Utilizing new CBO features after upgrade to Oracle 12c – Practical example

Jože Senegačnik, Oracle ACE Director, Member of OakTable
DbProf d.o.o.
Ljubljana, Slovenia

Keywords
Oracle 12c, Upgrade, Optimizer

Introduction
The purpose of this paper is to explain what can one expect from the new optimizer features when upgrading from any previous release of Oracle database to the latest 12c (12.1.0.x) version.

Let us start with a brief history of the optimizer development. Oracle introduced so called "Cost Based Optimizer" (CBO) many years ago in version 7. Of course the first version of the CBO was practically unusable. However, the CBO was then improved in each subsequent version of database. Before it actually became usable in version 8i and especially 9i, the SQL statements execution plans were prepared by so called RULE BASED OPTIMIZER, which was using predefined rules for optimization and knew nothing about the actual data, size of the tables, how many distinct values are in column etc. In the latest releases of the database like 10gR2, 11gR1 and 11gR2, optimizer was significantly improved especially in terms of so called "logical" optimization. Logical optimization is actually trying to apply different transformation methods on the text of the SQL statement in order to significantly improve the performance of the statement and of course return the same results. Since 10gR2 majority of these transformations are based on the calculation of the cost of execution and are therefore applied if their cost is lower than any other possible way of execution.

Although Oracle is constantly improving the CBO there are still many problems with it in terms of preparing optimal execution plan. The root problem for preparing a suboptimal execution plan is lack of optimizer's "knowledge" about actual data or as we usually name this "object statistics". In recent versions Oracle is constantly improving statistics gathering methods.

In version 12c Oracle introduced Adaptive Query Optimization which introduces self-learning mechanism based on previous sub-optimal executions. However, at least one sub-optimal execution must be performed that the CBO is able to apply corrective mechanism and prepare a better execution plan.

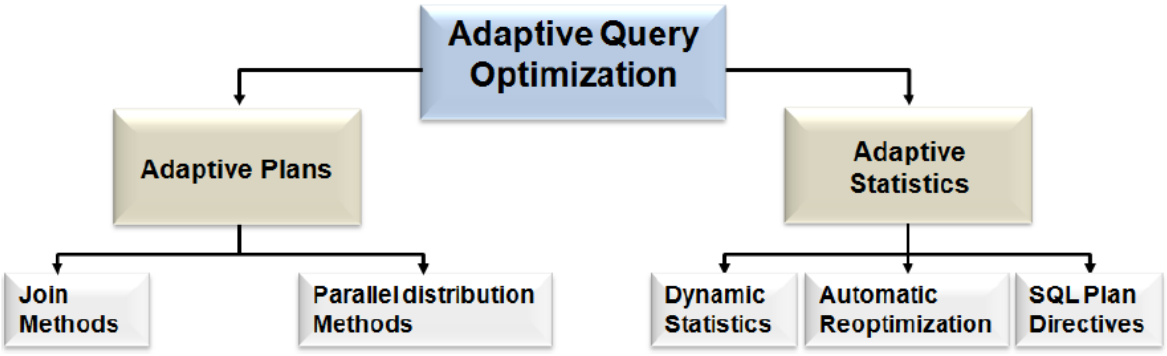

Figure 1: Adaptive Query Optimization methods

Since all features look great when they are introduced and later on might turn into not so bright features the purpose of my test upgrade from 11.2.0.3 to 12.1.0.1 was to see how these features are working in a real environment. For the test I used one particular database which was quite demanding for the previous upgrades, especially from 10gR2 to 11gR2. The purpose of my testing was to see how will new features of Adaptive Query Optimization resolve problems which were quite tough in previous upgrades.

Adaptive Plans

I will just briefly discuss this feature. A well-known problem in many SQL statement execution plans is the choice of selecting the appropriate join method. From the performance perspective the best method when many rows should be joined is hash join (HJ). There is a limitation with HJ that you can join only on equality condition like a.id = b.id. If the join should be done with an inequality condition a.id != b.id a sort merge join (SMJ) or nested loop join (NL) should be used.
The problem of selecting the right join method is coming from the fact that sometimes due to lack of having good object statistics the optimizer can't correctly estimate the number of rows that will be joined. For this purpose optimizer in 12c inserts a new step called "statistics collector" in the execution plan which is there to determine if the number of rows from the first data source is appropriate for a nested loop join which is always the starting join method.

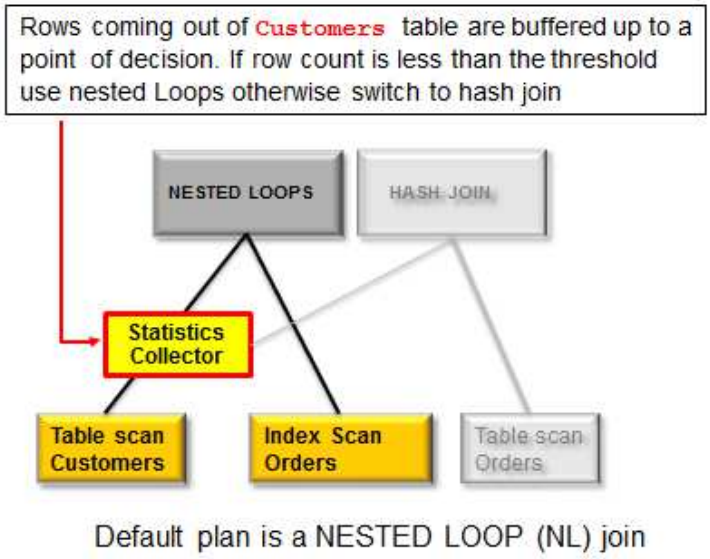

Figure 2: Adaptive plan with default nested loop join operation

When the number of rows is bigger than the threshold determined by the optimizer for this particular join, the execution plan dynamically switches to the alternate join method which is a hash join. In order not to introduce additional work during execution the rows from first data source are buffered until the runtime query engine can determine if the threshold will be reached or not. Figures 2. and 3. Show such execution plan which switches to alternate join method.
One can determine if the plan was actually an adaptive by looking the value in v$sql.IS_RESOLVED_ADAPTIVE_PLAN column which can have the follwoing 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

To detemine what join method was actually used in adaptive plans one can look at the runtime execution plan by using `dbms_xplan.display_cursor('<sql_id>', format=>'+adaptive').`

During my test runs after upgrade I was looking for the critical statements which were optimized with this new feature. The results will come later on in the paper.

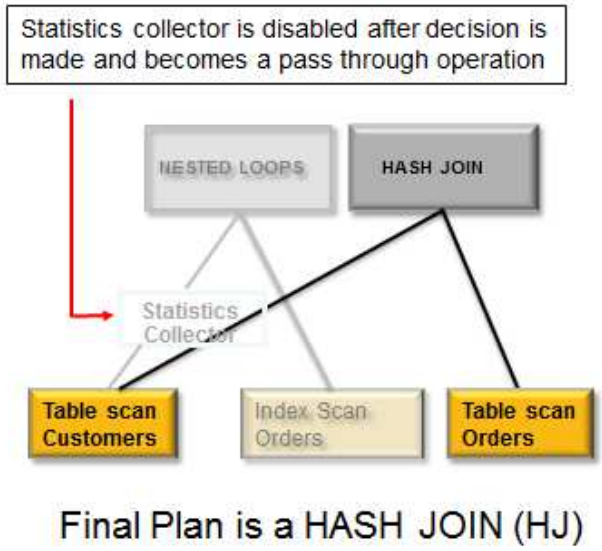

Figure 3: Final adaptive plan with hash join operation

Adaptive Statistics
Next features from which I was expecting quite a lot were the **Automatic Reoptimization** by using execution statistics and SQL Plan Directives.

The optimizer 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 of a SQL statement the optimizer compares original cardinality estimates to the actual cardinalities from the execution. If the estimates differ significantly from actual cardinalities, the optimizer stores the correct estimates for subsequent use.
The optimizer also creates a SQL plan directive in order that other SQL statements could benefit as well. However, if the original estimates are found o.k. then the reoptimization flag is reset and no further monitoring of statistics feedback is performed for this particular SQL statement.

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 and force using dynamic statistics
- They are created for query expressions rather than at a statement or object (table) level in order to be sharable among more SQL statements
- Automatically created by CBO via Automatic Reoptimization.

Testing
The scenario for testing after upgrade was:

- Gather fresh statistics on all objects (new statistics gathering methods and type of histograms were introduced in 12c)
- Enable flashback database
- Run billing process once and export created SQL Plan Directives, SQL Plan Baselines
- Flashback database
- Import baselines and directives
- Run again and monitor performance
- Repeat this several times and change parameters (dynamic_sampling=11), drop SQL Plan Baselines, SQL Profiles

Conclusion

The database was upgraded to a non-multitenant 12c version of the database. Several runs of the most critical billing process were required in order to achieve good results. Actually only few things were running as expected in the first test run.

The conclusions were:

1. Most of the SQL statements performed as before, especially those simple ones.

2. Some statements had worse execution plan, particularly one which was a tough problem already during upgrade from 10g to 11g. This statement had already a SQL Plan Baseline created before and it tuned out that 12c optimizer was not actually using the baseline. On contrary, the existing SQL Plan Baseline was causing more problems than benefit after upgrade. So it might be a good advice that when upgrading to a non-multitenant version of the database, one should disable all exiting SQL Plan Directives and SQL Profiles in order to let the optimizer prepare completely new execution plans.

3. On some complex statements the new features did not do a good job and they had to be optimized manually and the optimal execution plan fixed by baseline or profile.

4. Next important conclusion was the some parts of the code perform better after several runs what could be attributed to the fact that some of the SQL statements got better plans so the optimizer's new reoptimization did it job correctly.

Bottom line conclusion is that one should not expect, that the new features will resolve all problematic sub-optimal execution plans. Some of the problematic statements may get better execution plans, some not. So for all these statements additional tuning should be expected. However, the number of "problematic" statements with sub-optimal execution plans will be for sure lower than in previous versions of database.

Contact:
Jože Snegačnik
DbProf d.o.o.
Smrjene 153
SI-1291 Škofljica

Slovenia

Telefon: +386 41 72 44 61
E-Mail joze.senegacnik@dbprof.com
Internet: www.dbprof.com