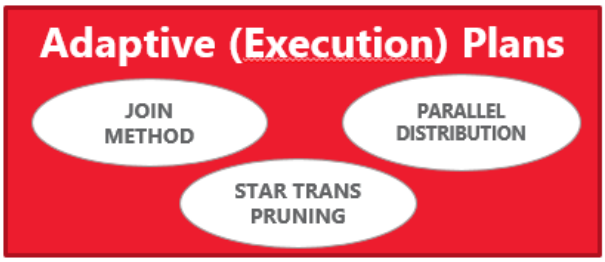
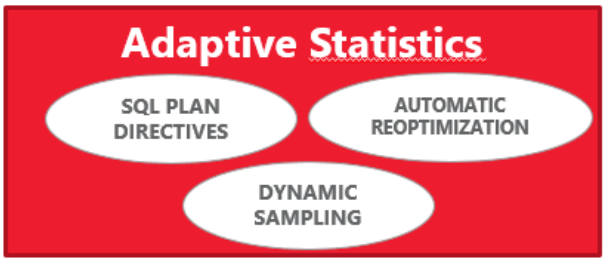
ADAPTIVE FEATURES OR: HOW I LEARNED TO STOP WORRYING AND TROUBLESHOOT THE BOMB

Ludovico Caldara
Senior Consultant at Trivadis, Oracle ACE Director

Oracle Database 12c became available to the public back in 2013, but it took more than one year for the Oracle customers to start upgrading of their existing databases to this new release. Many customers, in 2016, are still not in the process of migrating to 12c despite the premier support deadline for Oracle Database 11g has passed in January 2015.
I had the chance to spend the last two years by a customer who decided to embrace the new release and start the migration to 12c as soon as possible, in order to take the most out of the (many) new features that this release offers. When the very first production databases have been migrated to 12c, the users began noticing quite soon that some queries started to take much more time to complete, some of them were actually several orders of magnitude slower than before. After small investigation, I understood that most off those queries have been slowed down by the new "Adaptive Features" that have been introduced in 12c for the opposite reason: increasing performance. This is what this article is about.

Let's start with a quick recap of the new features.





Adaptive Features (Adaptive Query Optimization) fall into two distinct categories: the **Adaptive Plans** and the **Adaptive Statistics**.

ADAPTIVE PLANS

Adaptive Plans, as the name says, are execution plans that can "adapt" during the execution phase and change more or less dynamically.
When the query optimizer parses a new cursor, it can be unsure about what the correct operation would be. In that case, it will create an adaptive plan: it will define a default starting plan and will add and hide an alternative branch of the plan that will stay inactive during the first part of the execution; it will also add a STATISTICS COLLECTOR operator to the plan for each decision it has to take (that's it: there will be a collector for each decisional branch inside the adaptive plan): it will buffer the result set and check whether the inflection point is reached. If it is the case, the STATISTICS COLLECTOR will disable the current branch of the plan and activate the alternative branch. The STATISTICS COLLECTOR will be disabled and, if it is the last adaptive operation in the plan, the plan will be marked as final.
How does an adaptive plan look like?

```
SQL> explain plan FOR
  2  SELECT C.CUST_EMAIL,
  3    OI.PRODUCT_ID
  4  FROM CUSTOMERS C
  5  JOIN orders O
  6  ON O.CUSTOMER_ID=C.CUSTOMER_ID
  7  JOIN order_items OI
  8  ON OI.ORDER_ID=O.ORDER_ID;
```

```
Explained.

SQL> select * from table(dbms_xplan.display(format=>'+adaptive'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
Plan hash value: 4045828612

-------------------------------------------------------------------------------------------------------------------
|   Id  | Operation                                | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT                         |                 |   665 | 35910 |     9   (0)| 00:00:01 |
|  *  1 |  HASH JOIN                               |                 |   665 | 35910 |     9   (0)| 00:00:01 |
|-    2 |   NESTED LOOPS                          |                 |   665 | 35910 |     9   (0)| 00:00:01 |
|-    3 |    STATISTICS COLLECTOR                  |                 |       |       |            |          |
|  *  4 |     HASH JOIN                            |                 |   105 |  4830 |     7   (0)| 00:00:01 |
|-    5 |      NESTED LOOPS                        |                 |   105 |  4830 |     7   (0)| 00:00:01 |
|-    6 |       STATISTICS COLLECTOR               |                 |       |       |            |          |
|     7 |        TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS          |   105 |   840 |     2   (0)| 00:00:01 |
|  *  8 |         INDEX RANGE SCAN                 | ORD_CUSTOMER_IX |   105 |       |     1   (0)| 00:00:01 |
|-    9 |       TABLE ACCESS BY INDEX ROWID        | CUSTOMERS       |     1 |    38 |     5   (0)| 00:00:01 |
|- * 10 |        INDEX UNIQUE SCAN                 | CUSTOMERS_PK    |       |       |            |          |
|    11 |      TABLE ACCESS FULL                   | CUSTOMERS       |   319 | 12122 |     5   (0)| 00:00:01 |
|- * 12 |    INDEX RANGE SCAN                      | ORDER_ITEMS_UK  |     6 |    48 |     2   (0)| 00:00:01 |
|    13 |   INDEX FAST FULL SCAN                   | ORDER_ITEMS_UK  |   665 |  5320 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("OI"."ORDER_ID"="O"."ORDER_ID")
   4 - access("O"."CUSTOMER_ID"="C"."CUSTOMER_ID")
   8 - access("O"."CUSTOMER_ID">0)
  10 - access("O"."CUSTOMER_ID"="C"."CUSTOMER_ID")
  12 - access("OI"."ORDER_ID"="O"."ORDER_ID")

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

In this example, there are two joins that might change during the execution of the statement.
In order to get the full execution plan including the inactive operations, the +adaptive keyword is required in the format as previously shown.

There are three basic types of dynamic plans: **Parallel Distribution Method**, **Join Method** and **Star Transformation Bitmap Pruning**.
For all types, the optimizer calculates at the parse time an “inflection point”: the number of records in input (either in input for the parallel distribution, coming from the inner table of a join, or filtered by the dimension in a BITMAP MERGE during a star transformation) after which it would be more convenient to use a method rather than the other one.

PARALLEL DISTRIBUTION METHOD

The **Parallel Distribution Method** feature, introduces a new distribution named Hybrid Hash that can change depending on the inflection point that is, in this case, 2 times the parallel degree: if the rows returned in input are equal or greater than the inflection point, Hash/Hash is used; otherwise Broadcast/Round-Robin is used.

JOIN METHOD

With the **Join Method,** the query optimizer calculates an inflection point that, if reached during the query execution, can switch the plan from Nested Loops to Hash Join (or the opposite, depending on the number of lines that were estimated for the inner table during the parse).

One important difference between Join Method and the Parallel Distribution is that the latter can switch plan at every single execution of the cursor, whereas for the Join Method the choice can be done only once during the very first execution. It is possible to know if the final plan has been chosen by querying the new column IS_RESOLVED_ADAPTIVE_PLAN in GV$SQL.

```
SQL> select sql_id, child_number, IS_RESOLVED_ADAPTIVE_PLAN
2 from v$sql where sql_text like 'SELECT C.CUST_EMAIL%';

SQL_ID        CHILD_NUMBER I
------------- ------------ -
8d1g1zszjk0ky            0 Y
```

What does it mean from a performance point of view? Because only the first execution can influence the final plan, nothing really changes from the traditional execution plans. Once the final plan is effective, the cursor will always use the same plan, unless Adaptive Cursor Sharing kicks in and creates another child cursor, which will likely produce another Adaptive Plan.
How is the inflection point calculated for the join method? By looking at the optimizer trace of a query that produces an adaptive plan, it is evident that the optimizer tries to cost the two operations (NL and HJ) for different number of rows that will potentially be returned by the inner table:

```
SQL> alter session set tracefile_identifier='doag16_ds_2';
Session altered.

SQL> alter session set events '10053 trace name context forever, level 1';
Session altered.

SQL> SELECT C.CUST_EMAIL,
2  OI.PRODUCT_ID
3  FROM OE.CUSTOMERS C
4  JOIN OE.orders O
5  ON O.CUSTOMER_ID=C.CUSTOMER_ID
6  JOIN OE.order_items OI
7  ON OI.ORDER_ID=O.ORDER_ID;

...
665 rows selected.

SQL> alter session set events '10053 trace name context off';
Session altered.

SQL> ! grep -i "inflection point" *doag16_ds_2.trc
Searching for inflection point (join #1) between 0.00 and 105.00
AP: Computing costs for inflection point at min value 0.00
AP: Using binary search for inflection point search
AP: Costing Nested Loops Join for inflection point at card 0.00
AP: Costing Hash Join for inflection point at card 0.00
AP: Computing costs for inflection point at max value 105.00
AP: Costing Nested Loops Join for inflection point at card 105.00
AP: Costing Hash Join for inflection point at card 105.00
AP: Searching for inflection point at value 1.00
AP: Costing Nested Loops Join for inflection point at card 52.50
AP: Costing Hash Join for inflection point at card 52.50
AP: Searching for inflection point at value 52.50
AP: Costing Nested Loops Join for inflection point at card 26.25
AP: Costing Hash Join for inflection point at card 26.25
AP: Searching for inflection point at value 26.25
AP: Costing Nested Loops Join for inflection point at card 13.12
AP: Costing Hash Join for inflection point at card 13.12
```

```
AP: Searching for inflection point at value 13.12
AP: Costing Nested Loops Join for inflection point at card 6.56
AP: Costing Hash Join for inflection point at card 6.56
AP: Searching for inflection point at value 6.56
AP: Costing Nested Loops Join for inflection point at card 3.28
AP: Costing Hash Join for inflection point at card 3.28
AP: Searching for inflection point at value 3.28
AP: Costing Nested Loops Join for inflection point at card 4.92
AP: Costing Hash Join for inflection point at card 4.92
AP: Searching for inflection point at value 4.92
AP: Costing Nested Loops Join for inflection point at card 5.74
AP: Costing Hash Join for inflection point at card 5.74
AP: Costing Nested Loops Join for inflection point at card 5.74
Searching for inflection point (join #2) between 0.00 and 105.00
AP: Computing costs for inflection point at min value 0.00
AP: Using binary search for inflection point search
AP: Costing Nested Loops Join for inflection point at card 0.00
AP: Costing Hash Join for inflection point at card 0.00
AP: Computing costs for inflection point at max value 105.00
AP: Costing Nested Loops Join for inflection point at card 105.00
AP: Costing Hash Join for inflection point at card 105.00
AP: Searching for inflection point at value 1.00
AP: Costing Nested Loops Join for inflection point at card 52.50
AP: Costing Hash Join for inflection point at card 52.50
AP: Searching for inflection point at value 52.50
AP: Costing Nested Loops Join for inflection point at card 26.25
AP: Costing Hash Join for inflection point at card 26.25
AP: Searching for inflection point at value 26.25
AP: Costing Nested Loops Join for inflection point at card 13.12
AP: Costing Hash Join for inflection point at card 13.12
AP: Searching for inflection point at value 13.12
AP: Costing Nested Loops Join for inflection point at card 6.56
AP: Costing Hash Join for inflection point at card 6.56
AP: Searching for inflection point at value 6.56
AP: Costing Nested Loops Join for inflection point at card 3.28
AP: Costing Hash Join for inflection point at card 3.28
AP: Searching for inflection point at value 3.28
AP: Costing Nested Loops Join for inflection point at card 1.64
AP: Costing Hash Join for inflection point at card 1.64
AP: Searching for inflection point at value 1.64
AP: Costing Nested Loops Join for inflection point at card 2.46
AP: Costing Hash Join for inflection point at card 2.46
AP: Costing Nested Loops Join for inflection point at card 2.46
```

Without trace, it is not possible to know what the inflection point for a statement was.
It is also very hard to know if the plan has changed or not: when a plan switches the join method, the PLAN_HASH_VALUE changes in the dynamic views, but no other evidences let the DBA know that such change occurred.
From the point of view of troubleshooting, in my opinion there is some information missing in this release, but the experience shows that very rarely the adaptive execution plans themselves are a source of problems. Bad Adaptive Plans are rather consequence of other factors, such as bad statistics in the data dictionary, or bad statistics returned by the Adaptive Statistics features.

The last type of Adaptive Plan is the **Star Transformation Bitmap Pruning**. In star schemas, the fact table is joined with several dimension tables. For every dimension having a filter on it, a kind of join between the bitmap index on the fact table and the dimension table takes place. If that join returns many rows, this operation can be bad for the performance. With the Star Transformation Bitmap Pruning method, whenever the STATISTICS

COLLECTOR detects that the dimension does not filter enough, it can prune the access to that dimension: the plan will just skip that dimension and join it later.

ADAPTIVE STATISTICS

Adaptive Statistics is a set of features composed by **Automatic Reoptimization**, **SQL Plan Directives** and **Dynamic Statistics**.
Simply put, Automatic Reoptimization consists of two features named **Statistics Feedback**, the evolution of the Cardinality Feedback that was present also in 11gR2, and **Performance Feedback**.

PERFORMANCE FEEDBACK

In this article I will not talk much about **Performance Feedback**, as it has less negative impact (at least in my experience) to the performance of the databases; but it is worth to mention that depending on the *parallel_degree_policy* parameter, it may store some performance statistics of parallel executions when the DOP is not optimal, triggering the reoptimization of the statement at the next execution.

STATISTICS FEEDBACK

Statistics Feedback has also changed in 12c. It has been introduced in release 11.2 under the name of “Cardinality Feedback”. Simply put, the job of cardinality feedback is to determine, at execution time, if the actual cardinality is much different from the estimated cardinality. When the estimation is particularly bad, a correction is stored in the SQL Area in order to produce a more accurate plan in the future. When it is the case, it triggers the immediate reoptimization of the statement: at the very next execution, a new parse occurs and a new child cursor is created. In 12c, this feature adds feedback about join cardinalities.
It is possible to check if the reoptimization will occur by querying the column V$SQL.IS_REOPTIMIZABLE. Let’s see it in an example:

```
SQL> CREATE TABLE t AS
  2  SELECT rownum AS id,
  3  rownum AS val,
  4  lpad('p',1000) AS pad
  5  FROM dual CONNECT by level <= 50000;

Table created.

SQL> CREATE INDEX ix_t_val ON t (val);

Index created.

SQL> exec dbms_stats.gather_table_stats(user,'T');

PL/SQL procedure successfully completed.

-- Let’s fake the statistics by changing the data

SQL> update t set val=1;

50000 rows updated.

SQL> commit;

Commit complete.

SQL> alter session set statistics_level=all;

Session altered.

SQL> select count(*) from t where val=1 and id>0;

  COUNT(*)
----------
     50000
```

```
SQL> select * from table(dbms_xplan.display_cursor(format=>'allstats last'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
SQL_ID  04drgw5qk4m3u, child number 0
-------------------------------------
 select count(*) from t where val=1 and id>0

Plan hash value: 598156376

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |          |      1 |        |      1 |00:00:00.04 |    7489 |
|   1 |  SORT AGGREGATE                      |          |      1 |      1 |      1 |00:00:00.04 |    7489 |
|*  2 |   TABLE ACCESS BY INDEX ROWID BATCHED| T        |      1 |      1 |  50000 |00:00:00.04 |    7489 |
|*  3 |    INDEX RANGE SCAN                  | IX_T_VAL |      1 |      1 |  50000 |00:00:00.01 |     346 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("ID">0)
   3 - access("VAL"=1)


21 rows selected.

SQL> select sql_id, child_number, executions, is_reoptimizable from v$sql where
sql_id='04drgw5qk4m3u';

SQL_ID        CHILD_NUMBER EXECUTIONS I
------------- ------------ ---------- -
04drgw5qk4m3u            0          1 Y

SQL>  select count(*) from t where val=1 and id>0;

  COUNT(*)
----------
     50000

SQL>  select * from table(dbms_xplan.display_cursor(format=>'allstats last'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
SQL_ID  04drgw5qk4m3u, child number 1
-------------------------------------
 select count(*) from t where val=1 and id>0

Plan hash value: 2966233522

-------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.02 |    7153 |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.02 |    7153 |
|*  2 |   TABLE ACCESS FULL| T    |      1 |  50000 |  50000 |00:00:00.02 |    7153 |
-------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(("VAL"=1 AND "ID">0))

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - statistics feedback used for this statement
   - 1 Sql Plan Directive used for this statement
```

The note section of the second execution introduces the other two features of the Adaptive Statistics, the core of this article: Dynamic Statistics and SQL Plan Directives.

DYNAMIC STATISTICS

Dynamic Statistics is the new name of Dynamic Sampling in 12c. The new release introduces a new implementation of Dynamic Sampling called **Adaptive Dynamic Sampling** (ADS). The Dynamic Sampling level (parameter *optimizer_dynamic_sampling*) controls in which situations the Dynamic Sampling takes place and how many blocks it will read in that case. In 12c, if the level is set to 11, Adaptive Dynamic Sampling is used: this implies that the number of blocks read is not fixed. The actual number of blocks read may vary a lot, and the count of Dynamic Sampling queries executed during the parse of each query can easily reach several dozens. The actual Dynamic Sampling count and text can be found in SQL traces and in the SQL Area: they all begin with: “SELECT /* DS_SVC */”:

```
PARSING IN CURSOR #139986665500256 len=330 dep=1 uid=110 oct=3 lid=110 tim=9486290642425
hv=3320227945 ad='8f845b48' sqlid='f8guurb2yda39'
SELECT /* DS_SVC */ /*+ dynamic_sampling(0) no_sql_tune no_monitoring
optimizer_features_enable(default) no_parallel result_cache(snapshot=3600) */ SUM(C1) FROM (SELECT
/*+ qb_name("innerQuery") NO_INDEX_FFS( "LI")  */ 1 AS C1 FROM ADAPTIVE."PRODUCT" SAMPLE
BLOCK(1.85843, 8) SEED(1)  "LI" WHERE ("LI"."ENT_ID"=194924)) innerQuery
END OF STMT

$ grep DS_SVC theludot_ora_2905_doag16_ds_2.trc | wc -l
5
```

For this reason, the actual time needed for the parse of the statement can take up to several seconds (or minutes!) even for queries that would take a fraction of second to execute. From a performance perspective, it is clear that applications that do a lot of parses do not take big advantage of Adaptive Dynamic Sampling; indeed, it is a feature that benefits mostly OLAP environments.
Despite the fact that the default value of *optimizer_dynamic_sampling* is 2, Adaptive Dynamic Sampling is used instead whenever **Parallel Execution** is considered or whenever **SQL Plan Directives** are used during the parse.

SQL PLAN DIRECTIVES

The Statistics Feedback stores the correction to the misestimated cardinality in the SQL Area for a given cursor. This information is lost after the cursor ages out or gets flushed out of the shared pool.
In 12c, the Statistics Feedback persists the information about the involved objects/columns by creating **SQL Plan Directives** (Note: SQL Plan Directives do NOT store information about the cardinality).
The idea behind that is that if there is a misestimate on an operation on specific table/columns (joins, equality conditions, etc.), the same misestimate may occur also in other statements with similar conditions. For this reason, the SQL Plan Directives might benefit not only the query that leads to their creation, but all the statements involving the same objects.
In a previous example I have shown the creation of a SQL Plan Directive for the query.

```
SQL> select count(*) from t where val=1 and id>0;
```

A slight different query with the same condition, in 11gR2, would cause a hard parse and use again the index because of the wrong statistics. But because the columns in the where clause are the same, the SQL Plan

Directive tells to the optimizer to use Adaptive Dynamic Sampling instead. The plan uses a TABLE ACCESS FULL from the very first execution:

```
SQL> select count(*)+1 from t where val=1 and id>0;

COUNT(*)+1
----------
     50001

SQL> select * from table(dbms_xplan.display_cursor(format=>'allstats last'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
SQL_ID  8ky5d7h1dh3zn, child number 0
-------------------------------------
select count(*)+1 from t where val=1 and id>0

Plan hash value: 2966233522

-----------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.32 |    7154 |   7143 |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.32 |    7154 |   7143 |
|*  2 |   TABLE ACCESS FULL| T    |      1 |  49983 |  50000 |00:00:00.32 |    7154 |   7143 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(("VAL"=1 AND "ID">0))

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 2 Sql Plan Directives used for this statement


24 rows selected.
```

This is the proof that SQL Plan Directives are a good idea and could improve the performance of many statements.
The SQL Plan Directives are listed in the DBA_SQL_PLAN_DIRECTIVES view, and the objects/columns involved are listed in the DBA_SQL_PLAN_DIR_OBJECTS view. Actually, there are three types of misestimates considered (column DBA_SQL_PLAN_DIRECTIVES.REASON):

- JOIN CARDINALITY MISESTIMATE
- SINGLE TABLE CARDINALITY MISESTIMATE
- GROUP BY CARDINALITY MISESTIMATE

In all the cases, SQL Plan Directives force the optimizer to do Adaptive Dynamic Sampling upon the parse of the statement. For single table cardinality misestimates that involve multiple columns, the SQL Plan Directives might also lead to the creation of extended statistics (column groups) when the misestimate involves several columns of the same table. The Dynamic Sampling still takes place in this case until the statistics on the newly created column group are gathered. After that, the corresponding SQL Plan Directive becomes obsolete and will not be used anymore by the optimizer.

THE BOMB If the new adaptive features look so smart, why the customers complain about the poor performance in 12c? Well, like all the new features, the new features in Adaptive Query Optimization work well in most situations, but in a few, more complex cases they can lead to catastrophic results.

What are the symptoms? Depending on the specific situation, customers encounter excessive CPU utilization, excessive Library Cache contention, incremented usage of Result Cache (leading to insufficient Result Cache size) and sub-optimal execution plans.
Why? The formula is always the same: Statistics Feedback generates SQL Plan Directives upon misestimates, SQL Plan Directives force Adaptive Dynamic Sampling, Adaptive Dynamic Sampling consumes a lot of resources, generate contention and, sometimes, provides dynamic statistics to the optimizer that are less than optimal compared to the stored statistics.
With 12c, Statistics Feedback is particularly aggressive and can create SQL Plan Directives for different reasons:

- Lack of column groups
- Histograms of the wrong type/size
- Misestimates of join cardinalities: this is particularly frequent, SQL Plan Directives are created massively on highly normalized schemas and for almost every query consisting of more than a few joins

Even most of the queries on the dictionary views used by the DBAs use Adaptive Dynamic Sampling because of the SQL Plan Directives:

```
SQL> explain plan for select count(*) from dba_tables;

Explained.

SQL> select * from table(dbms_xplan.display(format=>'metrics'));


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 3585475976

-----------------------------------------------------------
| Id  | Operation                     | Name     | E-Rows |
-----------------------------------------------------------
|   0 | SELECT STATEMENT              |          |      1 |
|   1 |  SORT AGGREGATE               |          |      1 |
|*  2 |   HASH JOIN RIGHT OUTER       |          |   4337 |
|   3 |    TABLE ACCESS FULL          | SEG$     |   7496 |
|*  4 |    HASH JOIN RIGHT OUTER      |          |   2340 |
|   5 |     INDEX FULL SCAN           | I_USER2  |    126 |
|*  6 |     HASH JOIN OUTER           |          |   2340 |
|   7 |      NESTED LOOPS OUTER       |          |   2340 |
|*  8 |       HASH JOIN               |          |   2340 |
|   9 |        INDEX FULL SCAN        | I_USER2  |    126 |
|* 10 |        HASH JOIN              |          |   2340 |
|* 11 |         HASH JOIN             |          |   2340 |
|  12 |          MERGE JOIN CARTESIAN |          |     11 |
|* 13 |           HASH JOIN           |          |      1 |
|* 14 |            FIXED TABLE FULL   | X$KSPPI  |      1 |
|  15 |            FIXED TABLE FULL   | X$KSPPCV |    100 |
|  16 |           BUFFER SORT         |          |     11 |
|  17 |            TABLE ACCESS FULL  | TS$      |     11 |
|* 18 |          TABLE ACCESS FULL    | TAB$     |   2340 |
|* 19 |         TABLE ACCESS FULL     | OBJ$     |  92556 |
|* 20 |       INDEX RANGE SCAN        | I_OBJ1   |      1 |
|  21 |      INDEX FAST FULL SCAN     | I_OBJ1   |  92609 |
-----------------------------------------------------------

Sql Plan Directive information:
------------------------------

  Valid directive ids:
```

```
    250882735634668447
    9542766902214822429
    2083381208315424718
    6551113436581718606
    16273932978853091121

  Used directive ids:
    11259849835960924452
    3270421438167445494
    2733281086066737731

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 3 Sql Plan Directives used for this statement

68 rows selected.
SQL>
```

As one can infer from the previous example, the SQL Plan Directives used by a statement can be found by doing explicitly an *explain plan* of the statement and by displaying the plan with the *+metrics* format. Another convenient way to get the SQL Plan Directives used is to produce an optimizer trace (event 10053).

Note: in the previous example, the explain plan shows that the Dynamic Sampling level is 2. In 12.1.0.2, this information is wrong due to a bug. The real level used is 11: when Dynamic Sampling kicks in because SQL Plan Directives, level 11 (ADS) is always used in 12.1.0.2.
Sometimes SQL Plan Directives are created even when the query optimizer produces a good plan: If a big table contains millions of distinct values and the data is very skewed, the histograms may not give a perfect estimate and, despite a good execution plan is created, the Statistics Feedback can still kick in. In that case, when the Adaptive Dynamic Sampling takes place, it may give an even worse estimation and make the Query Optimizer produce a bad plan.
So, what is the solution to this problem?
Actually there are many.

- Some customers decide to deactivate the whole adaptive features by setting *optimizer_adaptive_features* to false: this would stop both the creation and usage of the SQL Plan Directives and the usage of Adaptive Plans. Beware that Adaptive Dynamic Sampling is still available and used for Parallel Executions or when optimizer_*dynamic_sampling* is set to 11.
- Other customers revert the *optimizer_features_enabled* to 11.2.0.4 (or lower). This is generally a bad recommendation as they lose a lot off improvements introduced with this release.

There are of course other, less intrusive solutions:

- Setting the hidden parameter *_optimizer_dsdir_usage_control* to 0 denies the SQL Plan Directives to trigger ADS, but not the creation of the SQL Plan Directives themselves.
- Setting the hidden parameter *_optimizer_enable_extended_stats* to false disables the usage of extended statistics (but not their creation).
- Setting the hidden parameter *_optimizer_adaptive_plans* to false disables the creation of Adaptive Plans (generally speaking, Adaptive Plans rarely introduce problems, but it may apply in your case)
- Setting the hidden parameter ***_sql_plan_directive_mgmt_control*** to 0 disables the creation of SQL Plan Directives; perhaps this is the most handy and useful way to avoid performance stability problems for newly upgraded databases.
- Setting the parameter *optimizer_adaptive_sampling* to 0 disables the usage of Dynamic Sampling globally, including when SQL Plan Directives exist.
- Disabling selectively the SQL Plan Directives is also a solution (the least intrusive): this operation must be done on a per-directive basis. Beware again: disabling a directive that was in internal state MISSING_STATS does not disable the creation of the extended statistics. For example, to disable the SQL Plan Directives used by the query:

```
select count(*)+1 from t where val=1 and id>0;
```

```
SQL> explain plan for select count(*)+1 from t where val=1 and id>0;

Explained.

SQL> select * from table(dbms_xplan.display(format=>metrics'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan hash value: 2966233522

-------------------------------------------
| Id  | Operation          | Name | E-Rows |
-------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |
|   1 |  SORT AGGREGATE    |      |      1 |
|*  2 |   TABLE ACCESS FULL| T    |  49995 |
-------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("VAL"=1 AND "ID">0)

Sql Plan Directive information:
-------------------------------

  Valid directive ids:
    17154883816957472097
    18093103661535947380

28 rows selected.

SQL>  SELECT d.directive_id,
  2    d.state,
  3    d.enabled,
  4    o.object_name ,
  5    o.subobject_name,
  6    d.reason ,
  7    SUBSTR(extract(d.notes, '/spd_note/spd_text/text()' ),1,30) spd_text,
  8    extract(d.notes, '/spd_note/internal_state/text()' ) internal_state,
  9    extract(d.notes, '/spd_note/redundant/text()') redundant
 10  FROM dba_sql_plan_directives d
 11  JOIN dba_sql_plan_dir_objects o
 12  ON (d.directive_id    =o.directive_id)
 13  WHERE d.directive_id IN (17154883816957472097,18093103661535947380)
 14  ORDER BY d.reason DESC,
 15    d.directive_id,
 16    rank() over (partition BY d.directive_id order by rownum ) ,
 17    o.object_name,
 18   o.subobject_name;

        DIRECTIVE_ID  STATE              ENA  OBJECT_NAME           SUBOBJECT_NAME     REASON
SPD_TEXT                       INTERNAL_STATE RED
-------------------- ---------- --- --------------- --------------- --------------------------
---------- ------------------------------ -------------- ---
17154883816957472097 USABLE     YES T               ID              SINGLE TABLE CARDINALITY
MISESTIMATE {C(OE.T)[ID, VAL]}             MISSING_STATS  NO
```

```
17154883816957472097 USABLE     YES T                VAL             SINGLE TABLE CARDINALITY
MISESTIMATE {C(OE.T)[ID, VAL]}            MISSING_STATS  NO
17154883816957472097 USABLE     YES T                                SINGLE TABLE CARDINALITY
MISESTIMATE {C(OE.T)[ID, VAL]}            MISSING_STATS  NO
18093103661535947380 USABLE     YES T                VAL             SINGLE TABLE CARDINALITY
MISESTIMATE {EC(OE.T)[VAL]}               MISSING_STATS  NO
18093103661535947380 USABLE     YES T                                SINGLE TABLE CARDINALITY
MISESTIMATE {EC(OE.T)[VAL]}               MISSING_STATS  NO

SQL>  BEGIN
  2    FOR rec IN
  3    (SELECT d.directive_id AS did
  4    FROM dba_sql_plan_directives d
  5    JOIN dba_sql_plan_dir_objects o
  6    ON (d.directive_id  =o.directive_id)
  7    WHERE o.owner       ='OE'
  8    AND d.directive_id IN (17154883816957472097,18093103661535947380)
  9    )
 10    LOOP
 11      DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE ( rec.did, 'ENABLED','NO');
 12      DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE ( rec.did, 'AUTO_DROP','NO');
 13    END LOOP;
 14 END;
 15 /

PL/SQL procedure successfully completed.
```

It is very important to set both ENABLED and AUTO_DROP attributes to NO, otherwise the SQL Plan Directives will be dropped automatically after 53 weeks and they will be created again at the next misestimate!

GOOD TO KNOW

There are a few tips that might help troubleshooting the adaptive features; I will try to list a few:

- Very long parse times due to Adaptive Dynamic Sampling: when this problem occurs, you may troubleshoot the problem when the parse is still happening. In this case, the statement is loaded in the SQL Area and its sql_id is visible in v$sqlstats. However, because the parse is still in progress, there are still no cursors (nor plans):

```
SQL> select sql_id,  executions, loads, cpu_time from v$sqlstats where sql_id='auyf8px9ywc6j';

SQL_ID        EXECUTIONS      LOADS   CPU_TIME
------------- ---------- ---------- ----------
auyf8px9ywc6j          0         11          0

SQL> select sql_id,  child_number from v$sql where sql_id='auyf8px9ywc6j';

no rows selected

SQL> select * from table (dbms_xplan.display_cursor('auyf8px9ywc6j',0, 'ALL +NOTE'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
SQL_ID  auyf8px9ywc6j, child number 0

...

NOTE: cannot fetch plan for SQL_ID: auyf8px9ywc6j, CHILD_NUMBER: 0
      Please verify value of SQL_ID and CHILD_NUMBER;
```

```
It could also be that the plan is no longer in cursor cache (check v$sql_plan)
```

- Beware of parses involving SQL Plan Directives and SQL Plan Baselines. When a statement has an existing baseline, the Query Optimizer will parse it anyway: the parse will eventually use a SQL Plan Directive and thus run Adaptive Dynamic Sampling queries. But when the plan is ready, it will be put into the plan history as non-verified; the verified plan(s) will be executed instead. When looking at the cursor, there is no evidence that Adaptive Dynamic Sampling happened. This is not a problem but keep it in mind when debugging.
- When the SQL Plan Directives have been created, tested and disabled on a non-production database, they should be exported and imported back into the production database, so you can avoid the reiteration of disabling them again and, most important, you can prevent performance problems.

IN 12.2 THINGS WILL CHANGE

Note: at the time of writing, release 12.2 is available on the Oracle Cloud only. The information I will show is true for the Oracle Cloud release and should not change for the GA version; however, it is not guaranteed.

- With the next release, the SQL Plan Directives will not trigger Adaptive Dynamic Sampling unless explicitly enabled. The creation of SQL Plan Directives will still be active by default.
- Statistics Feedback will be less aggressive in creating new SQL Plan Directives, specifically Statistics Feedback will not take place for join misestimates.
- The parameter *optimizer_adaptive_features* will be split in two different parameters:
  - *optimizer_adaptive_plans* (by default it will be true)
  - *optimizer_adaptive_statistics* (by default it will be false)

  Those two new parameters will allow activating only one part of the features, giving better control to the DBAs.
- There will be a new property for DBMS_STATS (AUTO_STATS_EXTENSION) that will allow disabling the automatic creation of extended statistics (currently this new property is also available in 12.1.0.2 by installing the patch 21171382). The creation of the extended statistics will be disabled by default.

It is possible to guess, by looking at the new features of 12.2, that the 12.1.0.2 will behave similarly by applying the patch 21171382 (no automatic creation of extended statistics) and setting *_optimizer_dsdir_usage_control* to 0.
It will be clearer as soon as the 12.2 will be GA.

LINKS AND CREDITS

- https://blogs.oracle.com/optimizer/entry/optimizer_adaptive_features_in_the
- https://antognini.ch/2016/10/adaptive-query-optimization-configuration-parameters-preferences-and-fix-controls/
- http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-12c-1963236.pdf
- http://www.slideshare.net/pachot/soug-2014-sqlplandirectives
- http://www.itoug.it/wp-content/uploads/2016/05/AdaptiveDynamicSampling_OTNMi_2016.pdf
- http://blog.dbi-services.com/matching-sql-plan-directives-and-queries-using-it/
- https://community.oracle.com/docs/DOC-918264
- https://blogs.oracle.com/optimizer/entry/dynamic_sampling_and_its_impact_on_the_optimizer
- http://www.toadworld.com/platforms/oracle/w/wiki/11453.spd-sql-plan-directives-in-12c-part-i