SQL Plan Management in Oracle Database 11g

An Oracle White Paper
June 2007

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.

SQL Plan Management in Oracle Database 11g

INTRODUCTION

The performance of any database application heavily relies on its query execution. While the Oracle optimizer is perfectly suited to evaluate the best possible plan without any user intervention, a SQL statement's execution plan can change unexpectedly, for a variety of reasons including: re-gathering optimizer statistics, changing optimizer parameters or schema/metadata definitions. Not being able to guarantee a plan will change always for the better has led some customers to freeze their execution plans (Stored Outlines) or lock their optimizer statistics. However, doing so prevents such environments from ever taking advantage of new optimizer functionality or access paths, which would improve the SQL statements performance. Being able to preserve the current execution plan amidst environment changes and allowing changes only for the better would be the ultimate solution.

Oracle Database 11g is the first database on the market capable of solving this challenge. SQL Plan Management (SPM) provides a framework for completely transparent controlled execution plan evolution. With SPM the optimizer automatically manages execution plans and ensures only known or verified plans are used. When a new plan is found for a SQL statement it will not be used until it has been verified by the database to have comparable or better performance than the current plan.

SQL PLAN MANAGEMENT

Guaranteed plan stability and controlled plan evolution.

SQL plan management (SPM) ensures that runtime performance will never degrade due to the change of an execution plan. To guarantee this, only accepted (trusted) execution plans will be used; any plan evolution will be tracked and evaluated at a later point in time and only be accepted as verified if the new plan causes no runtime change or an improvement of the runtime. The SQL Plan Management has three main components:

1. SQL plan baseline capture:
   Create **SQL plan baselines** that represents accepted (trusted) execution plans for all relevant SQL statements. The SQL plan baselines are stored in a plan history in the **SQL Management Base** in the SYSAUX tablespace.
2. SQL plan baseline selection:
   Ensure that only accepted execution plans are used for statements with a

SQL plan baseline and track all new execution plans in the **plan history** for a statement. The plan history consists of accepted and unaccepted plans. An unaccepted plan can be unverified (newly found but not verified) or rejected (verified but not found to be performant).

3. SQL plan baseline evolution:
   Evaluate all unverified execution plans for a given statement in the plan history to become either accepted or rejected.

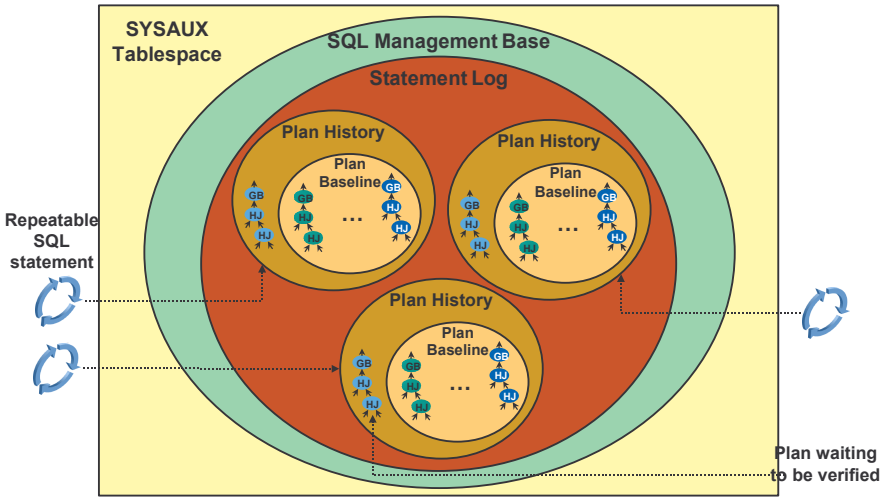

Figure 0 SQL Management base, consisting of the statement log and plan histories for repeatable SQL Statements.

SQL plan baseline capture

Capture plans 'on the fly' or bulk load SPM with plans from the cursor cache, a SQL Tuning Set or import plans from another system.

For SPM to work you must first seed the SQL Management Base with the current cost-based execution plans, which will become the SQL plan baseline for each statement. There are two different ways to populate a SQL Management Base:

- Automatic capture of execution plans
- Bulk load execution plans

Automatic plan capture – "on the fly"

Automatic plan capture can be switched on by setting the init.ora parameter *OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES* to *TRUE* (the default value is *FALSE*). With automatic plan capture enabled, the SPM repository will be automatically populated for any repeatable SQL statement. To identify repeatable SQL statements, the optimizer will log the identity (SQL Signature) of each SQL statement into a statement log the first time it is compiled. If the SQL statement is processed again (executed or compiled) the presence of its identity in the statement

log will signify it to be a repeatable statement. A SQL plan history will be created for the statement, which will include information used by the optimizer to reproduce the execution plan, such as the SQL text, outline, bind variables, and compilation environment. The current cost-based plan will be added as the first SQL plan baseline and this plan will be marked as accepted. Only accepted plans will be used; if some time in the future a new plan is found for this SQL statement, the execution plan will be added to the plan history and will be marked for verification. It will only be marked accepted if its performance is better than that of a plan chosen from current SQL plan baseline.

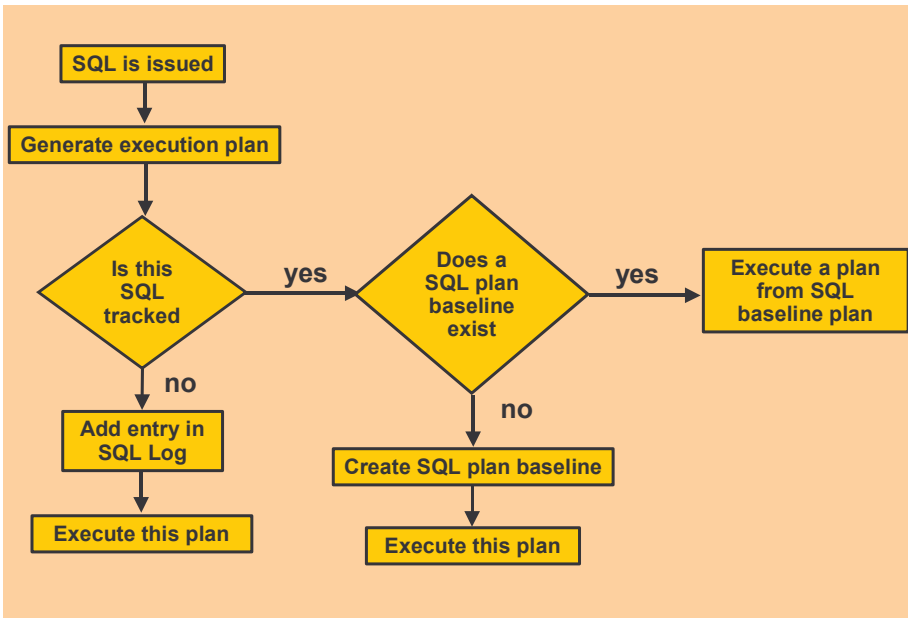

Figure 1 SQL Management base, consisting of the statement log and plan histories for repeatable SQL Statements.

Bulk Load

Bulk loading of execution plans is especially useful when a database is being upgraded from a previous version to Oracle Database 11g or when a new application is being deployed. Bulk loading can be done in conjunction with or instead of automatic plan capture. Execution plans that are bulk loaded are automatically accepted to create new SQL plan baselines or to add to an existing one. The SQL Management Base can be bulk loaded using three different techniques:

1. Populate the execution plans for a given SQL Tuning Set (STS)
2. Use the execution plans currently in the Cursor Cache

3. Unpack existing SQL plan baselines from a staging table

From a SQL Tuning Set (STS)

You can capture the plans for a (critical) SQL workload into a SQL Tuning Set (STS), then load these plans into the SQL Management Base as SQL plan baselines using the PL/SQL procedure *DBMS_SPM.LOAD_PLANS_FROM_SQLSET* or through Oracle Enterprise Manager (EM). Next time these statements are executed the SQL plan baselines will be used.

Bulk loading execution plans from a STS is an excellent way to guarantee no plan changes as part of a database upgrade. The following four steps is all it takes:

1. In an Oracle Database 10gR2 create a STS that includes the execution plan for each of the SQL statements.
2. Load the STS into a staging table and export the staging table into a flat file.
3. Import the staging table from a flat file into an Oracle Database 11g and unload the STS.
4. Use EM or *DBMS_SPM.LOAD_PLANS_FROM_*SQLSET to load the execution plans into the SQL Management Base.

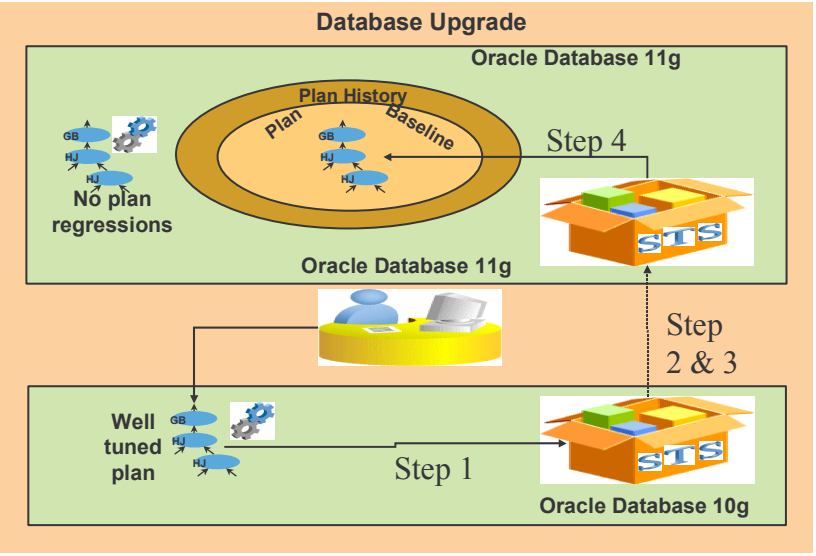

Figure 1 Bulk load the SMB for database upgrading using STS.

Once the SQL plan baselines have been created they will be used, guaranteeing no plan changes between 10gR2 and 11gR1. If the optimizer in the Oracle database 11g comes up with a different execution plan, that plan will be added to the plan history and will be marked for verification. It will only be marked accepted if its performance is as good as or better than the current SQL plan baseline (the 10gR2 plan).

From the Cursor Cache

It is possible to load plans for statements directly from the cursor cache into the SQL Management Base. By applying a filter - on the module name, the schema, or the SQL_ID - you can identify the SQL statement or set of SQL statements you wish to capture. The plans can be loaded using the PL/SQL procedure *DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE* or through Oracle Enterprise Manager. The next time these statements are executed their SQL plan baselines will be used.

Loading plans directly from the cursor cache can be extremely useful if the application SQL has been tuned by hand using hints. Since it is unlikely the application SQL can be changed to include the hint, by capturing the tuned execution plan as a SQL plan baseline you can ensure that the application SQL will use that plan in the future.

Unpack SQL baseline plans from a staging table

The deployment of a new application module means the introduction of completely new SQL statements into the database. With Oracle Database 11g, any 3rd party software vendor can ship their application software along with the appropriate SQL plan baselines for new SQL being introduced. This guarantees that all SQL statements that are part of the SQL Plan baseline will initially run with the plans that are known to give good performance under a standard test configuration. Alternatively, if an application is developed or tested in-house, the correct plans can be exported from the test system and imported into production using the following steps:

1. On the original system, create a staging table using the `DBMS_SPM.CREATE_STGTAB_BASELINE` procedure
2. Pack the SQL plan baselines you want to export from the SQL management base into the staging table using the `DBMS_SPM.PACK_STGTAB_BASELINE` function.
3. Export the staging table into a flat file using the export command or data pump.
4. Transfer this flat file to the target system.

5. Import the staging table from the flat file using the import command or data pump.
6. Unpack the SQL plan baselines from the staging table into the SQL management base on the target system using the `DBMS_SPM.UNPACK_STGTAB_BASELINE` function.

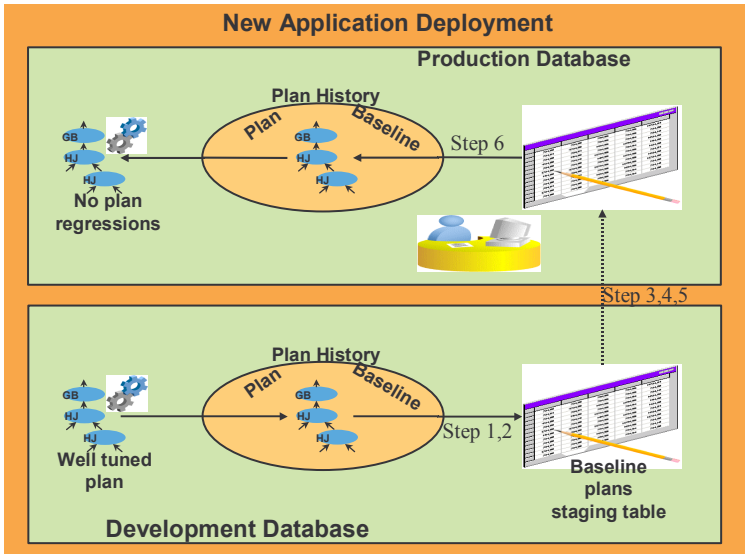

Figure 2 Import SQL plan baselines from test when implementing a new application

SQL Plan Baseline Selection

With SPM only known or verified plans will be selected for execution.

Each time a SQL statement is compiled, the optimizer first uses the traditional cost-based search method to build a best-cost plan. If the initialization parameter *`OPTIMIZER_USE_PLAN_BASELINES`* is set to *`TRUE`* (default value) then before the cost based plan is executed the optimizer will try to find a matching plan in the SQL statement's SQL plan baseline; this is done as in-memory operation, thus introducing no measurable overhead to any application. If a match is found then it proceeds with this plan. Otherwise, if no match is found, the newly generated plan will be added to the plan history; it will have to be verified before it can be accepted as a SQL plan baseline. Instead of executing the newly generated plan the optimizer will cost each of the accepted plans for the SQL statement and pick the one with the lowest cost (note that a SQL plan baseline can have more than one verified/accepted plan for a given statement). However, if a change in the system

(such as a dropped index) causes all of the accepted plans to become non-reproducible, the optimizer will use the newly generated cost-based plan.

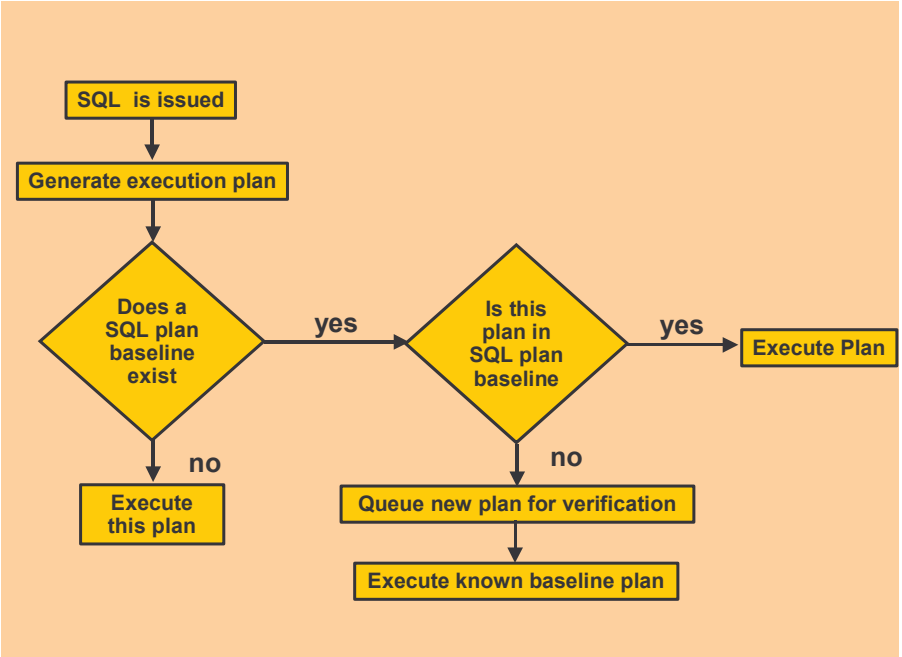

Figure 3 How a SQL execution plan is chosen with SPM

It is also possible to influence the optimizer's choice of plan when it is selecting a plan from a SQL plan baseline. SQL plan baselines can be marked as fixed. Fixed SQL plan baselines indicate to the optimizer that they are preferred. If the optimizer is costing SQL plan baselines and one of the plans is fixed, the optimizer will only cost the fixed plan and go with that if it is reproducible. If the fixed plan(s) are not reproducible the optimizer will go back and cost the remaining SQL plan baselines and select the one with the lowest cost. Note that costing a plan is nowhere near as expensive as a hard parse. The optimizer is not looking at all possible access methods but at one specific access path.

SQL Plan Baseline Evolution

Plans can be manually evolved or verified at any time or you can schedule a database job to run the evolve process.

When the optimizer finds a new plan for a SQL statement, the plan is added to the plan history as a non-accepted plan that needs to be verified before it can become an accepted plan. It is possible to evolve a SQL statement's execution plan using Oracle Enterprise Manager or by running the command-line function *DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE*. Using either of these methods you have three choices:

1. Accept the plan only if it performs better than the existing SQL plan baseline
2. Accept the plan without doing performance verification
3. Run the performance comparison and generate a report without evolving the new plan.

If you choice option 1, it will trigger the new plan to be evaluated to see if it performs better than a selected plan. If it does, then the new plan will be added to the SQL plan baseline, as an accepted plan, if not the new plan will remain in the plan history as a non-accepted plan but it's LAST_VERIFIED attribute will be updated with the current timestamp. A formatted text report is returned by the function, which contains the actions performed by the function as well as side-by-side display of performance statistics of the new plan and the original plan.

If you choose option 2, the new plan will be added to the SQL plan baseline, as an accepted plan without verifying its performance. The report will also be generated.

If you choose option 3 the new plan will be evaluated to see if it performs better than a selected plan but it will not be accepted automatically if it does. After the evaluation only the report will be generated.

USING AND MANAGING THE SQL MANAGEMENT BASE

Initialization parameters

There are two init.ora parameters to control SPM.

optimizer_capture_sql_plan_baselines Controls the automatic capture of new SQL plan baselines for repeatable SQL statements. Set to false by default in 11gR1.

optimizer_use_sql_plan_baselines controls the use of SQL plan baselines. When enabled, the optimizer looks for plans in SQL plan baselines for the SQL statement being compiled. If any are found, then the optimizer will cost each plan in the SQL plan baseline and pick the one with the lowest cost. Set to true by default in 11gR1.

Managing the space consumption of SQL Management Base

The statement log, the plan histories, and SQL plan baselines are stored in the SQL Management Base. The SQL Management Base is part of the database dictionary, stored in the SYSAUX tablespace. By default, the space limit for SQL Management Base is no more than 10% of the size of the SYSAUX tablespace. However, it is possible to change the limit to any value between 1% and 50% using the PL/SQL procedure `DBMS_SPM.CONFIGURE`. A weekly background process measures the total space occupied by the SQL Management Base, and when the defined limit is exceeded, the process will generate a warning in the alert log.

There is also a weekly scheduled purging task that manages the disk space used by SPM inside the SQL Management Base. The task runs automatically in the maintenance window and any plans that have not been used for more than 53

weeks are purged, thus ensuring any SQL statements that are run just once a year are kept available. It is possible to change the unused plan retention period using either using DBMS_SPM.CONFIGURE or Enterprise Manager. Its value can range from 5 to 523 weeks (a little more than 10 years).

Because SQL Management Base is stored entirely within the SYSAUX tablespace, SPM will not be used if this tablespace is not available.

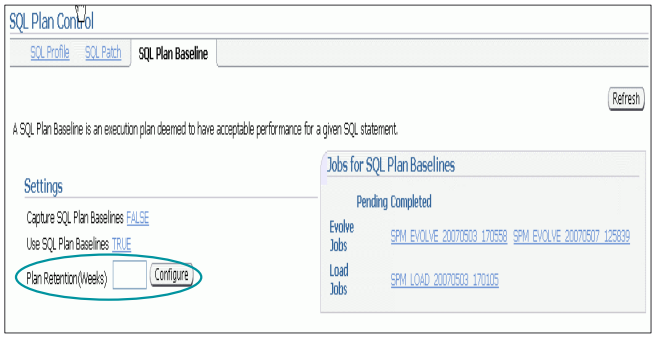

Figure 4 Change plan retention setting in EM

MONITORING THE SQL PLAN MANAGEMENT

Use either EM DBControl or the new dictionary view DBA_SQL_PLAN_BASELINES to monitor SPM.

Several new Enterprise Manage screens and DBA views have been introduced to monitor the SPM functionality in Oracle Database 11g.

Enterprise Manager

All aspects of managing and monitoring SQL plan baselines can be done through Enterprise Manager Database Control.

Getting started

To get to the SQL plan baseline page:

1. Access the Database Home page in Enterprise Manager.
2. At the top of the page, click Server to display the Server page.
3. In the Query Optimizer section, click SQL Plan Control.
4. The SQL Plan Control page appears. See the online help for information about this page.
5. At the top of the page, click SQL Plan Baseline to display the SQL plan baseline subpage.

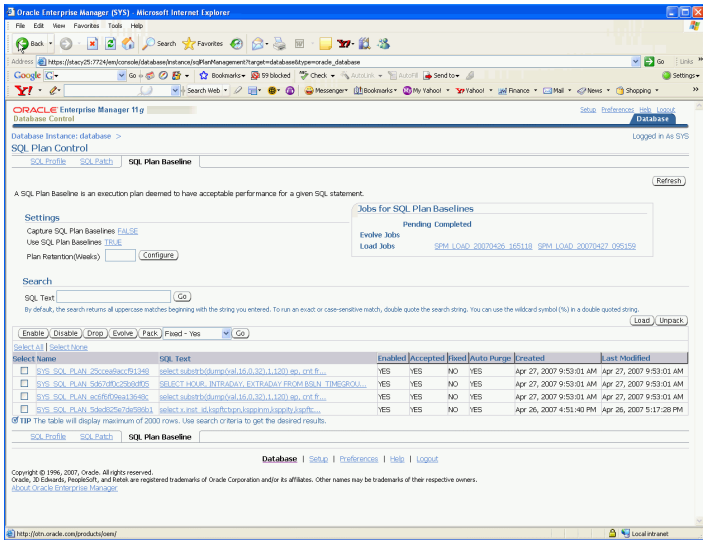

Figure 5 SQL plan baseline home page in Oracle Enterprise Manager DB control

From the main page you can control the init.ora parameters, schedule load or evolve jobs as well as change some attributes for an existing SQL plan baseline.

Change init.ora parameter values

In the upper left hand side of the main SQL plan baseline page is the setting section, which lists the parameters that control SQL Plan Management. A quick glance at this section will let you know if automatic SQL plan baseline capture is on or if a SQL plan baseline will be used or not. To change the value of an init.ora parameter

1. Click on the value of the parameter
2. The initialization parameter page will open. Select the value you want to change the parameter to from the drop down menu
3. Click on OK

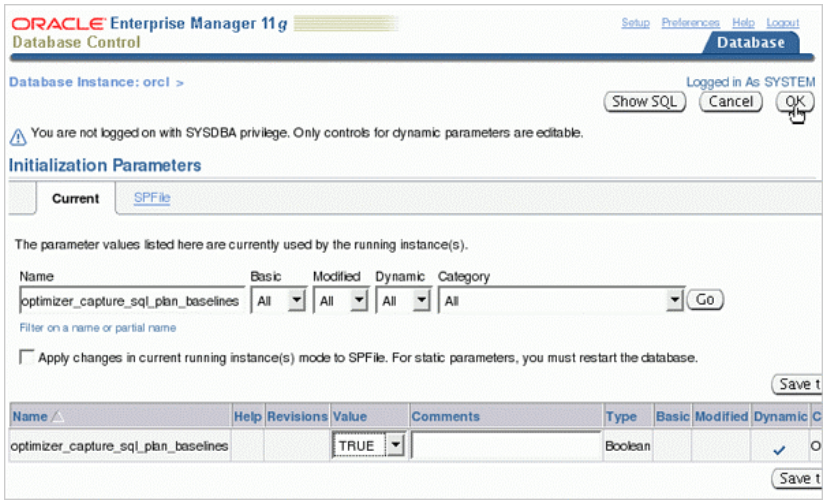

Figure 6 Setting SPM init.ora parameters in EM

Bulk Loading plans

You can load plans straight from the cursor cache using the load button on the right hand side above the list of SQL plan baselines. It is possible to load plans for all of the statements in the cursor cache or you can select a subset of plan.

1. Click on the load button
2. The load SQL plan baseline page will appear. Select the radio button load plans for "load from the cursor cache"
3. Enter one or more SQL_ID manually or click on the flashlight to see a list of all the SQL_ID and the SQL for every plan in the cursor cache.
4. After selecting your SQL_ID(s) complete the job-scheduling information (default load immediately)
5. Click OK

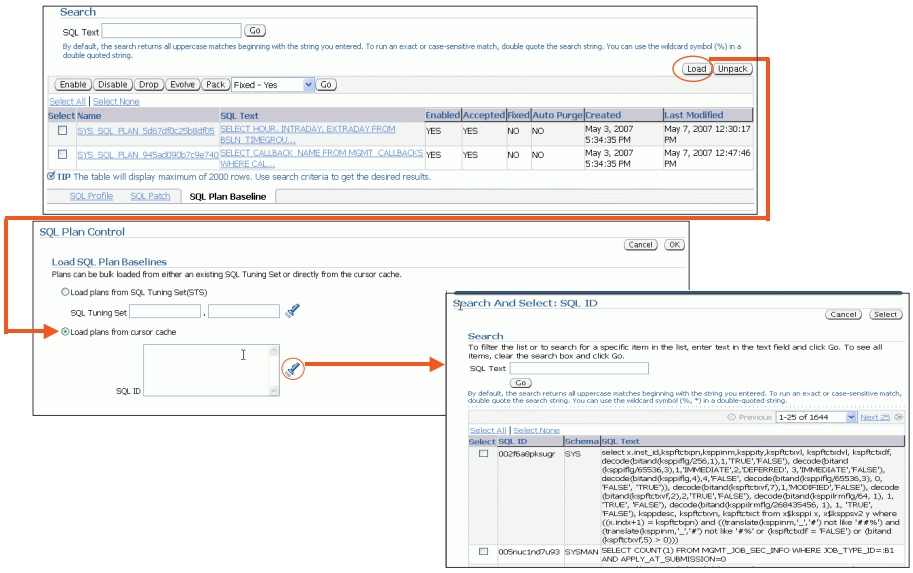

Figure 7 Bulk loading SQL plan baselines from the cursor cache in EM

Change an Attribute

From the main SQL plan baseline page it is possible to change any attribute of a plan baseline. To change an attribute

1. Click on the checkbox in front of the plan baseline
2. Click on the attribute button you want to change.
3. A dialog box will appear asking you to confirm your selection. Click OK

View a SQL plan baseline's execution plan

To view the actual execution plan for SQL plan baseline click on the plan name. To view execution plans for SQL plan baselines for a given SQL statement click on the SQL text.

Evolve a SQL plan baseline.

From the main SQL plan baseline page you can see which plans are accepted and which are not. If you would like to evolve an unaccepted plan

1. Click on the Checkbox in front of the plan and select the evolve button above the list
2. The evolve SQL plan baseline page will open with three radio button options

a. **Verify Performance** – if you want to guarantee the unaccepted plan performs as good as or better than the existing SQL plan baseline then select **YES**. If you already know the unaccepted plan has good performance and would like to by-pass the check select **NO.**

b. **Time Limit -** applies only when you select **Yes** for Verify performance. **Auto** means Oracle will decide how long to spend verifying the performance of non-accepted plans. **Unlimited** means the plan verification process will be run to completion. **Specify** means you need to set a time limit for the plan verification process.

c. **Action** – Do you want the new plan to be automatically accepted or do you just want a report on the outcome of the verification process based on which you can decide to accept the new plan or not.

3. Click OK

4. The SQL plan baseline main page will appear. You should see you evolve job listed in the jobs section in the upper right hand side of the page. (Click refresh if necessary).

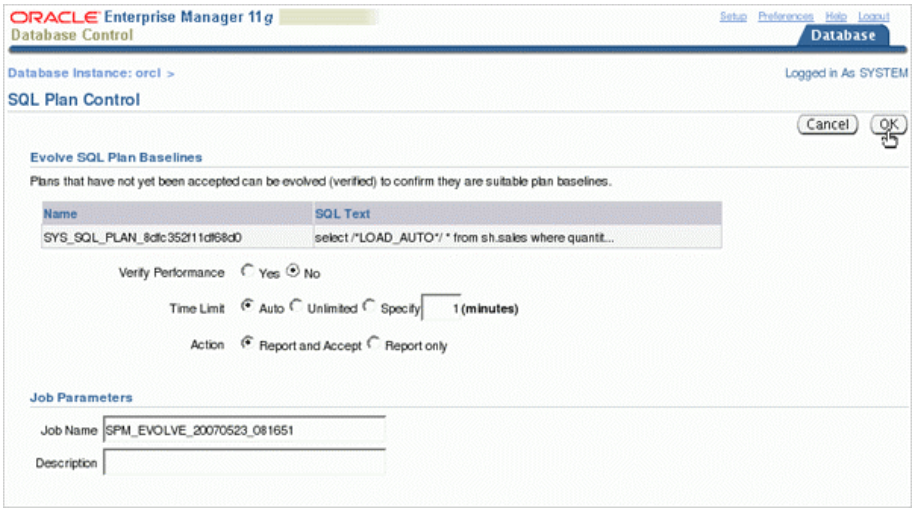

Figure 8 Plan Evolutions

Monitoring SPM through DBA views

The view *DBA_SQL_PLAN_BASELINES* displays information about the SQL plan baselines currently created for specific SQL statements.

```
select sql_handle, sql_text, plan_name, origin,
enabled, accepted, fixed, autopurge
from dba_sql_plan_baselines;
```

The above select statement returns the following rows

```
SQL_HANDLE   SQL_TEXT   PLAN_NAME      ORIGIN  ENA ACC FIX AUT
--------   ---------- -------------    ------- --- --- --- ---
SYS_SQL_6fe2   select... SYS_SQL_PLAN_1ea AUTO-CAP YES NO NO YES
SYS_SQL_6fe2   select... SYS_SQL_PLAN_4be AUTO-CAP YES YES NO YES

...
```

In this example the same SQL statement has two plans, both of which were automatically captured. One of the plans (SYS_SQL_PLAN_4be) is a plan baseline as it is both enabled and accepted. The other plan (SYS_SQL_PLAN_1ea) is a non-accepted plan, which has been queued for evolution or verification. It has been automatically captured and queued for verification; its accepted value is set to NO. Neither of the plans is fixed and they are both eligible for automatic purge.

To check the detailed execution plan for any SQL plan baseline you can use the procedure *DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE*.

It is also possible to check whether a SQL statement is using a SQL plan baseline by looking in *V$SQL*.If the SQL statement is using a SQL plan baseline the plan_name for the plan selected from that SQL plan baseline will be in the sql_plan_baseline column of *V$SQL*. You can join the *V$SQL* view to the *DBA_SQL_PLAN_BASELINES* view using the following query:

```
Select s.sql_text, b.plan_name, b.origin, b.accepted
From dba_sql_plan_baselines b, v$sql s
Where s. exact_matching_signature = b.signature
And   s.SQL_PLAN_BASELINE = b.plan_name;
```

INTEGRATION WITH AUTOMATIC SQL TUNING

In Oracle Database 11g, the SQL Tuning Advisor, a part of the Tuning and Diagnostics pack, is automatically run during the maintenance window. This automatic SQL tuning task targets high-load SQL statements. These statements are identified by the execution performance data collected in the Automatic Workload Repository (AWR) snapshots. If the SQL Tuning Advisor finds a better execution plan for a SQL statement it will recommend a SQL profile. Some of these high-load SQL statements may already have SQL plan baselines created for them. If a SQL profile recommendation made by the automatic SQL tuning task is

implemented, the execution plan found by the SQL Tuning Task will be added as an accepted SQL plan baseline.

The SQL Tuning Advisor can also be invoked manually, by creating a SQL Tuning Set for a given SQL statement. If the SQL Tuning Advisor recommends a SQL profile for the statement and it is manually implemented then that profile will be added as an accepted plan to the SQL statements plan baseline if one exists.

CONCLUSION

In Oracle Database 11g a new feature, SQL Plan Management (SPM) provides controlled execution plan evolution. With SPM the optimizer automatically manages execution plans and ensures only known or verified plans are used. When a new plan is found for a SQL statement it will not be used until it has been verified to have comparable or better performance than the current plan.

SQL Plan management in Oracle Database 11g
June 2007
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