
What Is New In The
Cost Based Optimizer In
Oracle 12c
Jože Senegačnik
Oracle ACE Director
joze.senegacnik@dbprof.com

About the Speaker

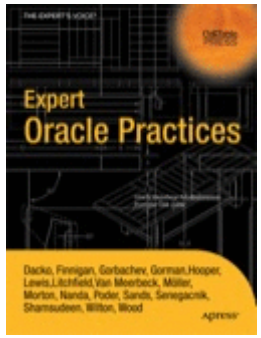
Jože Senegačnik

- Registered private researcher
- First experience with Oracle Version 4 in 1988
- 24 years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com

OakTable.net

ORACLE ACE Director

Oracle ACE



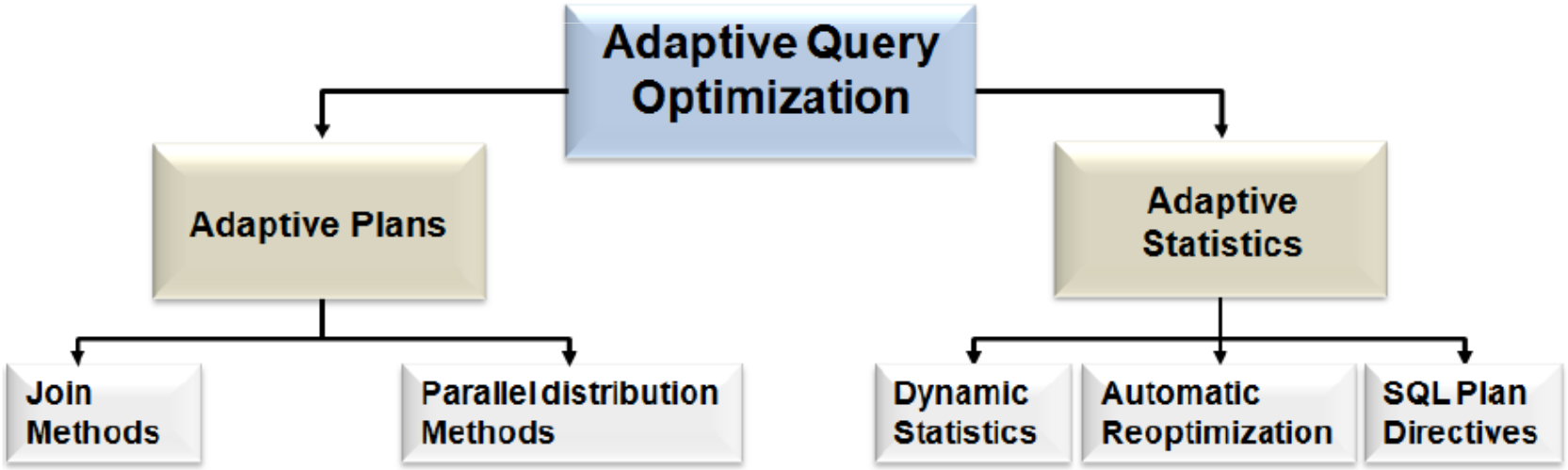
- PPL(A) – private pilot license PPL(A) / instrument rated IR/SE
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

Agenda

- Adaptive Query Optimization
- Online statistics gathering
- Dynamic Statistics (a.k.a dynamic sampling)
- SQL plan directives
- New types of histograms
  - **Top-Frequency histograms**
  - **Hybrid histograms**
- Session level statistics on Global Temporary Tables
- SQL Plan Management enhancements

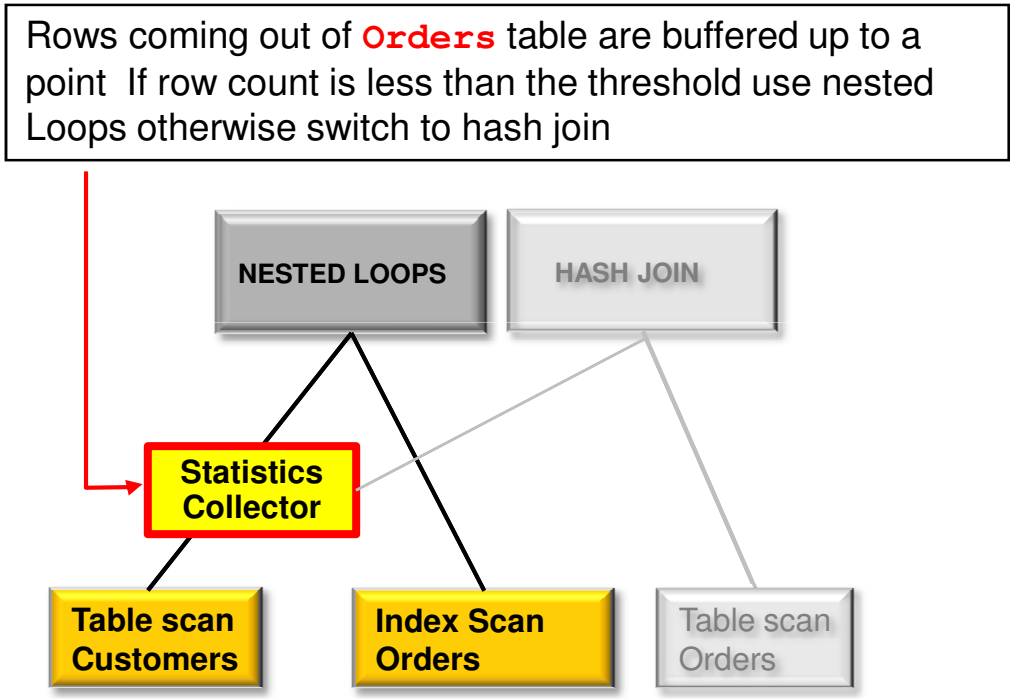
Adaptive Query Optimization

- Optimizer learning from actual execution and „own mistakes“.



Adaptive (Execution) Plan

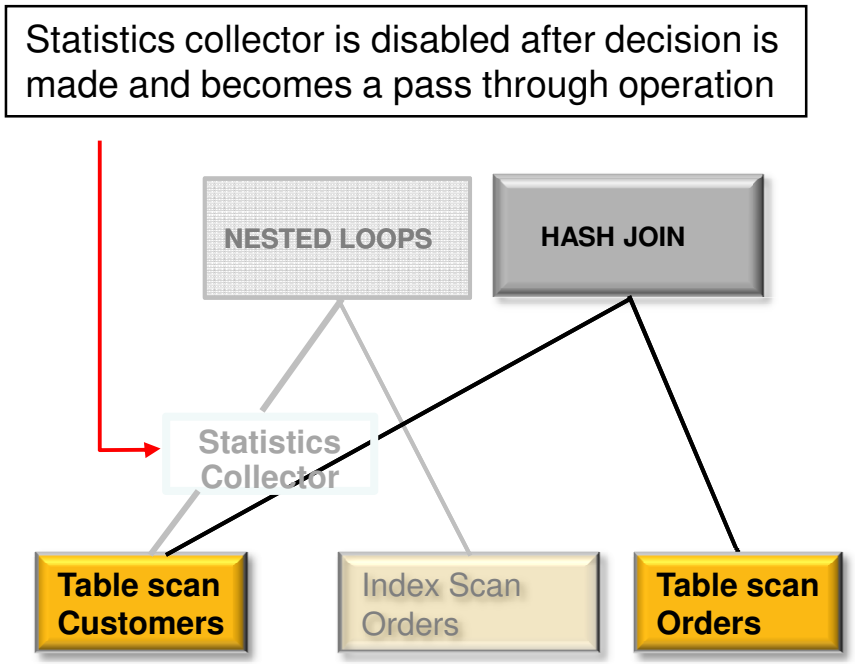
- Alternative sub-plans are prepared
- Sub-plans are stored in the cursor
- Stats collect is inserted before join
- Rows are buffered until final decision is made



Default plan is a NESTED LOOP (NL) join

Adaptive Plan

- If number of rows seen in statistics collector exceeds threshold
- Plan switches to hash join
- Statistics collect becomes disabled
- Plan is resolved on first execution and remains the same for subsequent executions



Final Plan is a HASH JOIN (HJ)

Adaptive Plan - Demo

- Random SQL statement with adaptive plan selected from V$SQL for demonstration

```
SELECT /*+ CONNECT_BY_FILTERING */ s.privilege# FROM sys.sysauth$ s
 CONNECT BY s.grantee# = PRIOR s.privilege#
 AND (s.privilege# > 0 OR s.privilege# = -352)
 START WITH (s.privilege# > 0 OR s.privilege# = -352)
 AND s.grantee# IN
   (SELECT c1.privilege# FROM sys.codeauth$ c1 WHERE c1.obj# = :1)
UNION
SELECT c2.privilege# FROM sys.codeauth$ c2 WHERE c2.obj# = :2
ORDER BY 1 ASC
```

```
SQL> select * from table(dbms_xplan.display_cursor('gngtvs38t0060',format=>'+adaptive'));
-------------------------------------------------------------------------------------------------------
|   Id  | Operation                                | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT                         |              |       |       |     3 (100)|          |
|     1 |  SORT UNIQUE                             |              |     5 |   130 |     3   (0)| 00:00:01 |
|     2 |   UNION-ALL                              |              |       |       |            |          |
|  *  3 |    CONNECT BY WITH FILTERING (UNIQUE)    |              |       |       |            |          |
|- *  4 |     HASH JOIN                            |              |     1 |    35 |     1   (0)| 00:00:01 |
|     5 |      NESTED LOOPS                        |              |     1 |    35 |     1   (0)| 00:00:01 |
|-    6 |       STATISTICS COLLECTOR               |              |       |       |            |          |
|  *  7 |        INDEX RANGE SCAN                  | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
|  *  8 |       INDEX RANGE SCAN                   | I_SYSAUTH1   |     1 |     9 |     1   (0)| 00:00:01 |
|- *  9 |      INDEX FAST FULL SCAN                | I_SYSAUTH1   |     1 |     9 |     1   (0)| 00:00:01 |
|    10 |     NESTED LOOPS                         |              |     3 |    66 |     2   (0)| 00:00:01 |
|    11 |      CONNECT BY PUMP                     |              |       |       |            |          |
|  * 12 |      INDEX RANGE SCAN                    | I_SYSAUTH1   |     3 |    27 |     1   (0)| 00:00:01 |
|  * 13 |    INDEX RANGE SCAN                      | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("S"."GRANTEE#"=PRIOR NULL)
   4 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
   7 - access("C1"."OBJ#"=:1)
   8 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
   9 - filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  12 - access("S"."GRANTEE#"="connect$_by$_pump$_003"."PRIOR s.privilege#      $POS99")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  13 - access("C2"."OBJ#"=:2)
Note
-----
   - this is an adaptive plan (rows marked '-' are inactive)
```

```
SQL> select * from table(dbms_xplan.display_cursor('gngtvs38t0060'));

-----------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |              |       |       |     3 (100)|          |
|   1 |  SORT UNIQUE                           |              |     5 |   130 |     3   (0)| 00:00:01 |
|   2 |   UNION-ALL                            |              |       |       |            |          |
|*  3 |    CONNECT BY WITH FILTERING (UNIQUE)|              |       |       |            |          |
|   4 |     NESTED LOOPS                       |              |     1 |    35 |     1   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                  | I_CODEAUTH1 |     1 |    26 |     0   (0)|          |
|*  6 |      INDEX RANGE SCAN                  | I_SYSAUTH1  |     1 |     9 |     1   (0)| 00:00:01 |
|   7 |     NESTED LOOPS                       |              |     3 |    66 |     2   (0)| 00:00:01 |
|   8 |      CONNECT BY PUMP                   |              |       |       |            |          |
|*  9 |      INDEX RANGE SCAN                  | I_SYSAUTH1  |     3 |    27 |     1   (0)| 00:00:01 |
|* 10 |    INDEX RANGE SCAN                    | I_CODEAUTH1 |     1 |    26 |     0   (0)|          |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("S"."GRANTEE#"=PRIOR NULL)
   5 - access("C1"."OBJ#"=:1)
   6 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
   9 - access("S"."GRANTEE#"="connect$_by$_pump$_003"."PRIOR s.privilege#      $POS99")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  10 - access("C2"."OBJ#"=:2)

Note
-----
   - this is an adaptive plan
```

Determining Adaptive Plans

- New column added V$SQL.IS_RESOLVED_ADAPTIVE_PLAN
- Values
  - Y – final plan was determined
  - N – final plan was not determined yet (not executed yet or executing but final plan was not determined yet)
  - NULL – non-adaptive plan

```
SQL> select nvl(IS_RESOLVED_ADAPTIVE_PLAN,'NULL') as val, count(*)
     from V$SQL
     group by nvl(IS_RESOLVED_ADAPTIVE_PLAN,'NULL');

VAL     COUNT(*)
---- ----------
Y          1153
NULL       7971
N            22 -> all SQLs were not executed yet (executions = 0)
```

Displaying Adaptive Plans

- FINAL Execution Plan

```
select * from
table(dbms_xplan.display_cursor('gngtvs38t0060'));

Note
-----
 - this is an adaptive plan
```

- ADAPTIVE Execution Plan

```
select * from table(dbms_xplan.display_cursor('gngtvs38t0060',
       format=>'+adaptive'));

Note
-----
  - this is an adaptive plan (rows marked '-' are inactive)
```

Disabling Adaptive Plans

- OPTIMIZER_ADAPTIVE_REPORTING_ONLY parameter
  - FALSE (default) – enable adaptive plan
  - TRUE – just report possibilities, but not change the plan. The default plan will always be used.
  - Information is collected on how the plan would have adapted in a non-reporting mode.
- Display execution plan in reporting only mode

```
select *
from table(dbms_xplan.display_cursor('gngtvs38t0060',
        format=>'+report'));
```

Adaptive Parallel Distribution Methods

- In parallel execution certain operations (sorts, aggregations, joins) require data to be redistributed among the parallel server processes.
- Crucial for parallel join of very small data sets with very large data sets.
- Statistics collector is inserted in order to determine the best method:
  - HASH
  - BROADCAST
- New operation is called HYBRID HASH
- If actual number of rows less than threshold, switch from HASH to Broadcast
  - Threshold number of total rows < 2x DOP
- Enabled by default

Dynamic Statistics

- New level 11 for DYNAMIC SAMPLING added
- CBO automatically decide to use dynamic statistics for and SQL statement
- Existing DBMS_STATS statistic can be improoved for:
  - Situations where optimizer was making a guess (LIKE predicates, wildcards, ..)
- The dynamic sampling results will be persisted in the cache, as dynamic statistics, allowing other SQL statements to share these statistics.

```
SQL> show parameter dynamic

NAME                                 TYPE         VALUE
------------------------------------ ------------ -----
optimizer_dynamic_sampling           integer      2

SQL> alter system set optimizer_dynamic_sampling = 11;
System altered.
```

Dynamic Statistics

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
SQL_ID  4ua5q8cgupu22, child number 0
-------------------------------------
select * from sh.sales where prod_id in (113,114,115)
------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |       |       |   517 (100)|          |       |       |
|   1 |  PARTITION RANGE ALL|       | 40222 |  1139K|   517   (2)| 00:00:01 |     1 |    28 |
|*  2 |   TABLE ACCESS FULL | SALES | 40222 |  1139K|   517   (2)| 00:00:01 |     1 |    28 |
------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("PROD_ID"=113 OR "PROD_ID"=114 OR "PROD_ID"=115))

Note
-----
   - dynamic statistics used: dynamic sampling (level=AUTO)
```

- Actual number of rows is 40222

Automatic Reoptimization

- Known also as Automatic Cardinality Feedback in 11g
- 12*c* supports multiple forms of reoptimization.
  - Statistics feedback (cardinality feedback)
  - Performance feedback

```
SQL> select IS_REOPTIMIZABLE, count(*) as COUNT
  2  from v$sql group by IS_REOPTIMIZABLE;

I      COUNT
- ----------
Y        911
N       8927
```

Statistics Feedback

- CBO enables statistics feedback in the following cases:
  - tables with no statistics,
  - multiple conjunctive or disjunctive filter predicates on a table,
  - predicates containing complex operators for which the optimizer cannot accurately compute cardinality estimates.
- After execution CBO compares original cardinality estimates to the actual cardinalities from the execution.
- If estimates differ significantly from actual cardinalities, CBO stores the correct estimates for subsequent use.
- CBO also creates SQL plan directive so other SQL statements can benefit as well (see later demo).
- If the original estimates are found o.k. then the reoptimization flag is reset and no further monitoring is performed.

Statistics Feedback

```
select /*+ gather_plan_statistics */
     c.cust_first_name, c.cust_last_name,
sum(s.amount_sold)
from sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_city = 'Los Angeles'
and c.cust_state_province = 'CA'
and c.country_id = 52790
and s.time_id = '19-FEB-00'
group by  c.cust_first_name, c.cust_last_name;
```

Dependent columns – break CBO rule of independency

First Execution

```
select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name           | Starts | E-Rows | A-Rows |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                |      1 |        |      3 |
|   1 |  HASH GROUP BY                               |                |      1 |      1 |      3 |
|   2 |   NESTED LOOPS                               |                |      1 |        |     10 |
|   3 |    NESTED LOOPS                              |                |      1 |      1 |    601 |
|   4 |     PARTITION RANGE SINGLE                   |                |      1 |    601 |    601 |
|   5 |      TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| SALES          |      1 |    601 |    601 |
|   6 |       BITMAP CONVERSION TO ROWIDS            |                |      1 |        |    601 |
|*  7 |        BITMAP INDEX SINGLE VALUE             | SALES_TIME_BIX |      1 |        |      1 |
|*  8 |     INDEX UNIQUE SCAN                        | CUSTOMERS_PK   |    601 |      1 |    601 |
|*  9 |    TABLE ACCESS BY INDEX ROWID               | CUSTOMERS      |    601 |      1 |     10 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   7 - access("S"."TIME_ID"='19-FEB-00')
   8 - access("C"."CUST_ID"="S"."CUST_ID")
   9 - filter(("C"."CUST_CITY"='Los Angeles' AND "C"."CUST_STATE_PROVINCE"='CA' AND
              "C"."COUNTRY_ID"=52790))
```

Second Execution

```
select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name            | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                 |      1 |        |      3 |
|   1 |  HASH GROUP BY                               |                 |      1 |      3 |      3 |
|   2 |   NESTED LOOPS                               |                 |      1 |        |     10 |
|   3 |    NESTED LOOPS                              |                 |      1 |     10 |    601 |
|   4 |     PARTITION RANGE SINGLE                   |                 |      1 |    601 |    601 |
|   5 |      TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| SALES           |      1 |    601 |    601 |
|   6 |       BITMAP CONVERSION TO ROWIDS            |                 |      1 |        |    601 |
|*  7 |        BITMAP INDEX SINGLE VALUE             | SALES_TIME_BIX  |      1 |        |      1 |
|*  8 |     INDEX UNIQUE SCAN                        | CUSTOMERS_PK    |    601 |      1 |    601 |
|*  9 |    TABLE ACCESS BY INDEX ROWID               | CUSTOMERS       |    601 |      1 |     10 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   7 - access("S"."TIME_ID"='19-FEB-00')
   8 - access("C"."CUST_ID"="S"."CUST_ID")
   9 - filter(("C"."CUST_CITY"='Los Angeles' AND "C"."CUST_STATE_PROVINCE"='CA' AND
              "C"."COUNTRY_ID"=52790))
Note
-----
   - statistics feedback used for this statement
```

SQL Plan Directives

- SQL plan directive is additional information that the optimizer uses to generate a more optimal execution plan.
- They are created for:
  - Data skew in join column, force using dynamic statistics
  - Query expressions rather than at a statement or object level in order to be sharable among more SQL statements
- Automatically created by CBO via Automatic Reoptimization.

See Created SQL Plan Directives

- First flush directives from shared pool to see them with the above query

```
SQL> exec dbms_spd.FLUSH_SQL_PLAN_DIRECTIVE

SQL> select to_char(d.directive_id) as dir_id, o.owner, o.object_name,
           o.subobject_name as col_name, o.object_type, d.type,d.state,d.reason
     from dba_sql_plan_directives d, dba_sql_plan_dir_objects o
     where d.directive_id = o.directive_id
     and o.owner='SH'
     order by 1,2,3,4,5;
```

DIR_ID	OWNER	OBJECT_NA	COL_NAME	OBJECT TYPE	STATE	REASON
18037153769499555617	SH	CUSTOMERS	COUNTRY_ID	COLUMN DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS	CUST_CITY	COLUMN DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS	CUST_STATE_PROVINCE	COLUMN DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS		TABLE DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
3240241301808666714	SH	CUSTOMERS		TABLE DYNAMIC_SAMPLING	NEW	JOIN CARDINALITY MISESTIMATE
3240241301808666714	SH	SALES		TABLE DYNAMIC_SAMPLING	NEW	JOIN CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS	CUST_FIRST_NAME	COLUMN DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS	CUST_LAST_NAME	COLUMN DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS		TABLE DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	SALES		TABLE DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE

SQL Plan Directives versus Extended Statistics

```
SQL> exec dbms_stats.gather_table_stats('SH','CUSTOMERS')

SQL> select table_name, extension_name, extension
  2  from dba_stat_extensions
  3  where owner='SH'
  4  and table_name='CUSTOMERS';
```

```
TABLE_NAME      EXTENSION_NAME                 EXTENSION
--------------- ------------------------------ --------------------------------------------------
CUSTOMERS       SYS_STSMZ$C3AIHLPBROI#SKA58H_N ("CUST_CITY","CUST_STATE_PROVINCE","COUNTRY_ID")
CUSTOMERS       SYS_STS_RXGTQQIMWO0T8EDCN5TKOK ("CUST_FIRST_NAME","CUST_LAST_NAME")
```

- Extended statistics is automatically created/gathered from directives when statistics is gathered.

Changed State of Directive

```
DIR_ID              OWNER OBJECT_NAME  COL_NAME            OBJECT TYPE              STATE         REASON
------------------- ----- ------------ ------------------- ------ ----------------- ------------- -----------------------------------
1803715376949955617 SH    CUSTOMERS    COUNTRY_ID          COLUMN DYNAMIC_SAMPLING HAS_STATS     SINGLE TABLE CARDINALITY MISESTIMATE
1803715376949955617 SH    CUSTOMERS    CUST_CITY           COLUMN DYNAMIC_SAMPLING HAS_STATS     SINGLE TABLE CARDINALITY MISESTIMATE
1803715376949955617 SH    CUSTOMERS    CUST_STATE_PROVINCE COLUMN DYNAMIC_SAMPLING HAS_STATS     SINGLE TABLE CARDINALITY MISESTIMATE
1803715376949955617 SH    CUSTOMERS                        TABLE  DYNAMIC_SAMPLING HAS_STATS     SINGLE TABLE CARDINALITY MISESTIMATE
3240241301808666714  SH    CUSTOMERS                        TABLE  DYNAMIC_SAMPLING MISSING_STATS JOIN CARDINALITY MISESTIMATE
3240241301808666714  SH    SALES                            TABLE  DYNAMIC_SAMPLING MISSING_STATS JOIN CARDINALITY MISESTIMATE
7939480357619414784  SH    CUSTOMERS    CUST_FIRST_NAME     COLUMN DYNAMIC_SAMPLING HAS_STATS     GROUP BY CARDINALITY MISESTIMATE
7939480357619414784  SH    CUSTOMERS    CUST_LAST_NAME      COLUMN DYNAMIC_SAMPLING HAS_STATS     GROUP BY CARDINALITY MISESTIMATE
7939480357619414784  SH    CUSTOMERS                        TABLE  DYNAMIC_SAMPLING HAS_STATS     GROUP BY CARDINALITY MISESTIMATE
7939480357619414784  SH    SALES                            TABLE  DYNAMIC_SAMPLING HAS_STATS     GROUP BY CARDINALITY MISESTIMATE
```

```
SQL> select state, count(*) from dba_sql_plan_directives group by state;

STATE          COUNT(*)
-------------- ----------
MISSING_STATS          99
PERMANENT             229
HAS_STATS             628
NEW                   254
```

```
SQL> select reason, count(*) from dba_sql_plan_directives group by reason;

REASON                                    COUNT(*)
----------------------------------------- ----------
JOIN CARDINALITY MISESTIMATE                     921
SINGLE TABLE CARDINALITY MISESTIMATE             254
GROUP BY CARDINALITY MISESTIMATE                  35
```

Performance Feedback in Parallel Execution

- Related to degree of parallelism (DOP)
- Helps improving the degree of parallelism chosen for repeated SQL statements when Automatic Degree of Parallelism (AutoDOP) is enabled in adaptive mode.
- Additional performance monitoring is enabled for the initial execution of any SQL statement executing in parallel.
- After execution parallel degree chosen by the CBO is compared to the to the actual performance statistics (like CPU time used) gathered during the first execution.
- If the values vary significantly then the statement is marked for reoptimization and the initial execution performance statistics are stored as feedback to help compute a more appropriate degree of parallelism for subsequent executions.

New Types of Histograms

- Before 12c there are two types of histograms available:
  - Frequency Histogram
  - Height-Balanced Histogram
- In 12c we got two new:
  - Top Frequency
  - Hybrid
- Frequency histogram can have more then 254 buckets (up to 2048).

Histograms In Default 12c Database

```
SQL> select histogram, count(*) cnt
  2  from dba_tab_col_statistics
  3  group by histogram;

HISTOGRAM                CNT
---------------- ----------
FREQUENCY              2855
HYBRID                  266
TOP-FREQUENCY             9
NONE                  60551
```

Gathering Histograms

```
SQL> exec dbms_stats.gather_table_stats('SH','SALES',
          estimate_percent=>dbms_stats.AUTO_SAMPLE_SIZE,
          method_opt=>'for all columns size 254')

PL/SQL procedure successfully completed.

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS HISTOGRAM
------------------------------ ------------ ---------- ---------------
PROD_ID                                  72          0 FREQUENCY
CUST_ID                                7059          0 HYBRID
TIME_ID                                1460          0 HYBRID
CHANNEL_ID                                4          0 FREQUENCY
PROMO_ID                                  4          0 FREQUENCY
QUANTITY_SOLD                             1          0 FREQUENCY
AMOUNT_SOLD                            3586          0 HYBRID
```

Gathering Histograms

```
SQL> exec dbms_stats.gather_table_stats('SH','SALES',
          estimate_percent=>dbms_stats.AUTO_SAMPLE_SIZE,
          method_opt=>'for all columns size 2048')

PL/SQL procedure successfully completed.

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS HISTOGRAM
------------------------------ ------------ ---------- ---------------
PROD_ID                                  72          0 FREQUENCY
CUST_ID                                7059          0 HYBRID
TIME_ID                                1460          0 FREQUENCY
CHANNEL_ID                                4          0 FREQUENCY
PROMO_ID                                  4          0 FREQUENCY
QUANTITY_SOLD                             1          0 FREQUENCY
AMOUNT_SOLD                            3586          0 HYBRID
```

Top-Frequency Histogram

- Freqnecy histogram is created when NDV <= 254
- But if a relatively small number of values occupies most of the rows (>99% rows) a frequency histogram on that small set of values is very useful.
- Unpopular values are ignored.
- Gathering mecahnism behind is the same as for frequency histograms
- Top-Frequency Histograms are only created with AUTO_SAMPLE_SIZE

Top-Frequency Histogram

```
SQL> select ATTR,count(*) from SYSMAN.EM_CCS_PARSED_DATA group by attr order by 2;

ATTR                                 COUNT(*)

@STARTMODE                                1
domain-version                            1
SSLSessionCacheTimeout                    1
DumpIOLogLevel                            1
...
...
...
staging-mode                             86
scope                                    86
security-dd-model                        93
source-path                              93
target                                  110
action                                  113
type                                    230
value                                   281
name                                    961
text_value                             1418

261 rows selected.
```

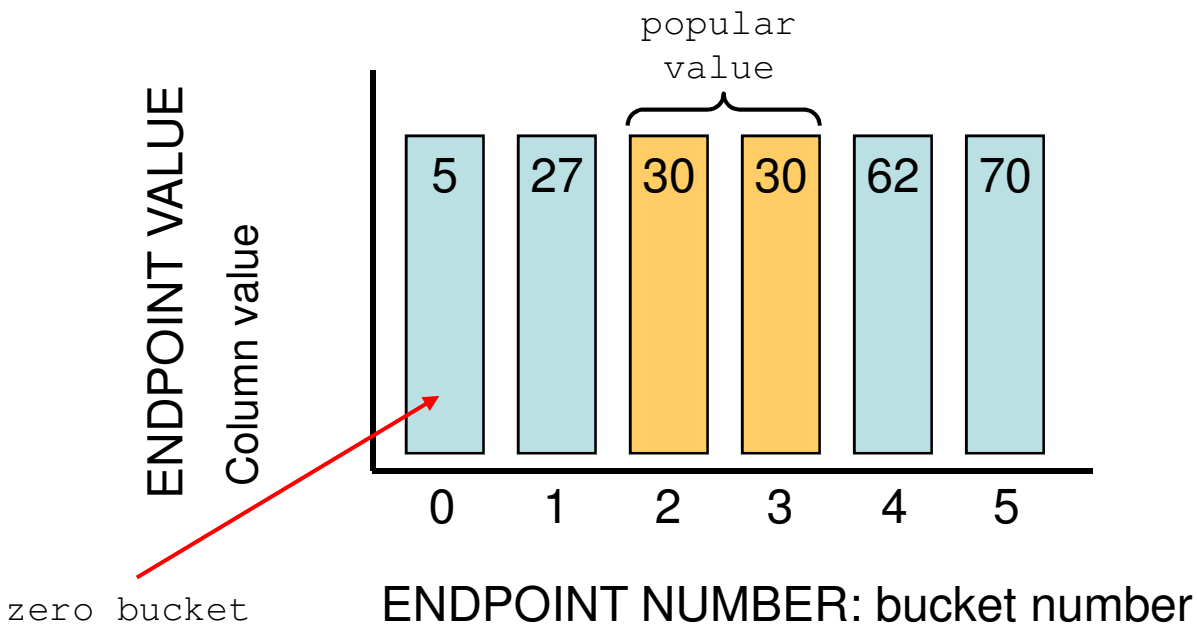
Un-popular values are ignored

Hybrid Histogram

- Like old fashion height balanced histogram when NDV > 254, but also contains elements of a frequency histogram as well as the “bucket” approach of the height-balanced.
- Store the actual frequencies of bucket endpoints in the histogram.
- No values are 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
- Frequency of endpoint values recorded in new column called ENDPOINT_REPEAT_COUNT.

Existing Height Balanced Histograms

5 buckets, 10 distinct values
(2000 rows per bucket)



Hybrid Histogram

```
SQL> select endpoint_number,endpoint_value,endpoint_repeat_count
  2  from DBA_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
--------------- -------------- --------------------
              2        2450815                    2
             24        2450829                    1
             47        2450837                    5
...
...
           5542        2452272                    5
           5543        2452273                    1
           5550        2452275                    3

254 rows selected.
```

Online Statistics Gathering

- Oracle automatically gathers optimizer statistics during the index creation.
- Same functionality now available for direct path operations:
  - create table as select (CTAS)
  - insert as select (IAS)
- Online statistics gathering does not gather histograms or index statistics due to too big overhead.
- To gather the necessary histogram and index statistics without re-gathering the base column statistics use the DBMS_STATS.GATHER_TABLE_STATS procedure with the new options parameter set to 'GATHER AUTO'.
- Prior gathering statistics fire some queries to populate COL_USAGE$ table which keep track of used columns in where clause in queries which will kick creating historam in case of skew distribution.

```
exec dbms_stats.gather_table_stats('SH','SALES3',options=>'GATHER AUTO')
```

Online Statistics Gathering

```
SQL> create table sh.sales3 as select * from sh.sales;

Table created.

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES3';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS HISTOGRAM       NOTES
------------------------------ ------------ ---------- --------------- --------------
AMOUNT_SOLD                            3586          0 NONE            STATS_ON_LOAD
QUANTITY_SOLD                             1          0 NONE            STATS_ON_LOAD
PROMO_ID                                  4          0 NONE            STATS_ON_LOAD
CHANNEL_ID                                4          0 NONE            STATS_ON_LOAD
TIME_ID                                1460          0 NONE            STATS_ON_LOAD
CUST_ID                                7059          0 NONE            STATS_ON_LOAD
PROD_ID                                  72          0 NONE            STATS_ON_LOAD
```

Online Statistics Gathering

```
SQL> select count(*) from sh.sales3 where channel_id=9;

  COUNT(*)
----------
      2074

exec dbms_stats.gather_table_stats('SH','SALES3', options=>'GATHER AUTO')

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES3';
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NOTES
AMOUNT_SOLD	3586	0	NONE	STATS_ON_LOAD
QUANTITY_SOLD	1	0	NONE	STATS_ON_LOAD
PROMO_ID	4	0	NONE	STATS_ON_LOAD
CHANNEL_ID	4	0	FREQUENCY	HISTOGRAM_ONLY
TIME_ID	1460	0	NONE	STATS_ON_LOAD
CUST_ID	7059	0	NONE	STATS_ON_LOAD
PROD_ID	72	0	NONE	STATS_ON_LOAD

Session private statistics for GTT

- Shared GTT statistics are not always optimal
- Until 12c statistics gathered on GTT are shared by all sessions
- In 12c each session can have its own version of GTT statistics
- Default value is SESSION (non shared)
- Controlled by new preference GLOBAL_TEMP_TABLE_STATS
- To force sharing (as in 11g) set table preference to SHARED

Session private statistics for GTT

```
SQL> create global temporary table my_gtt( c1 number);
Table created.


SQL> select dbms_stats.get_prefs('GLOBAL_TEMP_TABLE_STATS','JOC','MY_GTT') from dual;

DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','JOC','MY_GTT')
--------------------------------------------------------------
SESSION


SQL> exec dbms_stats.set_table_prefs('JOC','MY_GTT','GLOBAL_TEMP_TABLE_STATS','SHARED');


SQL> select dbms_stats.get_prefs('GLOBAL_TEMP_TABLE_STATS','JOC','MY_GTT') from dual;

DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','JOC','MY_GTT')
--------------------------------------------------------------
SHARED
```

SQL Plan Management enhancements

- New SPM Evolve Advisor
- Runs in the maintenance window as a task called SYS_AUTO_SPM_EVOLVE_TASK.
- Information on task found in DBA_ADVISOR_TASKS.
- Ranks all non-accepted plans and runs evolve process for them.
- If new plan performs better than the existing plan it is automatically accepted!!
- Monitor what was accepted with DBMS_SPM. REPORT_AUTO_EVOLVE_TASK

Features Missing In This Presentation

- Enhanced incremental statistics
- Concurrent statistics gathering
- Statistic gathering reporting
- Automatic detection of column groups
- New Transformations
  - Null-Accepting Semi Join
  - Scalar Subquery Unnesting
- .....

Conclusions

- Optimizer is now learning from own mistakes.
- Wrong Statistics = Wrong Plan
- Big efforts to utilize every oportunity for statistics improvement.
- New optimization paths.
- Let us see what will the real practice show.

Thank you for your interest!

Q&A