Upgrading from Oracle Database 9i to 10g: What to expect from the Optimizer

An Oracle White Paper
February 2008

ORACLE

NOTE:

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Upgrading from Oracle Database 9i to 10g: What to expect from the Optimizer

Upgrading from Oracle Database 9i to 10g: What to expect from the Optimizer

EXECUTIVE SUMMARY

Since its introduction in Oracle 7.0, people have been fascinated and intimidated by the cost-based optimizer (CBO) and the statistics that feed it. It has long been felt that the internals of the CBO were shrouded in mystery and that a degree in wizardry was needed to work with it. One of the most daunting activities a DBA can therefore undertake is upgrading the database to a new version. Having to comprehend all of the new features and to deal with potential plan changes can be overwhelming. This paper aims to dispel the mystery by explaining in detail what to expect from the CBO when you upgrade from Oracle database 9i to 10g.

One key area that has changes in Oracle Database 10g is the Optimizer statistics, specifically the DBMS_STATS package. New types of statistics have been introduced; default parameter values have changed and there is a new automatic statistics gathering job.

INTRODUCTION

The purpose of the Cost Based Optimizer (CBO) is to determine the most efficient execution plan for your queries. It makes these decisions based on the statistical information it has about your data and by leveraging Oracle database features such as hash joins, parallel query, and Oracle Partitioning. After an upgrade, the CBO is expected to generate the same or a better performing execution plan for most SQL statements. Still it is inevitable that the CBO will generate a sub-optimal plan for some SQL statements in the new release compared to the prior release. This paper aims to prepare you to upgrade the Optimizer from Oracle Database 9i to Oracle Database 10g by introducing the new features and what steps you should take before and after the upgrade to minimize plan regressions and help you deal any plan changes that do occur.

The paper is divided into three sections: the first outlines the new features in the Optimizer and statistics areas, the second explains what pre-upgrade steps you need to execute, and the third covers what to expect after the upgrade. This paper is by no means a complete guide for upgrading your Oracle Database 9i. You should refer to the Oracle Database Upgrade Guide for complete details and guidelines for your upgrade.

NEW OPTIMIZER AND STATISTICS FEATURES IN 10G

Init.ora parameters

Several of the initialization parameters that govern Optimizer behavior have new default values in Oracle Database 10g. There is also one new initialization parameter. Below are the details on what parameters have new default values and details on the new parameter.

Optimizer_ mode

The parameter `optimizer_mode` has a new default value of **`ALL_ROWS`** in Oracle database 10g.

This means the Optimizer will no longer operate under RULE mode (RBO) when a table has no statistics. In Oracle database 10g the Optimizer operates under ALL_ROWS mode (CBO) and will use dynamic sampling to get statistics for any tables that do not have statistics and will use CBO. The other possible values are `FIRST_ROWS_1, FIRST_ROWS_10, FIRST_ROWS_100`, and `FIRST_ROWS_1000`. The `CHOOSE, RULE`, and `FIRST_ROWS` modes have been deprecated.

Optimizer_dynamic_sampling

The parameter `optimizer_dynamic_sampling` has a new default value of **`2`** in Oracle Database 10g.

This means dynamic sampling will be applied to all unanalyzed tables. It also means that twice the number of blocks will be use to calculate statistics than were used in Oracle database 9i. The default value for dynamic sampling in 9i was 1.

Optimizer_secure_view_merging

A new init.ora parameter called `optimizer_secure_view_merging` has been introduced in Oracle Database 10g with a default value of **`TRUE`**. The setting of this parameter disables some unsafe view merging capabilities that were present in Oracle Database 9i.

When a SQL statement that refers to a view is parsed, the view referenced in a query is expanded into a separate query block, which represents the view definition, and therefore the result of the view. The view text is then merged into the original SQL statement and the new combined query is optimized as a whole. View merging is a SQL transformation that is completely internal to the Optimizer and transparent to the end user.

While merging views does not change the actual result of a query, it can potentially change the internal order of the various operations that have to take place to process the SQL statement. Prior to Oracle Database 10g, some view merging could have taken place where the new ordering could create a potential security

breach. With optimizer_secure_view_merging set to true additional conservative security checks will be applied and will prevent any view merging that could be regarded as such a breach. It is important to understand that while the Optimizer disallows certain view merging operations, these do not have to be actual security breaches but rather that the Optimizer cannot **guarantee** the integrity of a potential view merging – e.g. with some embedded PL/SQL functions - thus it disallows it to begin with.

With the new security checks it's possible that the execution plan for a SQL statement that refers to a view or contains an inline view will change after the migration from Oracle Database 9i to Oracle Database 10g. If you do not have any security concerns with your application or any possibility of a malicious database user, you can disable the additional checks and revert to the old Oracle Database 9i behavior by setting this new initialization parameter optimizer_secure_view_merging to FALSE.

Parameter	9.2 Value	10gR2 Value
Optimizer_mode	CHOOSE	ALL_ROWS
Optimizer_dynamic_sampling	1	2
Optimizer_secure_view_merging	N/A	TRUE

Table 1 Summary of init.ora parameter changes between 9i and 10g

Changes to the DBMS_STATS package

The ANALYZE command has been officially obsolete for gathering statistics. Use the DBMS_STATS package instead.

In Oracle 8i a new PL/SQL package, called DBMS_STATS was introduced to gather and manage optimizer statistics. DBMS_STATS is Oracle's preferred method for gathering statistics and succeeds the ANALYZE command for collecting statistics. The DBMS_STATS package has been extended in Oracle Database 10g to accommodate new types of statistics and monitoring data that can now be collected. Changes have also been made to the default value for several of the parameters used in the gather statistics procedures. There is also a new automatic statistics gathering job that is on by default in 10g.

New subprograms in the DBMS_STATS package

System statistics

In Oracle Database 9i system statistics were introduce to enable the CBO to effectively cost each operation in an execution plan, by using information about the actual system hardware executing the statement, such as CPU speed and IO performance. However, if system statistics were not gathered in 9i the CBO would revert back to the costing model present in Oracle Database 8i.

In Oracle Database 10g the use of systems statistics is enabled by default and system statistics are automatically initialized with heuristic default values; these values do not represent your actual system. When you gather system statistics in Oracle Database 10g they will override these initial values. To gather system statistics you can use `DBMS_STATS.GATHER_SYSTEM_STATS` during your peak workload time window.

At the beginning of the peak workload window execute the following command:

```
BEGIN
DBMS_STATS.GATHER_SYSTEM_STATS('START');
END;
/
```

At the end of the peak workload window execute the following command:

```
BEGIN
DBMS_STATS.GATHER_SYSTEM_STATS('END');
END;
/
```

Oracle recommends gathering system statistics during a representative workload, ideally at peak workload time. You will only have to gather system statistics once. System statistics are not automatically collected as part of new statistics gather job (see the automatic statistics gathering job section below for more details).

Statistics on Dictionary Tables

Since the default value for `optimizer_mode` in Oracle Database 10g forces the use of the CBO, all tables in the database need to have statistics including all of the dictionary tables (tables owned by 'sys' and reside in the system tablespace). During the upgrade process Oracle will automatically gathers statistics on the dictionary tables. Appendix C of the Oracle® Database Upgrade Guide provides scripts that collect optimizer statistics for dictionary objects. By running these scripts prior to performing the actual database upgrade, you can decrease the amount of downtime incurred during the database upgrade.

Statistics on the dictionary tables will be maintained via the automatic statistics gathering job run during the nightly maintance window. If you choose to switch off the automatic statistics gathering job for your main application schema consider leaving it on for the dictionary tables. You can do this by changing the value of `AUTOSTATS_TARGET` to `ORACLE` instead of `AUTO` using the procedure `DBMS_STATS.SET_PARAM`.

```
BEGIN
DBMS_STATS.SET_PARAM(AUTOSTATS_TARGET,'ORACLE');
END;
/
```

Statistics on Fixed Objects

You will also need to gather statistics on dynamic performance tables (fixed objects) these are the X$ tables on which the V$ view (V$SQL etc.) are built. Fixed objects now need statistics due to the new default value for optimizer_mode. It's

important to gather statistics on the fixed objects as they are often queried to supply information to Statspack and the new Automatic Workload Repository (AWR) in Oracle Database 10g and you need to give the CBO accurate statistics for these objects. You only need to gather fixed objects statistics once for a representative workload and they are not updated by the automatic statistics gathering job. You can collect statistics on fixed objects using `DBMS_STATS.GATHER_FIXED_OBJECTS_STATS`.

```
BEGIN
DBMS_STATS.GATHER_FIXED_OBJECTS_STATS;
END;
/
```

Restoring Statistics

In Oracle Database 10g when you gather statistics using `DBMS_STATS`, the original statistics are automatically kept as a backup in dictionary tables and can be easily restored by running `DBMS_STATS.RESTORE_TABLE_STATS` if the newly gathered statistics prove to be suboptimal.

The example below restores the statistics for the table `SALES` back to what they were yesterday and automatically invalidates all of the cursors referencing the `SALES` table. We want to invalidate all of the cursors because we are restoring yesterday's statistics since today's statistics gave us an unacceptable plan. The value of the no_invalidate parameter determines if the cursors referencing the table will be invalidated or not.

```
BEGIN
DBMS_STATS.RESTORE_TABLE_STATS('SH','SALES',SYSTIMESTAMP-
1,false, false);
END;
/
```

Comparing Statistics

When it comes to deploying a new application or application module it is standard practice to test and tune the application in a test environment before it goes production. However, even with testing it's possible that SQL statements in the application will have different execution plans in production then they did on the test system. One of the key reasons an execution plan can differ from one system to another (from test and production) is because the optimizer statistics on each system are different. In Oracle Database 10g Release 2, [1]the `DIFF_TABLE_STATS_*` functions can be used to compare statistics for a table from two different sources. The statistics can be from:

- A user statistics table and current statistics in the dictionary
- A single user statistics table containing two sets of statistics that can be identified using statids

[1] Only available in Oracle Database 10g Release 2 patch set 3 10.2.0.4.

- Two different user statistics tables
- Two points in history

The function also compares the statistics of the dependent objects (indexes, columns, partitions). The function displays statistics for the object(s) from both sources if the difference between the statistics exceeds a certain threshold (%). The threshold can be specified as an argument to the function; the default value is 10%. The statistics corresponding to the first source will be used as the basis for computing the differential percentage.

In the example below we compare the current dictionary statistics for the table `EMP` with the statistics for `EMP` in the stats table tab1; the SQL statement will generate a report-like output on the screen.

```
SQL> select
dbms_stats.diff_table_stats_in_stattab('scott','emp','tab1')
from dual;
```

More examples of how to use the new `DIFF_TABLE_STATS_*` functions and an example of it's output can be found in Appendix B.

New default parameter values for DBMS_STATS.GATHER_*_STATS

The default value for a number of the parameters used in the `DBMS_STATS` gathers statistics subprograms have changed in Oracle Database 10g. Table 2 below highlights these changes.

Parameter	9.2 Value	10gR2 Value
METHOD_OPT	FOR ALL COLUMNS SIZE 1	FOR ALL COLUMNS SIZE AUTO
ESTIMATE_PRECENT	100 (Compute)	DBMS_STATS.AUTO_SAMPLE_SIZE
GRANULARITY	DEFAULT(Table&Partition)	AUTO
CASCADE	FALSE	DBMS_STATS.AUTO_CASCADE
NO_VALIDATE	FALSE	DBMS_STATS.AUTO_INVALIDATE

Table 2 Default values for parameters used in DBMS_STATS

The **METHOD_OPT** parameter controls the creation of histograms during statistics collection. With the new default value of `FOR ALL COLUMNS SIZE AUTO`, Oracle automatically determines which columns require histograms and the number of buckets that will be used based on the column's usage statistics. A column is a candidate for a histogram if it has been seen in a where clause predicate e.g. an equality, range, LIKE, etc. Oracle will verify whether the column is skewed before creating a histogram, for example a unique column will not have a histogram created on it.

The **ESTIMATE_PERCENT** parameter determines the percentage of rows used to calculate the statistics. In Oracle Database 9i the default percentage was 100% or all of the rows in the table. However, in Oracle Database 10g statistics are gathered

using a sampling method. Oracle automatically determines the appropriate sample size for every table in order to get accurate statistics.

The **GRANULARITY** parameter dictates at which level statistics will be gathered. The possible levels are table (global), partition, or subpartition. With the new default setting of `AUTO` Oracle will determine the granularity based on the objects partitioning type.

The **CASCADE** parameter determines whether or not statistics are gathered for the indexes on a table. In Oracle Database 10g, this parameter is set to `DBMS_STATS.AUTO_CASCADE` by default, which means Oracle will determine whether index statistics need to be collected or not.

In Oracle Database 9i the **NO_INVALIDATE** parameter determined if the dependent cursors will be invalidated immediately after statistics are gathered or not. With the new setting of `DBMS_STATS.AUTO_INVALIDATE` in Oracle database 10g, cursors that have already been parsed will not be invalidated immediately. They will continue to use the plan that was obtained using the original statistics until Oracle decides to invalidate the dependent cursors. The invalidations will happen gradually over time to ensure there is no performance impact on the shared pool as there could be if all of the dependent cursors were hard parsed all at once.

Changing the default parameter values for DBMS_STATS

To change the default value for any of the parameters used by the `DBMS_STATS` subprograms, use the `DBMS_STATS.SET_PARAM` procedure, e.g:

```
BEGIN
DBMS_STATS.SET_PARAM('CASCADE','DBMS_STATS.AUTO_CASCADE');
END;
/
```

Histograms are probably the statistics that help and hinder the CBO the most.

Histograms

With the new default setting for the `METHOD_OPT` parameter Oracle will automatically determine which columns should have histograms created on them when gathering statistics.

Oracle bases the decision to create a histogram on internal information recorded about column usage, such as what number and type of `WHERE` clause predicates (= < > Like etc.) were used for each column. Histograms allow the Optimizer to better estimate the cardinality of a particular column after applying all where clause predicates. A histogram is a series of buckets, where the number of values occurring within a range is tracked in these buckets. The range for each bucket is established during statistics collection. Histograms are potentially useful when:

1. The column is used in an equality predicate or an equi-join predicate AND there are frequency skews in the column data.
2. The column is used in a range or like predicate AND there are either or both frequency skews or range skews in the column.

Information about existing histograms can be viewed in *_TAB_HISTOGRAMS and *_TAB_COL_STATISTICS.

```
SELECT S.TABLE_NAME, S.COLUMN_NAME, S.HISTOGRAM
FROM USER_TAB_COL_STATISTICS S;
```

Histograms and Bind Peeking

When optimizing a SQL statement that contains bind variables in the WHERE clause the Optimizer peeks at the values of these bind variables on the first execution (during hard parse). The Optimizer determines the execution plan based on the initial bind values. On subsequent executions of the query, no peeking takes place (no hard parse happens), so the original execution plan will be used by all future executions, even if the value of the bind variables change. The presence of a histogram on the column used in the expression with the bind variable may cause a different execution plan to be generated for the statement depending on the initial value of the bind variabl being peeked, so the execution plan could vary depending on the values of the bind variables on its first invocation. This issue may surface in light of the change in the default behavior in DBMS_STATS (see the section on the **METHOD_OPT** parameter in New Default Parameter Values for DBMS_STATS) If this change causes performance problems then you can regather statistics on this table without histograms or change the value of **METHOD_OPT** parameter.

Automatic statistics gathering job

Oracle will automatically collect statistics for all database objects, which are missing statistics or have stale statistics by running an Oracle Scheduler job (GATHER_STATS_JOB) during a predefined maintenance window (10 pm to 6 am weekdays and all day at the weekends). You can adjust the predefined maintenance windows to a time suitable to your database environment using the DBMS_SCHEDULER.SET_ATTRIBUTE procedure. For example, the following statement moves the WEEKNIGHT_WINDOW to midnight through to 8 a.m. every weekday morning:

```
EXECUTE DBMS_SCHEDULER.SET_ATTRIBUTE(
'WEEKNIGHT_WINDOW',
'repeat_interval',
'freq=daily;byday=MON, TUE, WED, THU,
FRI;byhour=0;byminute=0;bysecond=0');
```

This job gathers optimizer statistics by calling the internal procedure DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC. This procedure operates in a very similar fashion to the DBMS_STATS.GATHER_DATABASE_STATS

procedure using the GATHER AUTO option. The primary difference is that Oracle internally prioritizes the database objects that require statistics, so that those objects, which most need updated statistics, are processed first. You can verify that the automatic statistics gathering job exists by viewing the DBA_SCHEDULER_JOBS view:

```
SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME =
'GATHER_STATS_JOB';
```

Statistics on a table are considered stale when more than 10% of the rows are changed (total # of inserts, deletes, updates) in the table. Oracle monitors the DML activity for all objects and records it in the SGA. The monitoring information is periodically flushed to disk and is exposed in the *_tab_modifications view.

```
SELECT TABLE_NAME, INSERTS, UPDATES, DELETES
FROM USER_TAB_MODIFICATIONS;
```

It is also possible to manually flush this data by calling the procedure DBMS_STATS.FLUSH_MONITORING_INFO.

The automatic statistics gathering job uses the default parameter values for the DBMS_STATS procedures. If you wish to change these defalut values you can use the DBMS_STATS.SET_PARAM procedure. Remember these values will be used for all schemas including 'SYS'. To change the 'ESTIMATE_PERCENT' you can use

```
BEGIN
DBMS_STATS.SET_PARAM('ESTIMATE_PERCENT','5');
END;
/
```

If you already have a well established statistics gather procedure or if for some other reason you need to disable automatic statistics gathering altogether, the most direct approach is to disable the GATHER_STATS_JOB as follows:

```
BEGIN
DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');
END;
/
```

If you choose to switch off the automatic statistics gathering job for your main application schema consider leaving it on for the dictionary tables. You can do this by changing the value of AUTOSTATS_TARGET to ORACLE instead of AUTO using DBMS_STATS.SET_PARAM.

Parallel execution plans

In Oracle Database 10g the parallel execution model for queries has changed from a slave SQL model to a parallel single cursor (PSC) model. Instead of having the

query coordinator (QC) build different SQL statements for each individual block of parallel operations in a query plan, and having each slave set parsing and executing his own cursor, we now build and compile just one cursor that contains all the information required for the entire parallel statement. All of the slaves now share this single cursor[2].

The new model of a single parallel plan means that more SQL statements can now be parallelized. If your Oracle Database 9i environment uses parallel query because you had explicitly set a parallel degree on the one or more tables, then its possible you will see an increase in the number of queries being executed in parallel on your system in 10g. If you only enable parallel query through the use of hints in certain SQL statement then you will not see any change in the number of statements being executed in parallel. In order to ensure you have enough parallel slave resources in 10g, you should make a note of the maximum number of parallel slaves being used on your systems during peek times across the course of a week. You will need to ensure that you set the initialization parameter `PARALLEL_MAX_SERVERS` to be greater than your maximum value used in Oracle Database 9i. Note also that the default value for PARALLEL_MAX_SERVERS has changed from Oracle Database 9i to Oracle Database 10g; it has gone from a fixed value of 10 to an automatically derived value. You have to manually set this parameter to 10 to revert back to the default behavior in Oracle Database 9i.

Cost Based Transformations

Oracle transforms SQL statements using a variety of sophisticated techniques during query optimization. The purpose of this phase of query optimization is to transform the original SQL statement into a semantically equivalent SQL statement that can be processed more efficiently. In Oracle Database 9i the following Optimizer transformations were heuristic based.

- Complex View Merging
- Subquery Unnesting
- Join Predicate Push Down

This means the transformations were applied to SQL statements based on the structural properties of the query: e.g., number of tables, availability of indexes, types of joins and filters, presence of grouping clauses, etc.; however, the selectivity, cardinality, join order, and other related costs of various database operations, were not taken into account.

In Oracle Database 10gR2 a new general framework for cost-based query transformations was introduced, so that the three transformations mentioned above became cost-based transformations. In cost-based transformation, queries

2 For a detailed description of Oracle's parallel execution capabilities, see the Oracle documentation, namely the Data Warehousing Guide

are rewritten or transformed into various forms and their costs are estimated. This process is repeated multiple times applying a new set of transformations each time. The optimizer then selects the best execution plan based on the combination of one or more transformations with the lowest cost.

The types of queries that are affected by the above three transformations can be characterized by views with group-by clause or distinct key word (known in Oracle as complex views), by subqueries with multiple tables or with aggregate functions, and by multiple-table outer-joined views or views with UNION/UNION ALL.

Extensions to the DBMS_XPLAN package

Plan output from additional sources

An explain plan displays the execution plan chosen by the Optimizer for a given SQL statement. Generating and displaying the execution plan of a SQL statement is a common task for most DBAs. In Oracle Database 9i the PL/SQL package DBMS_XPLAN was introduced to provide an easier way to format the output of the EXPLAIN PLAN command. In Oracle Database 10g the DBMS_XPLAN package has been extended to enable you to display execution plans from three additional sources:

1. V$SQL_PLAN
2. Automatic Workload Repository (AWR)
3. SQL Tuning Set (STS)

The V$SQL_PLAN dictionary view introduced in Oracle 9i shows the execution plan for a SQL statement that has been compiled into a cursor in the cursor cache. The advantage of looking at the execution plan from V$SQL_PLAN, rather than from the EXPLAIN PLAN command, is that the value of any bind variable(s) are taken into account during the plan generation process in V$SQL_PLAN but not in the EXPLAIN PLAN command.

You can display an execution plan from V$SQL_PLAN by providing the SQL_ID to the new DBMS_XPLAN.DISPLAY_CURSOR function or by running the following query with the leading edge of your SQL text.

```
SELECY PLAN_TABLE_OUTPUT
FROM V$SQL s,
TABLE(DBMS_XPLAN.DISPLAY_CURSOR(s.SQL_ID, s.CHILD_NUMBER,'BASIC'))t
 WHERE s.SQL_TEXT LIKE 'select * from emp%';
```

The Automatic Workload Repository (AWR) was introduced in Oracle Database 10g; it collects, processes, and maintains performance statistics for the database and stores them in the database. It gathers and store performance information similar to what is stored in the Statspack schema in 9i. Using the new

DBMS_XPLAN.DISPLAY_AWR function you can display an execution plan stored in AWR based on its SQL_ID.

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY_AWR('gm9t6ycmb1yu6'));
```

A SQL Tuning Set (STS) is a database object that includes one or more SQL statements along with their execution statistics and execution context. They were introduced in Oracle Database 10g as part of the new manageability framework. They can be used as input to the new SQL Tuning Advisor and are transportable across databases. It's possible to display the execution plan for a SQL statement stored in a STS by supplying its SQL_ID to the new DBMS_XPLAN.DISPLAY_SQLSET function.

```
SELECT * FROM table
(DBMS_XPLAN.DISPLAY_SQLSET('gm9t6ycmb1yu6'));
```

Extended and more granular plan output

Each of the DBMS_XPLAN.DISPLAY* functions takes a format parameter, the valid values are basic, typical, all. The format parameter controls the amount of detail displayed in the plan output, from a high level summary that only includes the execution plan (format=>'basic'), to finer grained detail (format=>'all'). The default is 'typical'. In Oracle 10g, additional options can also be passed with the format parameter to selectively display the detailed information, such as predicates used and the value of the bind variables used to generate the execution plan. Take for example a simple SQL statement that run against the SH sample schema.

```
SQL>  SELECT prod_category, avg(amount_sold)
  2   FROM sales s, products p
  3   WHERE p.prod_id = s.prod_id
  4   AND prod_category != :pcat
  5  GROUP BY prod_category;
```

After running this statement you can look at the execution plan in V$SQL_PLAN and see what the value of the bind variable :pcat was by running the following query

```
SQL> select plan_table_output from

table(dbms_xplan.display_cursor(null,null,'typical +peeked_binds'));


PLAN_TABLE_OUTPUT
------------------------------------------------------------
SQL_ID  4tatrz7dcp9kb, child number 0
```

```
------------------------------------

select prod_category, avg(amount_sold) from sales s, products p where p. prod_id = s.
prod_id  and prod_category != :pcat

group by prod_category


Plan hash value

| Id| Operation             | Name    | Rows| Bytes| Cost(%CPU)| Time     | Pstart| Pstop
|-------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT      |         |     |      | 555 (100)|          |       |       |
| 1 |  HASH GROUP BY        |         |   1 |   30 |  555 (13)| 00:00:07 |       |       |
|*2 |   HASH JOIN           |         | 787K|   22M|  508  (5)| 00:00:07 |       |       |
|*3 |    TABLE ACCESS FULL  | PRODUCTS|  62 | 1302 |    3  (0)| 00:00:01 |       |       |
| 4 |    PARTITION RANGE ALL|         | 918K| 8075K|  498  (4)| 00:00:06 |     1 |    28 |
| 5 |     TABLE ACCESS FULL | SALES   | 918K| 8075K|  498  (4)| 00:00:06 |     1 |    28 |
-------------------------------------------------------------------------------------------

Peeked Binds (identified by position):

--------------------------------------

   1 - :PCAT (VARCHAR2(30), CSID=178): 'Women'


Predicate Information (identified by operation id):

---------------------------------------------------

   2 - access("P"."PROD_ID"="S"."PROD_ID")

   3 - filter("PROD_CATEGORY"<>:PCAT)
```

SQL Test Case Builder[3]

If you ever need to contact Oracle support about a SQL issue obtaining a reproducible test case is the single most important factor to ensure a speedy resolution. This can also be the longest and most painful step for a customer. A new tool called the SQL Test Case Builder was introduced to help customers to gather as much information as possible relating to a SQL incident and package it up ready to send to Oracle. This package of information will allow a developer at Oracle to reproduce the problem standalone on a different Oracle instance and resolve the issue sooner. You can access the SQL Test Case Builder through the PL/SQL package `DBMS_SQLDIAG`. There are two procedures; `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE`, which enables you to export a SQL test case for a given SQL statement into a given directory and `DBMS_SQLDIAG.IMPORT_SQL_TESTCASE`, which enables you to import a given SQL test case from a given directory.

3 This functionality is generally available beginning with Oracle Database 10g patchset 3, a.k.a. 10.2.0.4

Optimizer Features Enable

The initialization parameter `OPTIMIZER_FEATURES_ENABLE` acts as an umbrella parameter that can be used to enable or disable a series of optimizer-related features for a given release. After you upgrade to Oracle Database 10g, if you wanted to revert back to 9.2 optimizer behavior you can set `OPTIMIZER_FEATURES_ENABLE` = 9.2.0

This would set the default value for dynamic sampling back to 1, revert the optimizer costing model back to what it was in 9.2, and change all the cost based transformations back to being heuristic based. However, you should note that it would not change the PL/SQL package `DBMS_STATS` and its default parameter values back to what they were in 9.2, nor will it change the new parallel execution model or remove the new secure view merging feature. Details on how to revert any of these features back to their 9.2 behavior is outline in the above appropriate sections.

PREPARING TO UPGRADE

Undertaking a database upgrade is a daunting task for any DBA. Once the database has been successfully upgraded you must still run the gauntlet of possible database behavior changes. On the top of every DBA's list of possible behavior changes are execution plan changes. In order to easily detect these changes and rectify any execution plans that may have regressed you need to have a very accurate understanding of the execution plans and Optimizer statistics you had before you began the upgrade. You also need to test your applications against the new release before upgrading your production system.

Testing your application

- Ideally your pre-upgrade testing should be conducted on exactly the same hardware you use in your production system. That means, the same CPU brand, memory architecture, O/S release etc.
- Always use a copy of the 'live' data from the production system for testing. Never test with just a small percentage of the real data or with 'hand-crafted' data sets. This can lead to an unrealistic sense of security and will not prepare you for what changes may occur on production.
- Ensure **all** important queries and reports are tested.
- Check for increases in batch job execution times as it may indicate problems or plan changes in one or more steps in the job.
- Make sure you have comparable test results from your current Oracle Database releases (elapsed times, execution plans, Statspack reports, system statistics). This may mean having to run the tests on both the current and new release.

Pre-Upgrade Checklist

Before you upgrade your production system to Oracle Database 10g you must collect and save the following pieces of information

1. Gather Instance-wide performance statistics from the Production database (during peak load times) as a baseline. These baselines may be used for future comparison if the need arises. Instance-wide performance statistics include:
    a. Statspack data and reports. Configure Statspack to take level 7 snapshots so you can collect segment statistics and plan information. Schedule Statspack snapshots every hour for at least one week to capture expensive SQL.
    b. OS statistics including CPU, memory and IO (such as sar, vmstat, iostat)
2. Be sure to perform all business critical transactions as well as month-end processes and common ad-hoc queries during the baseline capture.
3. Export the Statspack schema owner, PERFSTAT. Keep the export file as backup.
4. Export a complete set of Optimizer statistics into a statistics table, and export the table as a backup.
5. Make a backup of your init.ora file.
6. Create Stored Outlines for all key SQL statements as a backup mechanism to ensure you have a way to revert back to the 9i execution plan should the plan regress in 10g. Key statements include the current Top SQL statements, and any important SQL statements in the application. Detailed instruction on how to do this can be found in Appendix A.

AFTER THE UPGRADE

Once you have successfully upgraded to Oracle Database 10g and your application is running, you will have to monitor your environment careful to ensure you do not encounter any performance issues or plan regressions. The steps below outline what you should do.

Post-Upgrade Checklist

1. Install or upgrade Statspack and set the level to 7. Follow Statspack instructions specially if upgrading it.
2. Schedule Statspack snapshots every hour. This will let Statspack capture expensive SQL in your 10g environment. If you have licensed the Diagnostic Pack then you can use the new AWR reports, which will be automatically captured hourly.
3. Capture OS statistics, which coincide with your Statspack reports.

4. Identify the expensive SQL (top SQL) using Statspack or AWR reports. Compare these SQL statements to the top SQL statements you had prior to the upgrade. If they are not the same you will need to investigate why.

    a. If the pre-upgrade instance is still available, execute the source transaction in both instances (9i and 10g) and compare the execution plans, buffer gets, CPU time, and total elapse times.

    b. If pre-upgrade instance is no longer available, use the export of the PERFSTAT schema (you took as part of the pre-upgrade check list) to find the SQL statement and its execution plan (use script sprepsql.sql)

5. Determine root cause of sub optimal plans and take corrective action. Corrective action may be in the form of: re-gathering statistics with different parameter settings, use SQL Tuning Advisor, index creation, creation of a SQL Profile, use of Optimizer hints, research of known Bugs, logging an SR, etc.
6. In the short term you can use the 9i Store Outline you capture before the upgrade to revert the execution plan back to what it was in 9i. See Appendix A for details on how to activate a captured Stored Outline.

CONCLUSION

Since the introduction of the Cost Based Optimizer (CBO) in Oracle 7.0, people have been fascinated by it and the statistics that feed it. In Oracle Database 10g the CBO's effectiveness and ease of use have been greatly improved. By outlining in detail the changes made to the CBO and its statistics in this release we hope to remove some of the mystery surrounding them and help make the upgrade process smoother as forewarned is forearmed.

APPENDIX A: STORED OUTLINES CAN PROVIDE PLAN STABILITY

Capturing Stored Outlines for your mission critical SQL statements prior to your upgrade to 10g provides a safety net or fallback method should the execution plan for any of these statements regress.

There are two ways to capture Stored Outlines, you can either manually create one for each SQL statement using the `CREATE OUTLINE` command or let Oracle automatically create a Stored Outline for each SQL statement that is executed. Below are the steps needed to let Oracle automatically create the Stored Outlines for you in your Oracle Database 9i prior to your upgrade to 10g.

Prior to Upgrade Steps

1. You should begin by starting a new session and issuing the following command to switch on the automatic capture of a Stored Outline for each SQL statement that gets parsed from now on until you explicitly turn it off.

   ```
   SQL > alter system set CREATE_STORED_OUTLINES=OLDPLAN;
   ```

   `NOTE:` Ensure that the schemas in which outlines are to be created have the `CREATE ANY OUTLINE` privilege. If they don't not Stored Outlines will actually be captured.

2. Now execute your workload either by running your application or manually issuing SQL statements. NOTE: if you manually issue the SQL statements ensure you use the exact SQL text used by the application, if it uses bind variables you will have to use them too.

3. Once you have executed your critical SQL statements you should turn off the automatic capture by issuing the following command:

   ```
   SQL > alter system set CREATE_STORED_OUTLINES=false;
   ```

4. To confirm you have captured the necessary Stored Outlines issue the following SQL statement.

   ```
   SQL> select name, sql_text, category
       from user_outlines;
   ```

   NOTE: Each Stored Outline should be in the OLDPLAN category.

5. The actual Stored Outlines are stored in the OUTLN schema. Before you upgrade you should export this schema as a backup.

   ```
   exp outln/outln file=soutline.dmp owner=outln rows=y
   ```

After the Upgrade

Once you have upgraded to 10g the captured Stored Outlines will act, as a safety net for you should any of your SQL statements regress. The following steps show you how you can use a Stored Outline to revert the changed execution plan back to the plan you had in your 9i system.

1. Once you have identified a SQL statement whose plan has regressed you will need to find it's corresponding Stored Outline. This is a three step process.

    a. The sql_text for each stored outline is stored as a long and it is not possible to use a `LIKE` predicate on a long column. So we need to create a temporary table to convert the long to a clob.

    ```
    SQL> Create table match_outlines (

         name varchar2(30),

         sql_text clob);
    ```

    b. Then select the name and sql_text columns from the user_outlines view and insert them into the new match_outlines table.

    ```
    SQL> Insert into match_outlines

         Select name, to_lob(sql_text)

         From user_outlines;
    ```

    c. Finally retrieve the name of the corresponding stored outline by searching the sql_text for the leading edge of your regressed SQL statement.

    ```
         SQL> select name

              From match_outlines

              Where sql_text like 'select prod_id%';
    ```

2. Once you identify the stored outline by name, you will need to change its category from `OLDPLAN` to `FIXPLAN`.

    ```
    SQL> alter outline <name> change category to FIXPLAN;
    ```

3. Then set the parameter `USE_STORED_OUTLINES` to `FIXPLAN`.

    ```
     SQL> Alter system set USE_STORED_OUTLINES=FIXPLAN;
    ```

4. You can verify the SQL statement is using the Stored Outline by checking the new notes section on the explain plan or by querying the outline_category column in the V$SQL view.

```
SQL> select sql_text, outline_category

     From   v$SQL

     Where  sql_text like 'select prod_id%';
```

APPENDIX B: THE NEW DIFF_TABLE_STATS_* FUNCTION

In Oracle Database 10.2.0.4 the DBMS_STATS.DIFF_TABLE_STATS_* functions can be used to compare statistics for a table from two different sources. The statistics can be from:

- A user statistics table and the current dictionary statistics
- A single user statistics table containing two sets of statistics that can be identified using different statids
- Two different user statistics tables
- Two different points in history

The functions are defined as

```
DBMS_STATS.DIFF_TABLE_STATS_IN_STATTAB(
ownname         IN  VARCHAR2,
tabname         IN  VARCHAR2,
stattab1        IN  VARCHAR2,
stattab2        IN  VARCHAR2 DEFAULT NULL,
pctthreshold    IN  NUMBER  DEFAULT 10,
statid1         IN  VARCHAR2 DEFAULT NULL,
statid2         IN  VARCHAR2 DEFAULT NULL,
stattab1own     IN  VARCHAR2 DEFAULT NULL,
stattab2own     IN  VARCHAR2 DEFAULT NULL)
RETURN DiffRepTab pipelined;


DBMS_STATS.DIFF_TABLE_STATS_IN_HISTORY(
ownname         IN  VARCHAR2,
tabname           IN  VARCHAR2,
time1                IN  TIMESTAMP WITH TIME ZONE,
time2                IN  TIMESTAMP WITH TIME ZONE DEFAULT NULL,
pctthreshold    IN  NUMBER  DEFAULT 10)
RETURN DiffRepTab pipelined;
```

Below are examples of possible use cases for the diff_table_stats functions.

Comparing statistics found in a user statistics table to those currently in the dictionary for a given table

In this example we compare the statistics in the user statistic table TAB1 with the current dictionary statistics for the table EMP.

```
SQL> select
dbms_stats.diff_table_stats_in_stattab(null,'emp','tab1')

from dual;
```

Comparing two sets of statistics identified by different statids in the same user statistics table

In this example we compare two different sets of statistics, for the table `EMP`, which are stored in the user statistics table `TAB1`. Each set of statistics is identified by a different statid (stats1, stats2)

```
SQL> select
dbms_stats.diff_table_stats_in_stattab(null,'emp','tab1',NULL,1
0,'stats1','stats2')

from dual;
```

Comparing the current statistic for a table with those from a week ago

In this example we compare the current statistics for the table `EMP` with those from a week ago.

```
SQL> select
dbms_stats.diff_table_stats_in_history(null,'emp',SYSTIMESTAMP-
7) from dual;
```

Example of a DBMS_STATS.DIFF_TABLE_STATS_* report

Below is an example of the report that is generated after running an of the DBMS_STATS.DIFF_TABLE_STATS_* functions.

```
DBMS_STATS.DIFF_TABLE_STATS_IN_STATTAB(NULL,'EMP','TAB1')

################################################

STATISTICS DIFFERENCE REPORT FOR:

TABLE         : EMP

OWNER         : SCOTT

SOURCE A      : User statistics table TAB1

              : Statid     :

              : Owner      : SCOTT

SOURCE B      : Current Statistics in dictionary

PCTTHRESHOLD  : 10
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

NO DIFFERENCE IN TABLE / (SUB)PARTITION STATISTICS

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

COLUMN STATISTICS DIFFERENCE:

COL_NAME  SRC NDV  DENSITY HIST NULLS  LEN  MIN   MAX SIZE

DEPTNO          A   3       .333333333 NO          0           3
C10B  C11F  14

                            B   3       .035714285 YES         0
3    C10B  C11F  14
```

REFERENCES

Optimizer

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 13 The Query Optimizer

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 18 Using Plan Stability

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 16 Using Optimizer Hints

Oracle® Database Concepts 10g Release 2 (10.2) Chapter 25 SQL, PL/SQL

Statistics

Oracle® Database PL/SQL Packages and Types Reference 10g Release 2 (10.2) Chapter 103 DBMS_STATS

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 14 Managing Optimizer Statistics

Automatic Statistic Gathering Job

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 14 Managing Optimizer Statistics

Oracle® Database Administrator's Guide 10g Release 2 (10.2) Chapter 23 Managing Automatic System Tasks Using the Maintenance Window.

Parallel Query

Oracle® Database Data Warehousing Guide 10g Release 2 (10.2) Chapter 25 Using Parallel Execution

Oracle® Database Performance Tuning Guide 10g Release 2 (10.2) Chapter 11 SQL Tuning Overview.

Initialization Parameters

Oracle® Database Reference 10g Release 2 (10.2) Chapter 1 Initialization Parameters

Upgrading

Oracle® Database Upgrade Guide 10g Release 2 (10.2)

Upgrading from Oracle Database 9i to 10g: What to expect from the Optimizer
February 2008
Author: Maria Colgan

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com