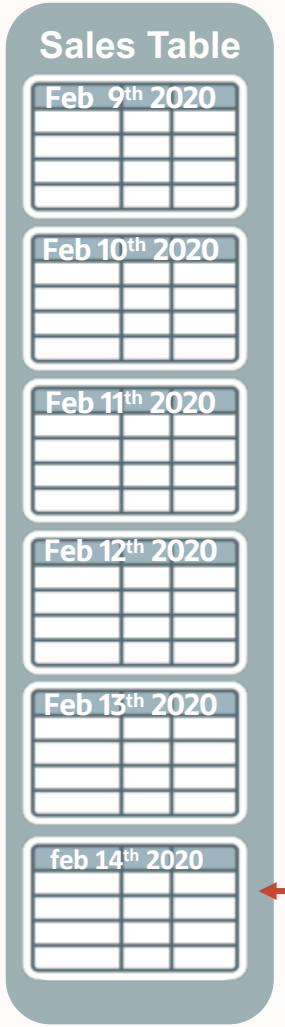
2. Use CTAS command to create non-partitioned table

3. Alter table Sales exchange partition Sept_14_2020 with table tmp_sales

4. Gather Statistics

Incremental Statistics Gathering

What happens if you do a partition exchange load?



1. Create external table for flat files
2. Use CTAS command to create non-partitioned table
3. Set `INCREMENTAL` to true & `INCREMENTAL_LEVEL` to `TABLE`
4. Gather Statistics
5. Alter table Sales exchange partition Sept_14_2020 with table tmp_sales

AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

When not to gather statistics

Volatile Tables

- Volume of data changes dramatically over a period of time
- For example orders queue table
  - Starts empty, orders come in, order get processed, ends day empty

Global Temp Tables*

- Application code stores intermediate result
- Some session have a lot of data, some have very little

Intermediate work tables

- Written once, read once and then truncated or deleted
- For example part of an ETL process

* Not the case in Oracle Database 12c

When not to gather statistics

Volatile tables

- Data volume changes dramatically over time
- When is a good time to gather statistics?

- Gather statistics when the table has a representative data volume
- Lock statistics to ensure statistics gathering job does not over write representative statistics

When not to gather statistics

Intermediate work tables

Often seen as part of an ETL process

Written once, read once, and then truncated or deleted

When do you gather statistics?

Don’t gather statistics it will only increase ETL time

Use Dynamic sampling

- Add dynamic sampling hint to SQL statements querying the intermediate table

When not to gather statistics

Intermediate work tables

Add dynamic sampling hint or set it at the session or system level

```
SELECT /*+ dynamic_sampling(c 2) */ c.*
FROM   customers_staging_tab c
WHERE  cust_address_change = 'Y';
```

Session private statistics for GTT

Overview

- Traditionally statistics gathered on GTT were shared by all sessions
- Share statistics are not always optimal
- Now each session can have its own version of statistics for GTT
- Controlled by new preference `GLOBAL_TEMP_TABLE_STATS`
- Default value is `SESSION` (non shared)
- To force sharing (as in 11g) set table preference to `SHARED`

AGENDA

1. How to Gather Statistics
2. What Basic Statistics to Gather
3. Additional Statistics
4. When to Gather Statistics
5. Statistics Gathering Performance
6. When Not to Gather
7. Other Types of Statistics

Other types of statistics

- Dictionary Statistics
- Fixed Object Statistics
- System Statistics
- Dynamic Sampling
- Cloning databases

Dictionary statistics

- Statistics on dictionary tables
- Maintained as part of the automatic statistics job
  - Set preference `AUTOSTATS_TARGET` to `ORACLE` to only gather Dictionary stats
- Can be manually gather using
  - `Dbms_stats.Gather_dictionary_stats`
  - Set options to `AUTO`

Fixed object statistics

- Fixed objects are the x$ tables and their indexes
  - v$ performance views built on top of them
- NOT maintained as part of the automatic statistics task in 11g but in 12c*
- Dynamic sampling not used on X$ tables when stats are missing or stale
- Can be manually gathering using
  - `dbms_stats.Gather_fixed_object_stats`
- X$ tables are transient so gather when representative workload is active
- May need to be re-gathered after converting to a PDB from non-PDB

* In 12c fixed object statistics are gathered as part of auto statistics gathering if they have not been previously collected!

System statistics

- Information about hardware database is running on
- NOT maintained as part of the automatic statistics job
- Manually gathered using
  - `dbms_stats.Gather_syste`
- Can be gathered with or without a
- Default setting are recommended stems
- For Exadata systems new "EXA no able

Not recommend to mess with system statistics – defaults are generally best

SQL Maria®

Join the Conversation

https://twitter.com/SQLMaria

https://blogs.oracle.com/optimizer/

https://sqlmaria.com

https://www.facebook.com/SQLMaria