An Oracle White Paper
October 2009

SQL Plan Management in Oracle Database 11g

ORACLE

Introduction

The performance of any database application heavily relies on 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 lead 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 11*g* 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

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:

Guaranteed plan stability and controlled plan evolution.

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 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.

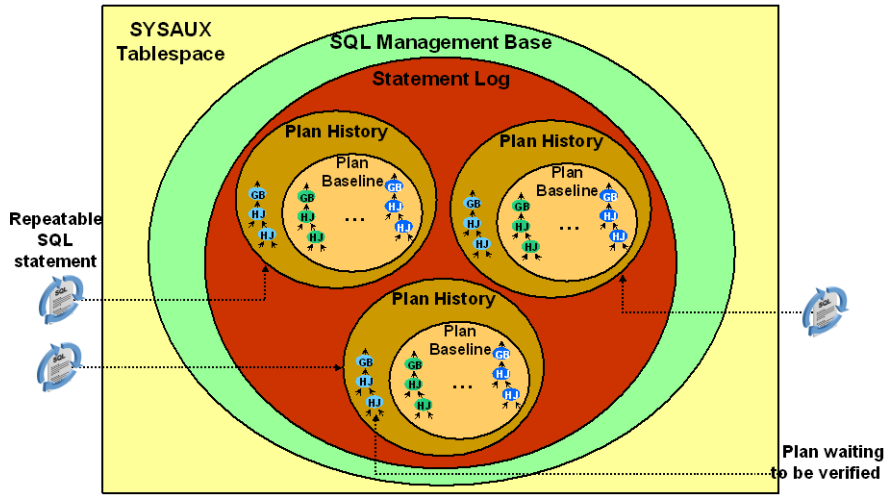

Figure 1 SQL Management base, consisting of the statement log and plan histories for repeatable SQL Statements

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` (default `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.

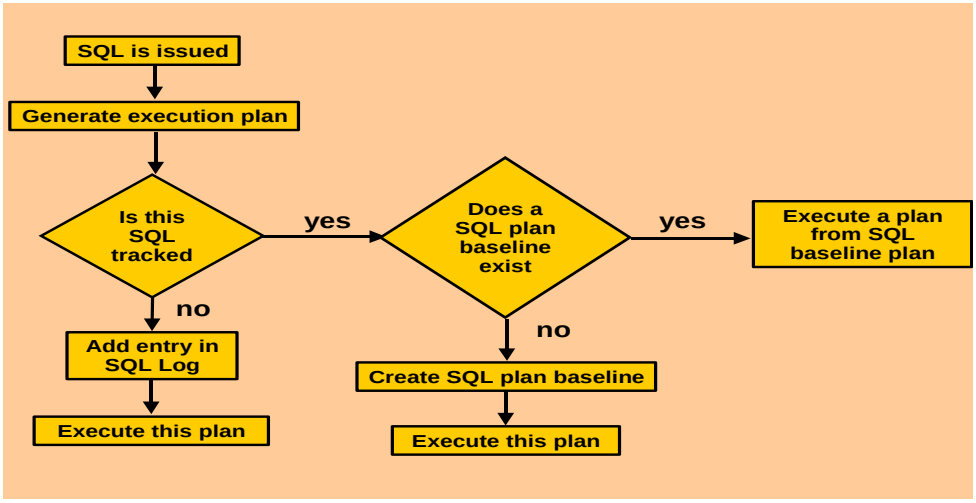

Figure 2 Flow chat of how Automatic plan Capture works.

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 four different techniques:

1. Populate the execution plans for a given SQL Tuning Set (STS)
2. Populate the execution plans from Stored Outlines
3. Use the execution plans currently in the Cursor Cache
4. 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 an 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.

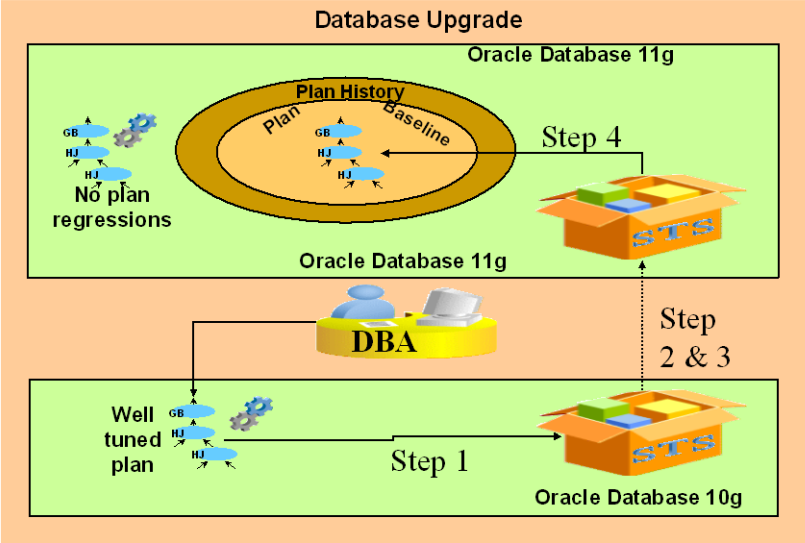

Figure 3 Bulk load the SMB for database upgrading using STS.

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 Stored Outlines

If you don't have access to SQL Tuning Sets or if you are upgrading from an earlier version than Oracle Database 10gR2 you can capture your existing execution plan using Stored Outlines. Stored Outlines can be loaded into the SQL Management Base as SQL plan baselines using the PL/SQL procedure `DBMS_SPM.MIGRATE_STORED_OUTLINE` or through Oracle Enterprise Manager (EM). Next time these statements are executed the SQL plan baselines will be used.

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.

1. You should begin by starting a new session and 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.
2. Then execute the workload either by running the 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.
4. The actual Stored Outlines are stored in the OUTLN schema. You can either export the schema and import it into the 11g database or upgrade your existing database to 11g.
5. Use EM or `DBMS_SPM.MIGRATE_STORED_OUTLINE` to load the Stored Outlines into the SQL Management Base.

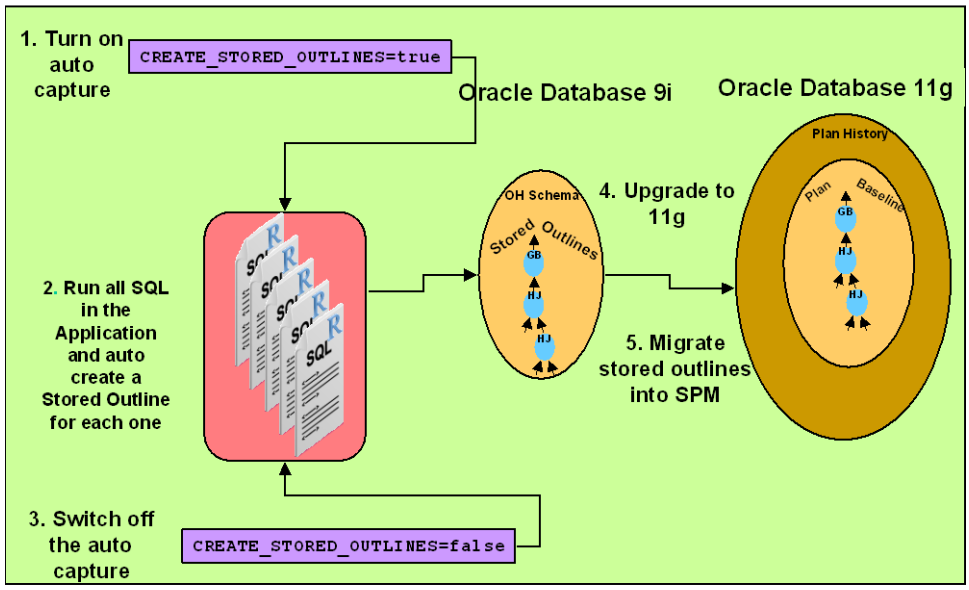

Figure 4 Bulk load the SMB after upgrade using Stored outlines.

From the Cursor Cache

Starting in Oracle Database 11g 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 statement 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 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. By using the simple steps below you can use SPM to capture the hinted execution plan and associate it with the non-hinted SQL statement. You begin by capturing a SQL plan baseline for the non-hinted SQL statement.

1. In a SQL*Plus session run the non-hinted SQL statement so we can begin the SQL plan baseline capture

```
SQL> SELECT prod_name, SUM(amount_sold)
           FROM   Sales s, Products p
           WHERE  s.prod_id=p.prod_id
            AND      prod_category = :ctgy
            GROUP BY prod_name;
```

2. Then find the SQL_ID for the statement in the V$SQL view.

```
SQL> SELECT sql_id, sql_fulltext
           FROM   V$SQL
           WHERE  sql_text LIKE '%SELECT prod_name, SUM(%';

SQL_ID                    SQL_FULLTEXT
-------------       ----------------------------------------
74hnd835n81yv      select SQL_ID, SQL_FULLTEXT from v$SQL
chj6q8z7ykbyy      SELECT PROD_NAME, SUM(AMOUNT_SOLD)
```

3. Using the SQL_ID create a SQL plan baseline for the statement.

```
SQL>  variable cnt number;
SQL> EXECUTE :cnt :=DBMS_SPM.LOAD_PLAN_FROM_CURSOR_CACHE(
     sql_id=>'chj6q8z7ykbyy');
```

4. The plan that was captured is the sub-optimal plan and it will need to be disabled. The SQL_HANDLE & PLAN_NAME are required to disable the plan. These can found by looking in DBA_SQL_PLAN_BASELINE view.

```
SQL> SELECT sql_handle, sql_text, plan_name, enabled FROM
dba_sql_plan_baselines;
```

```
   SQL_HANDLE              SQL_TEXT              PLAN_NAME           ENABLE
---------------------- --------------------- ---------------------- ------
SYS_SQL_bf5c9b08f72bde3e  SELECTPROD_NAME,SUM  SQL_PLAN_byr4v13vkrrjy42949306 Y
```

5. Using DBMS_SPM.ALTER_SQL_PLAN_BASELINE disable the bad plan

```
SQL>  variable cnt number;
SQL>  exec :cnt :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE(
         SQL_HANDLE =>      'SYS_SQL_bf5c9b08f72bde3e',
         PLAN_NAME
=>      'SQL_PLAN_byr4v13vkrrjy42949306',
         ATTRIBUTE_NAME =>  'enabled',
         ATTRIBUTE_VALUE => 'NO');

SQL> SELECT sql_handle, sql_text, plan_name, enabled
     FROM   dba_sql_plan_baselines;

   SQL_HANDLE              SQL_TEXT              PLAN_NAME           ENABLE
---------------------- --------------------- ---------------------- ------
SYS_SQL_bf5c9b08f72bde3e  SELECTPROD_NAME,SUM  SQL_PLAN_byr4v13vkrrjy42949306 N
```

6. Now you need to modify the SQL statement using the necessary hints & execute the modified statement.

```
SQL> SELECT /*+ INDEX(p) */ prod_name, SUM(amount_sold)
   FROM   Sales s, Products p
   WHERE  s.prod_id=p.prod_id
   AND    prod_category = :ctgy
   GROUP BY prod_name;
```

7. Find the SQL_ID and PLAN_HASH_VALUE for the hinted SQL statement in the V$SQL view.

```
SQL> SELECT sql_id, plan_hash_value, fulltext
           FROM   V$SQL
    WHERE  sql_text LIKE '%SELECT /*+ INDEX(p) */
prod_na%';

SQL_ID         PLAN_HASH_VALUE    SQL_FULLTEXT
-------------  ---------------  --------------------------

9t5v8swp79svs   3262214722      select SQL_ID, SQL_FULLTEXT
djkqjd0kvgmb5   3074207202      SELECT /*+ INDEX(p) */
```

8. Using the SQL_ID and PLAN_HASH_VALUE for the modified plan, create a new accepted plan for original SQL statement by associating the modified plan to the original statement's `SQL_HANDLE`.

```
exec :cnt:=dbms_spm.load_plans_from_cursor_cache(
      sql_id => 'djkqjd0kvgmb5',
      plan_hash_value => 3074207202,
      sql_handle => 'SYS_SQL_bf5c9b08f72bde3e');
```

Unpack 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 Oracle 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 Oracle 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.

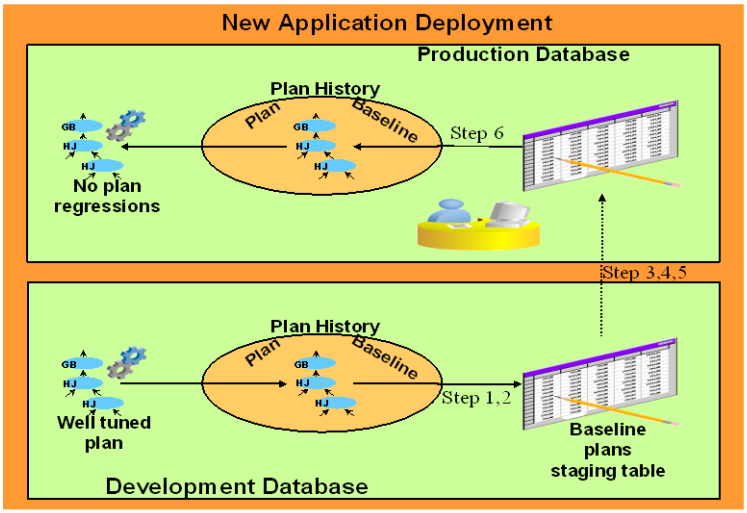

Figure 5 Import SQL plan baselines from test when implementing a new application

More information on all of the bulk loading techniques can be found in the Using SQL Plan Management For Upgrade section below.

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_SQL_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.

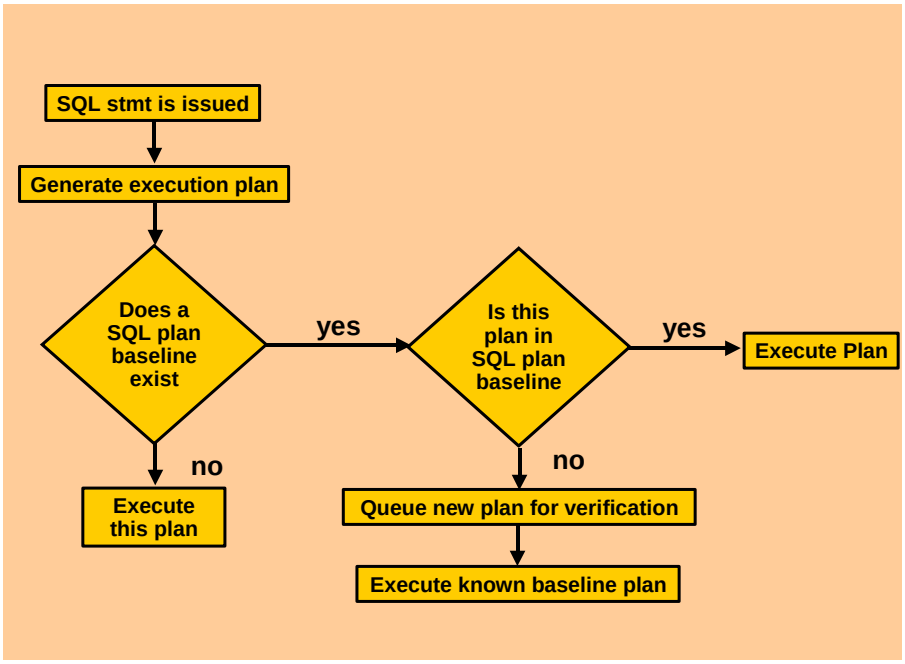

Figure 6 How a SQL execution plan is chosen with SPM

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 choose option 1, it will trigger the new plan to be evaluated to see if it performs better than a selected plan baseline. 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 its `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 baseline 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 new 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 has 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). See Figure 6 below.

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

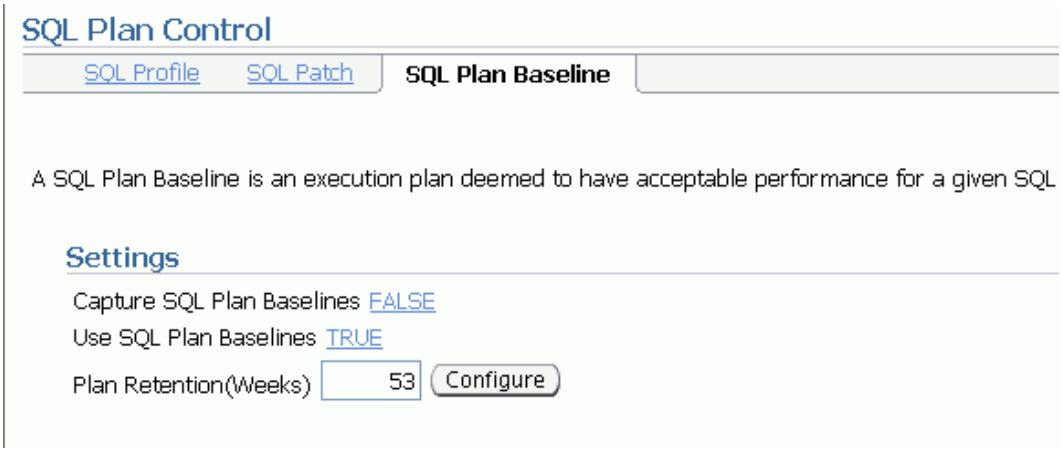

Figure 7 Change plan retention setting in EM

Monitoring SQL Plan Management

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

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

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.

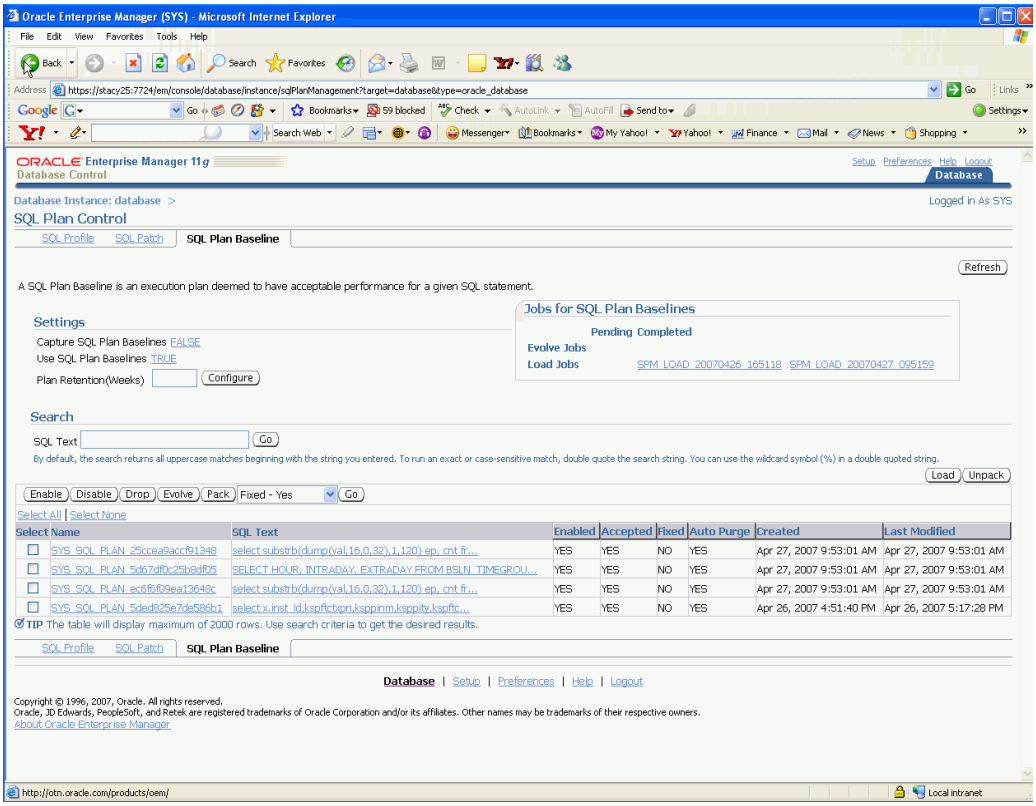

Figure 8 SQL plan baseline home page in Oracle Enterprise Manager DB Control

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 Settings 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 (see figure 8). Select the value you want to change the parameter to from the drop down menu
3. Click on OK

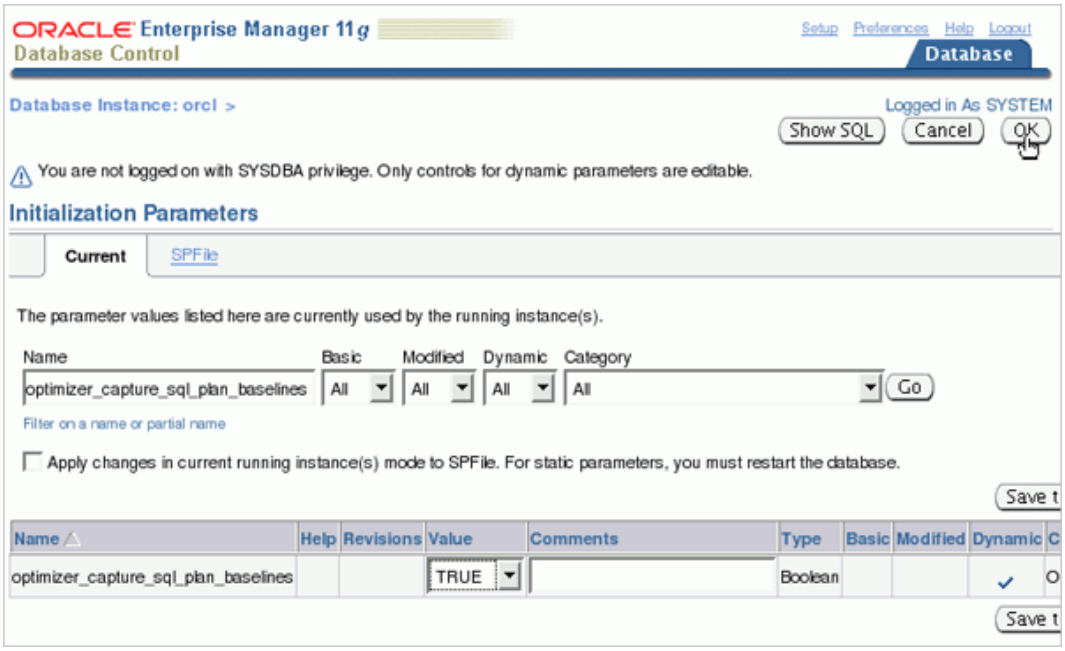

Figure 9 Setting SPM init.ora parameters in EM

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 for "load from the cursor cache" (as shown in the middle of figure 9)
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

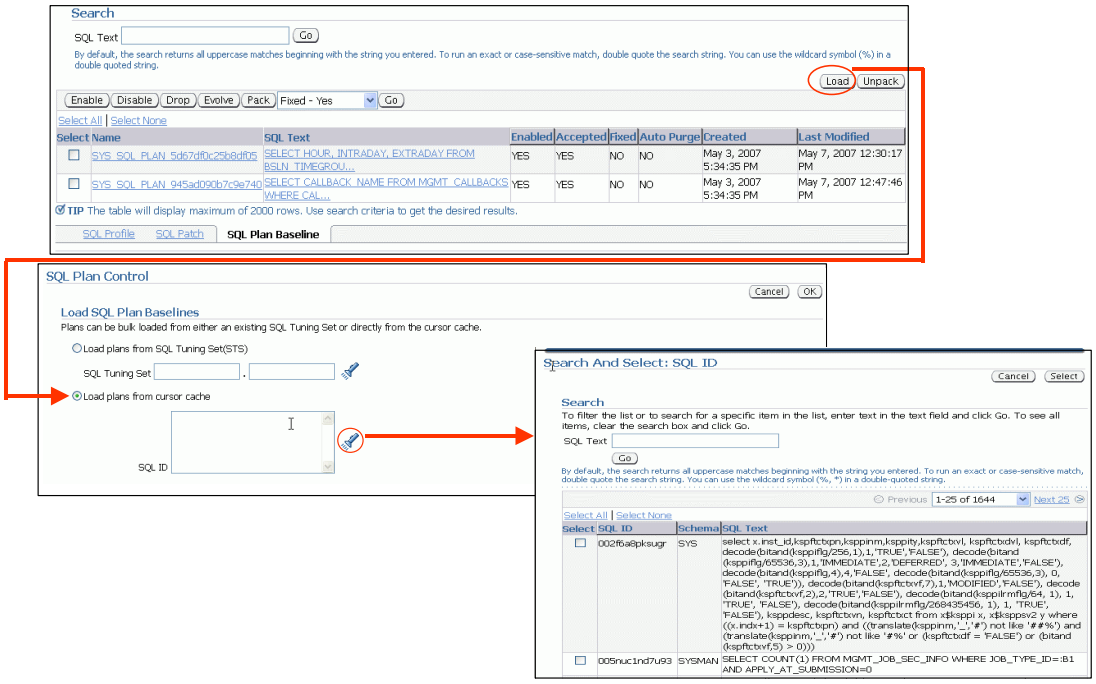

Figure 10 Bulk loading SQL plan baselines from the cursor cache in EM

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 all execution plans 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 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 your evolve job listed in the Jobs section in the upper right hand side of the page. (Click refresh if necessary)

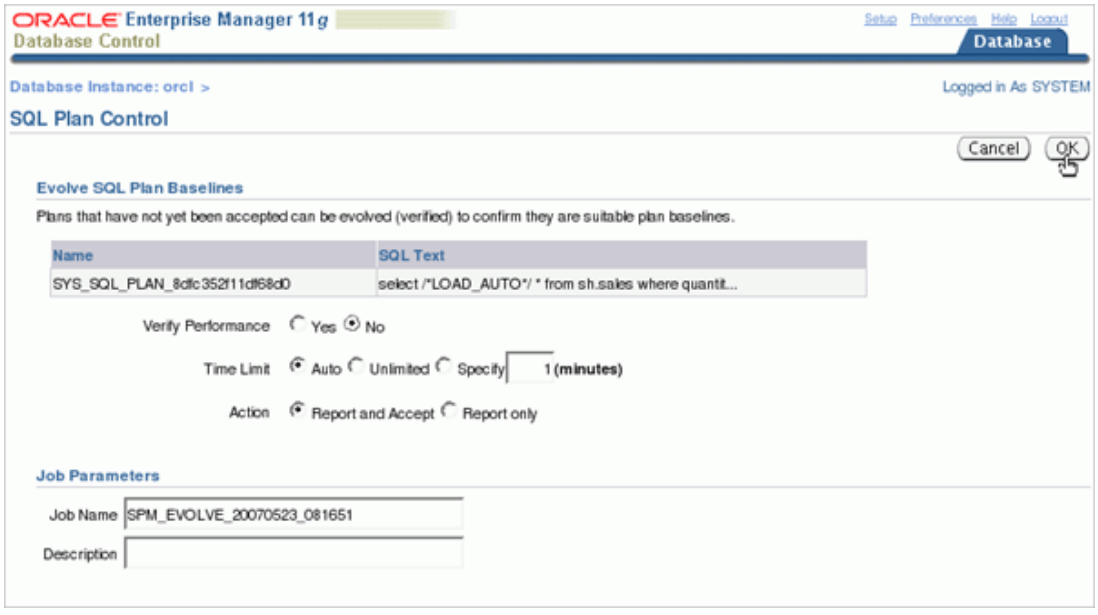

Figure 11 Plan Evolutions

Monitoring SPM through DBA views

The view DBA_SQL_PLAN_BASELINES displays information about the SQL plan baselines currently created for specific SQL statements. Here is an example.

```
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.

Using SQL Plan Management for 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 potential behavior changes are execution plan changes. With the introduction of SQL Plan Management you now have an additional safety net to ensure execution plans don't change during the upgrade. In order to take full advantage of this safety net you need to capture your existing execution plans before you upgrade so they can be used to seed SPM.

Using SQL Tuning Sets

If you have access to SQL Tuning Sets (STS) in the diagnostics pack then this is the easiest way to capture your existing 10g execution plans. An STS is a database object that includes one or more SQL statements along with their execution statistics, execution context and their current execution plan. (An STS in Oracle Database 10gR1 will not capture the execution plans for the SQL statements so it can't be used to seed SPM. Only a 10gR2 STS will capture the plans).

To begin you will need to create a new STS. You can either do this through Oracle Enterprise Manager (EM) or using the `DBMS_SQLTUNE` package. In this example we will use `DBMS_SQLTUNE`.

```
BEGIN
 SYS.DBMS_SQLTUNE.CREATE_SQLSET (
            sqlset_name  => 'SPM_STS',
            description => '10g plans');
END;
\
```

Once the STS has been created we need to populate it. You can populate an STS from the workload repository, another STS or from the cursor cache. In this case we will capture the SQL statements and their execution plans from the cursor cache. This is a two-step process. In the first step we create a ref cursor to select the specified SQL from the cursor cache (in this case all non sys SQL statements). Then in the second step we use that ref cursor to populate the STS.

```
DECLARE
   stscur   dbms_sqltune.sqlset_cursor;
BEGIN
   OPEN stscur FOR
     SELECT VALUE(P)
     FROM   TABLE(dbms_sqltune.select_cursor_cache(
            'parsing_schema_name <> ''SYS''',
            null, null, null, null, 1, null, 'ALL')) P;

   -- populate the sqlset
   dbms_sqltune.load_sqlset(sqlset_name      => 'SPM_STS',
                            populate_cursor  => stscur);
END;
/
```

Once the software upgrade is completed the execution plans can be bulk loaded from an STS into SPM using the PL/SQL procedure `DBMS_SPM.LOAD_PLANS_FROM_SQLSET` or through Oracle Enterprise Manager (EM).

```
SQL> Variable cnt number
SQL> execute :cnt := DBMS_SPM.LOAD_PLANS_FROM_SQLSET( -
                        sqlset_name  => 'SPM_STS');
```

Using Stored Outlines

If you don't have access to SQL Tuning Sets you can capture your existing execution plan using Stored Outlines. 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.

1. Start a new session and issue 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 user for which the Stored Outlines are to be created has the `CREATE ANY OUTLINE` privilege. If they don't the Stored Outlines will not 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
```

6. `After the upgrade to` Oracle Database 11gR2, you can migrate stored outlines for one or more SQL statements to SQL plan baselines using `DBMS_SPM.MIGRATE_STORED_OUTLINE` or through Oracle Enterprise Manager (EM). You can specify which stored outline(s) to be migrated based on outline name, SQL text, or outline category, or migrate all stored outlines in the system to SQL plan baselines.

```
SQL> variable report clob;

-- Migrate a single Stored Outline by name

SQL> exec :report:=DBMS_SPM.MIGRATE_STORED_OUTLINE( -
      attribute_name=>'OUTLINE_NAME', attribute_value =>
      'stmt01');

-- Migrate all Stored Outlines

SQL> exec :report:=DBMS_SPM.MIGRATE_STORED_OUTLINE( -
      attribute_name=>'ALL');
```

Note: If you are not planning on doing an in-place upgrade you will have to move the STS, or Stored Outlines to the Oracle Database 11g system.

Bulk loading from the Cursor Cache

Loading plans directly from the cursor cache can be extremely useful if you were unable to capture plans for some or all of your statements prior to the upgrade.

By setting the parameter `OPTIMIZER_FEATURES_ENABLE` to the 10g version used before the upgrade, you should be able to revert back to the same execution plan you had prior to the upgrade. By capturing these 10g execution plans from the cursor cache you will be able to seed SPM with the 10g plans before setting `OPTIMIZER_FEATURES_ENABLE` to your 11g version. Note you must use the exact same Optimizer statistics you were using in 10g. Statistics should not be re-gathered until after all of the 10g plans have been captured.

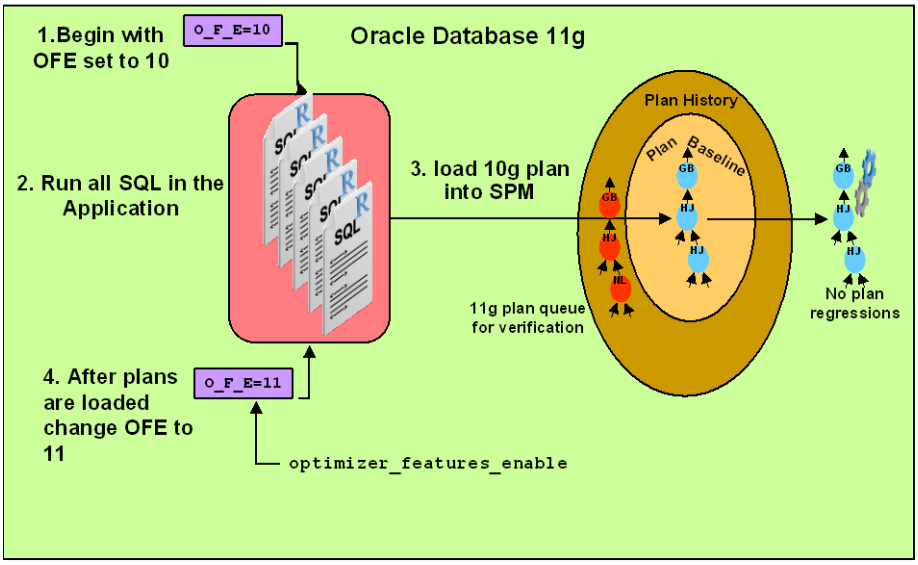

Figure 12 Upgrading by capturing 10g plans from the cursor cache.

Conclusion

In Oracle Database 11g a new feature, SQL Plan Management was introduced to provide controlled execution plan evolution. With SPM the optimizer automatically manages execution plans and ensures only known or verified plans are used. By seeding or loading SPM with execution plans from your current version of the Oracle Database prior to upgrading, you can prevent performance regressions due to plan changes. By only executing the known or seeded plans your application will behave exactly the same why in Oracle Database 11gR2 as it did in the previous release. Any new execution plans found in Oracle Database 11gR2 will be recorded in the plan history for that statement but will not be used. The recorded or unaccepted plans will only be used if they have been verified to perform better than the existing accepted plan for each statement. By allowing you to control what execution plans will be used not only during upgrade but as your system grows and evolves on Oracle Database 11g your system will be more stable and will have consistently good performance.

White Paper Title SQL Plan Management in
oracle Database 11g
October 2009
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

Oracle is committed to developing practices and products that help protect the environment

0109