Finally… Oracle Plan Stability

Dave Hungle, DBCORP Information Systems Inc.

Introduction

Many Oracle DBAs have spent countless hours tracking down sudden changes in Oracle database performance. Often, they find that an SQL statement, which was running fine just a few days ago, is now running very poorly. Frequently, after a significant amount of time and effort, they find that an environmental change has caused the Oracle optimizer to alter the execution plan for some key SQL statements.

When I first started working with the Oracle DBMS in the early 1990s, I was very surprised to find there was no way to preserve the performance characteristics for SQL statements. Having worked with other relational database products which had this type of capability, I was startled to learn that the Oracle optimizer would dynamically determine an SQL statement's execution plan each time it was run, if it was not already in the shared pool. Due to the dynamic nature of the Oracle optimizer and its sensitivity to environmental issues, very minor changes in the environment can cause the Oracle optimizer to change an SQL statement's execution plan without warning. As many database professionals know, the Oracle optimizer is not perfect and it can create execution plans that may have a negative impact on performance.

Oracle8ihas introduced a new feature called 'Plan Stability'. This feature allows you to preserve an SQL statement's performance characteristics. This capability guarantees that the Oracle optimizer will create the same execution plan for an SQL statement no matter what changes are made to the database environment. The plan stability feature should go a long way towards providing some performance stability for production applications.

What is Plan Stability?

The Oracle plan stability feature allows you to store information about individual SQL statements in the Oracle data dictionary. The stored information consists of Oracle optimizer hints that can be used by the optimizer to generate an execution plan. This stored information is referred to as an 'OUTLINE'. A single outline maps to a single SQL statement.

If you create outlines for all your SQL statements and then enable the use of stored outlines, the Oracle optimizer will search for the appropriate outline for the SQL statement being executed and uses the stored outline information to generate the execution plan. The execution plan generated will be the same as the one generated when the outline was created. This action guarantees that Oracle always generates the same execution plan for an SQL statement that has a matching outline.

Why use the Plan Stability Feature?

Most DBAs know that the Oracle optimizer can be very sensitive to environmental factors. Plan stability provides a facility to prevent Oracle from generating different execution plans for SQL statements due to environmental changes. Unless you work in a very proactively managed type of environment, most DBAs will not know what an SQL statement's execution plan looks like until a performance issue arises. Therefore, it can take considerable time and effort to determine that some kind of environmental change has caused the Oracle optimizer to change an SQL statement's execution plan. Some of the most common environmental changes that can influence the way the Oracle optimizer works are:

- **Oracle initialization parameter changes.** DBAs frequently change init.ora parameters. Changes in many of these parameters can influence and change the way the Oracle optimizer constructs execution plans. Two parameters that can have a large impact are `OPTIMZER_MODE` and `SORT_AREA_SIZE`.

- **Oracle release level changes.** Oracle is continually developing new data access techniques and data operations. In order to take advantage of these new features Oracle must change the Optimizer code. Every new release level of Oracle typically puts all execution plans at risk of changing.
- **Schema object changes.** Some of the most common schema changes that can cause the optimizer to change an execution plan are table indexes changes or the addition of new indexes.
- **Changes in database statistics.** The existence of object statistics in the Oracle data dictionary can cause the optimizer to automatically switch from RULE based to COST based mode. Often, major performance issues arise as soon as statistics are generated for database objects. The Oracle cost based optimizer depends heavily on database statistics when creating an execution plan. As the statistics change, the Oracle optimizer can make changes to an SQL statement's execution plan.

Environmental changes often cause the optimizer to make execution plan changes that are intended to improve the performance for a given SQL statement. Unfortunately, these types of environmental changes can also cause execution plan changes that have a negative performance impact for an SQL statement. Over the course of my career, I have spent a great deal of time investigating changes in applications performance characteristics. The use of the plan stability feature will be a great way of preserving execution plans for performance sensitive SQL statements. Use of this feature will prevent the optimizer from changing a statement's execution plan. This will eliminate the risk of execution plans being negatively impacted by environmental changes.

It is important to understand that once you create a stored outline for an SQL statement and enable its use, Oracle will always use the same outline information to generate an execution plan. In other words, if you are using an outline for a given SQL statement and environmental changes are made that do improve the performance of the SQL statement, the statement will not benefit from the improved performance.

There are still a surprising number of applications today that run on Oracle with the optimizer in a RULE based mode. Out of the box, Oracle is configured for an optimizer mode of CHOOSE, which means that unless statistics are generated for the database objects the optimizer will operate in its RULE based mode. Many DBAs have generated statistics for an application's schema objects only to have the phone ring off the hook with complaints about poor performance. As soon as object statistics are generated, the optimizer automatically switches to COST based mode. This action can dramatically change all SQL statement execution plans. This can negatively effect the performance for key SQL statements, which may have been specifically tuned for use with the RULE based optimizer. The Plan Stability feature provides a way to preserve the execution plans of key SQL statements and move to the use of the cost based optimizer in a more controlled fashion, without negatively impacting some of the applications key SQL statements.

The Plan Stability feature provides you with the ability to gather information to build a repository of SQL statements in the database. Even if you do not intend to use them, creating stored outlines will build a repository of SQL statement information that can be very useful for tuning exercises and analyzing the impact of schema changes.

Enabling Plan Stability

There are no special init.ora parameters that need to be set before you use the plan stability feature. Oracle recommends caution if you are using the plan stability feature in an environment that uses distributed database queries. Ensure that parameter settings, especially those ending with the suffix "`_ENABLED`", be consistent across the execution environments for outlines to function properly. The main ones to check are:

- QUERY_REWITE_ENABLED
- STAR_TRANSFORMATION_ENABLED
- OPTIMIZER_FEATURES_ENABLED

Oracle also recommends that you do not set the `CURSOR_SHARING` initialization parameter to `FORCE` if you are going to use the plan stability feature. If this parameter is set, Oracle may substitute constants in predicates with internal bind variables and in turn will not be able to match these SQL statements to outlines.

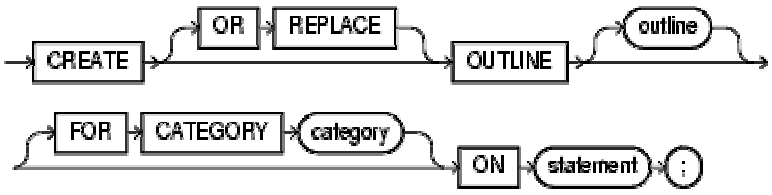
The Plan Stability option seems to be enabled only in the Enterprise Edition of Oracle8i. You can check the `v$option` view to check the 'Plan Stability' setting on your database. I also recommend that you do not use the plan stability feature unless you are running Oracle version 8.1.6 or higher.

Creating Outlines

Before a user can create outlines they must be granted the '`create any outline`' privilege. You can either explicitly create stored outlines or Oracle can create outlines automatically. When you explicitly create a stored outline, you direct Oracle to analyze a single SQL statement, create an execution plan, and store the appropriate hints in the data dictionary. This process is very similar to the process for explaining an SQL statement. If you direct Oracle to automatically create outlines, they are created for each SQL statement run for any given session or, optionally, instance wide. Be careful if you choose to generate outlines for all SQL statements system wide. If your system is busy and there are many SQL statements, you may generate a large amount of stored data in the Oracle data dictionary.

Explicitly creating stored outlines

The syntax to explicitly create a stored outline for a single SQL statement is straightforward.



Stored Outlines map one to one to single SQL statements. Stored outline names must be unique across the instance. If you do not specify an outline name, Oracle generates an outline name consisting of the character string `'SYS_OUTLINE'` appended to a date/time string like: `SYS_OUTLINE_0007211019300008.`

The `'FOR CATEGORY` *`category'`* clause is optional and I recommend that you always use it. The category name is a way to group stored outlines. It can be used to reference a group of stored outlines when you want to use or manage stored outlines. If you do not specify a category name, the outline is stored with a category name of `'DEFAULT'.`

The ' ON statement' clause contains the SQL statement for which Oracle will create an outline when the statement is compiled. All DML statements can be in a stored outline. From my tests, the only DDL statement that I found that could be part of an outline is the `'CREATE TABLE'` statement.

Multiple outlines can be created for a single SQL statement, but each outline for the same SQL statement must be in a different category. The following statement creates an outlines called `scott_select_1` in a category named `scott_outlines`:

```
CREATE OUTLINE scott_select_1 FOR CATEGORY scott_outlines
          ON SELECT ename, job, sal, dname
               FROM emp, dept
              WHERE emp.deptno = dept.deptno
                AND NOT EXISTS (SELECT *
                                  FROM salgrade
                                 WHERE emp.sal BETWEEN losal AND hisal);
```

When explicitly creating store outlines it is possible to get the following error:

```
ORA-18004: outline already exists
```

This error usually means that there is already a stored outline with the same name, but it can mean that there is already a stored outline for this SQL statement in the same category.

Automatically Creating Stored Outlines.

Oracle can automatically generate outlines for all eligible SQL statements executed during a particular session or all eligible SQL statements executed system wide. To enable the automatic generation of outlines, use the

`ALTER SESSION or ALTER SYSTEM commands to set a new Oracle8idynamic parameter called, 'CREATE_STORED_OUTLINES'. This parameter is not an init.ora initialization parameter.`

To enable automatic generation of outlines for a particular session you issue the command:

```
ALTER SESSION SET CREATE_STORED_OUTLINES = TRUE;
```

If you set `'CREATE_STORED_OUTLINES = TRUE'` then outlines with be created in the `DEFAULT` category. You can substitute `'TRUE'` for the name of a category to have the outlines created in a specific category. To disable the automatic generation of outlines for a session, set this parameter to `'FALSE'`.

To enable automatic generation of outlines system wide you issue the command:

```
ALTER SYSTEM SET CREATE_STORED_OUTLINES = TRUE;
```

This works similar to the session setting. If set to `'TRUE'`, outlines are created in the `DEFAULT` category. The value `'TRUE'` can be substituted for a category name to create outlines in specific categories. When generating outlines for all eligible SQL statements system wide, only the user sessions that have the `'create any outline'` privilege will generate store outlines. It is possible to `'grant create any outline to public'` and have outlines generated for all eligible SQL in all user sessions in the system. Generally speaking, creating outlines for all SQL statements in all user sessions system wide in a production database is not a good idea. If the system is busy, you can quickly generate a tremendous amount of data being inserted into the Oracle data dictionary. When setting this parameter system wide, you can specify whether or not you want the system setting to over-ride the session setting for this parameter. By default, the system setting takes effect in all sessions. To preserve individual session settings, use the `NOOVERDIDE` option:

```
ALTER SYSTEM SET CREATE_STORED_OUTLINES = TRUE NOOVERRIDE;
```

When automatically creating store outlines you can not control outline names. Oracle will automatically generate outline names for you.

How Oracle Matches SQL Statements with Outlines

Oracle relies on its "exact text matching" technique when trying to match SQL statements to SQL text stored in outlines. This is the same technique that is used for cursor matching in the shared memory pool. Oracle will not match the SQL text to the outline SQL text if there are any differences including, text case, embedded white space, carriage return variations, and even comment text differences including hints.

There is a one to one correspondence between an SQL statement and its stored outline. If literal values are used in an SQL statement predicate, then the outlines must be created with the same literal values in order for the SQL text to match an outline. This same issue exists for matching SQL text in the shared memory pool. For outlines to be effective, SQL statements should be using bind variables whenever possible.

Using Outlines

No privileges or specific grants are required to use stored outlines. To enable the use of stored outlines you must set the parameter `'USE_STORED_OUTLINES'` to `'TRUE'`. The value `'TRUE'` can be substituted for a category name. If it is set to `'TRUE'`, Oracle uses outlines in the `'DEFAULT'` category only. If you specify a category name, outlines in the specified category name will be used and if a matching outline can not be found in the specified category, the `'DEFAULT'` category will be searched for a matching outline. The use of outlines can be set for a single session or system wide. To use outlines for a single session user session, issue the command:

```
ALTER SESSION SET USE_STORED_OUTLINES = TRUE;
```

The use of stored outlines can be disabled by setting the 'USE_STORED_OUTLINES' parameter to 'FALSE':

```
ALTER SESSION SET USE_STORED_OUTLINES = FALSE;
```

By default, when setting the 'USE_STORED_OUTLINES' parameter system wide with the 'ALTER SYSTEM' statement, all sessions will search for and use stored outlines. The 'NOOVERRIDE' parameter can be used to prevent the system setting from overriding the session setting:

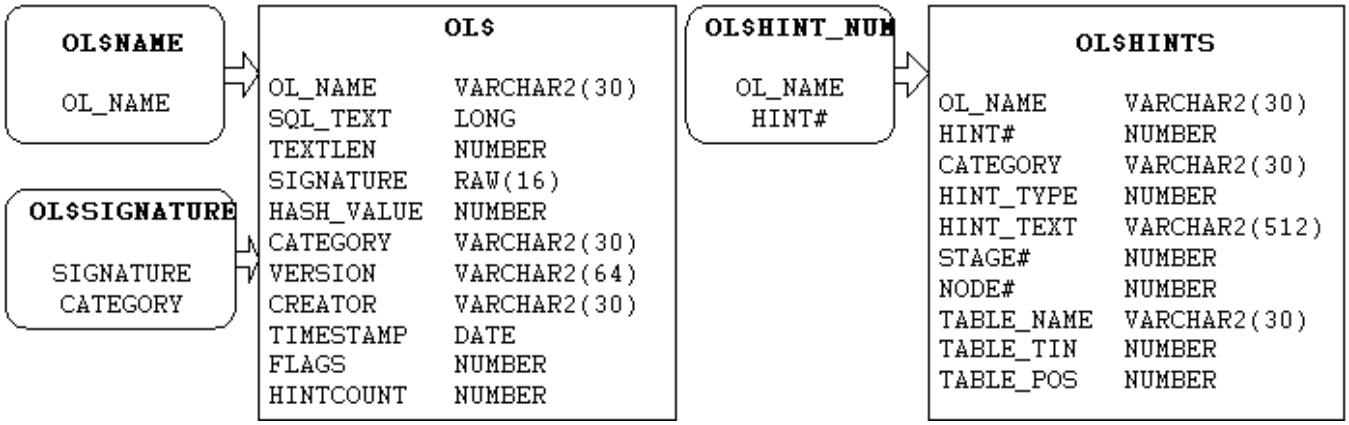
```
ALTER SYSTEM SET USE_STORED_OUTLINES = scott_outlines NOOVERRIDE;
```

Managing Outlines

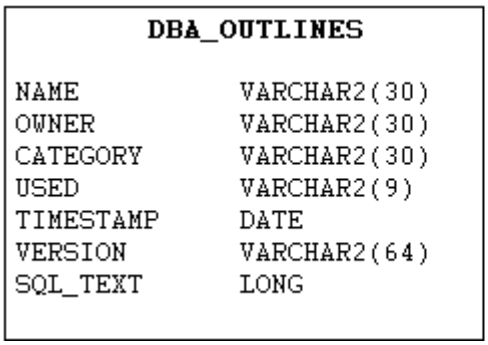
Querying Outline Information

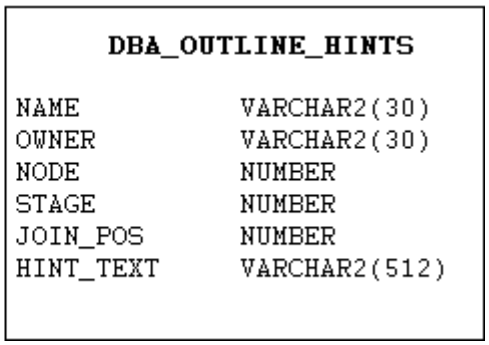
When an Oracle outline is created, Oracle does not actually store the execution plan in the data dictionary. A series of Oracle hints and the SQL text are stored in the data dictionary. Stored outlines are not what I would consider normal database objects because there is no reference to them in the dba_objects view. Outline information is stored in two new Oracle data dictionary tables OL$ and OL$HINTS.

These tables are owned by a schema named OUTLN. This is a new schema that is setup automatically when an Oracle8iinstance is created via the catol.sql and dbmsol.sql scripts which are called from the main catalog.sql and catproc.sql scripts. By default, the schema password for the OUTLN schema is OUTLN. This password should be changed after a new instance is created. By default the OUTLN is granted the CONNECT and RESOURCE roles and the EXECUTE ANY PROCEDURE and UNLIMITED TALESPACE system privileges.



Oracle has also created some views to query information from these outln tables.





There is a user_outlines and user_outline_hints views that are filtered by owner. All_outlines and all_outline_hints views are provided but they are identical to the user_ views.

The following query will display the SQL text from the outline information created with our sample SQL statement on the scott/tiger tables:

```
select sql_text from dba_outlines where name = 'SCOTT_SELECT_1';
```

The sql_text column is a long data type and the case, comments, and white space in the SQL statement are preserved. You may have to use the `SET LONG` sqlplus command to increase the maximum display length for long data types. The default is 80 characters.

The following sqlplus query displays the outline hints for our sample SQL statement:

```
column node# format 999
column stage# format 999
column hint_text format a35
column table_name format a30
SELECT node#, stage#, hint_text, table_name
  FROM outln.ol$hints
 WHERE OL_NAME = 'SCOTT_SELECT_1'
 ORDER BY node#,stage#,table_tin,table_pos;
```

OUTPUT:

```
NODE# STAGE# HINT_TEXT                          TABLE_NAME
----- ------ ---------------------------------- ---------------
    1      1 NOREWRITE
    1      1 RULE
    1      2 NOREWRITE
    1      3 NO_EXPAND
    1      3 ORDERED
    1      3 NO_FACT(EMP)                       EMP
    1      3 FULL(EMP)                          EMP
    1      3 USE_NL(DEPT)                       DEPT
    1      3 NO_FACT(DEPT)                      DEPT
    1      3 INDEX(DEPT PK_DEPT)                DEPT
    2      1 NOREWRITE
    2      2 NOREWRITE
    2      3 NO_EXPAND
          2      3 ORDERED
          2      3 NO_FACT(SALGRADE)                  SALGRADE
          2      3 FULL(SALGRADE)                     SALGRADE
```

As you can see, the outline information is a series of Oracle hints, it is not really the execution plan for the statement. You can get a sense of the execution flow, but it shouldn't be used as a definite reference for the execution plan. If we explain our example SQL statement, the execution plan looks like:

```
SELECT STATEMENT    Current Optimizer Mode = CHOOSE  Cost =
  1.1 FILTER
    2.1 NESTED LOOPS
      3.1 TABLE ACCESS FULL EMP
      3.2 TABLE ACCESS BY INDEX ROWID DEPT
        4.1 INDEX RANGE SCAN PK_DEPT NON-UNIQUE
    2.2 TABLE ACCESS FULL SALGRADE
```

Even if you have enabled the use of stored outlines, the explain plan facility will not check for and use matching outlines to build execution plans. In other words, the explain plan facility ignores all outline information.

The following sqlplus script will give you a summary report for all outlines in the system, listing the number of outlines, and number of used outlines, and total number of hints grouped by category.

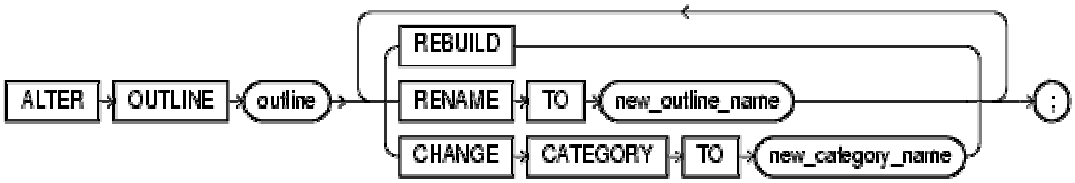
```
break on report
compute sum of "Outlines" on report
compute sum of "Outlines Used" on report
compute sum of "Hint Count" on report
SELECT category "Category",
       COUNT(ol_name) "Outlines",
       sum(flags) "Outlines Used",
       sum(hintcount) "Hint Count"
  FROM outln.ol$
 GROUP BY category;
```

Trying to determine if an outline exists for a given SQL statement can be difficult because of the long data type on the `SQL_TEXT` column. You can try searching the dba_outline_hints view and query the hint_text column for specific table and index names, but this can be tedious if you are trying to find a specific SQL statement. The following stored procedure will allow you to do text searches within the first 200 characters of the `SQL_TEXT` column in the `dba_outlines` view.

```
CREATE OR REPLACE PROCEDURE find_sql (s_string IN VARCHAR2)
IS
TYPE string_table IS
   TABLE OF VARCHAR2(200) INDEX BY BINARY_INTEGER;
o_name          VARCHAR2(30);
c_name          VARCHAR2(30);
long_sql_string LONG;
char_sql_string VARCHAR2(200);
sql_count       NUMBER := 0;
match_count     NUMBER := 0;
found_flag      NUMBER;
CURSOR select_sql_text IS
   SELECT name, category, sql_text FROM dba_outlines;
BEGIN
DBMS_OUTPUT.ENABLE(1000000);
OPEN select_sql_text;
LOOP
   FETCH select_sql_text INTO o_name, c_name, long_sql_string;
   EXIT WHEN select_sql_text%NOTFOUND;
   char_sql_string := SUBSTR(long_sql_string,1,200);
   found_flag := INSTR(char_sql_string,s_string);
   IF found_flag <> 0 then
      DBMS_OUTPUT.PUT_LINE('Found in '||o_name||' category '||c_name);
      match_count := match_count +1;
   END IF;
   sql_count := sql_count + 1;
END LOOP;
DBMS_OUTPUT.PUT_LINE('SQL statements found > '||match_count);
DBMS_OUTPUT.PUT_LINE('Total SQL statements searched > '||sql_count);
CLOSE select_sql_text;
END;
```

Altering Outlines

Outlines can be altered through the alter command. The `ALTER ANY OUTLINE` system privilege must be granted to a user before they can alter an outline.



The `REBUILD` option will regenerate an execution plan for an SQL statement using the current conditions to generate a new set of hints for the SQL statement.

The `RENAME TO` option is used to re-name an outline. The outline name must be unique.

The `CHANGE CATEGORY TO` option is used to reassign an outline to a new or existing category. You can not change an outlines category to the default category.

Altering an outline does not reset the `USED` flag to `UNUSED`.

To assist with reassigning all outlines in one category to another, Oracle has supplied a stored procedure that can be used. Before a user can execute this procedure, they must be granted the execute privilege for the `OUTLN_PKG` package.

```
execute outln_pkg.UPDATE_BY_CAT('old_category','new_category');
```

Dropping Outlines

Outlines can be dropped via the `DROP` command. A user must be granted the `DROP ANY OUTLINE` system privilege before they can drop outlines.

Dropping groups of outlines can be tedious. To assist, Oracle has supplied new stored procedures that can be used to help drop outlines. Before a user can run these procedures, they must be granted execute privileges for the OUTLN_PKG package. The following procedure can be used to drop all unused outlines in all categories:

```
execute outln_pkg.drop_unused;
```

The following can be used to drop all outlines in a specific category:

```
execute outln_pkg.drop_by_cat('category_name');
```

This procedure does not provide any feedback, so you may want to check that it did drop what you expected. It does not change the parameter value category_name to upper case, so ensure that you use upper case when specifying the category name.

Monitoring Outline Usage

You can determine if an outline has been used by checking the `'USED'` column in the `dba_outline` view. Unfortunately, you can not determine the last date used, the number of times it has been used, or if it is currently being used in the dba_outline view.

You can determine if an outline has been used for a given SQL statement in the shared memory pool by checking the value of a new column called `OUTLINE_CATEGORY` in the `V$SQL` view. If this column is null, then no outline has been used; otherwise, it will contain the name of the category for the outline used. This is a little awkward, it would be more useful if the outline name was listed. The sql_hash_value can be used to join into the `OL$` table to find the outline being used.

This SQLPLUS query will list all statements in the shared pool that are using outlines:

```
set pages 200
column sql_text format a40 word_wrapped
column outline_category format a18
column ol_name format a18
SELECT v.outline_category,
             o.ol_name,
             v.sql_text
        FROM v$sql v,
             outln.ol$ o
       WHERE o.hash_value = v.hash_value
         AND o.category = v.outline_category;
```

It is important to check for the use of stored outlines when explaining SQL statements from the V$SQL view. When you explain an SQL statement, the explain plan facility does not check for the existence of an outline. Therefore, if an outline has been used, the current execution plan generated by the explain plan facility may not be the actual execution plan in use.

If outlines are in use on your system, be especially careful when monitoring active SQL statements. If the use of stored outlines has been only been enabled for some sessions, it is possible to see two versions of the same SQL statement in the V$SQL view (One for use with the stored outline, one for use without the stored outline). The issue is that both statements will have the same hash_value. If one user is currently running the one with the outline and the other user is currently running the one without the outline, it can be difficult to determine which user is running which SQL statement.

Managing and moving the Outline data

The OL$ and OL$HINTS OUTLN tables by default are created in the system tablespace. Before you create too many outlines, it is a good idea to move these tables out of the system tablespace and into their own tablespace. The OL$HINTS table can get quite large. Our sample SQL statement only generates a 6 line execution plan; but when an outline is created, it inserts 16 rows in the OL$HINTS table. To move these tables follow this procedure:

1. Export the outln tables… `EXP OUTLN/OUTLN FILE=OUTLN.DMP TABLES='OL$' 'OL$HINTS'`
2. Drop the OUTLN tables, from SQLPLUS connect as outln/outln… `DROP TABLE OL$; DROP TABLE OL$HINTS;`
3. Revoke space privileges from the OUTLN user… `REVOKE UNLIMITED TABLESPACE FROM OUTLN;`
4. Create a new tablespace… `CREATE TABLESPACE OUTLINE_TS DATAFILE 'outline_ts.dbf' SIZE 50M;`
5. Give space quota to the OUTLN user… `ALTER USER OUTLN QUOTA UNLIMITED ON OUTLINE_TS;`
6. Change the default tablespace for OUTLN… `ALTER USER OUTLN DEFAULT TABLESPACE OUTLINE_TS;`
7. Import tables… `IMP OUTLN/OUTLN FILE=OUTLN.DMP TABLES='OL$' 'OL$HINTS'`

The export/import utility can also be use to move/copy outline information from one instance to another. Using the query-based export option, you can filter the exported outlines based on outline name or category name. Here is an example of how you would export all outlines for a category from the UNIX command line:

```
EXP OUTLN/OUTLN FILE=SPECIAL.DMP TABLES='OL$' 'OL$HINTS'
            QUERY=\"WHERE CATEGORY=\'SCOTT_OUTLINES\'\"
```

Because the outln tables will already exist when importing the outlines into a system with existing outlines, use the ignore=y import parameter. Using this technique, you can work on tuning an SQL statement in one Oracle instance then, when you are happy with the execution plan, you can generate the outline, export it and import it into your production instance.

Other Uses for Plan Stability

The Plan stability feature provides us with a nice facility to build a repository of SQL statements. Even if you do not intend to use stored outlines you may still create them and capture your application SQL statement text. This information can be useful when you wan to make schema changes. You can search through your SQL_TEXT to find SQL statements that reference a particular table or view in order to analyze the impact of schema changes. The dba_outline_hints table can be useful to determine which indexes are in use on a particular table.

DBAs deal with many different third party software packages that run on Oracle. Many of these software packages encounter performance problems that lead to countless discussions with software vendors about performance. In most cases the DBA's hands are tied because they can not change the SQL statement text. The plan stability feature should become very popular with many of the third party software vendors. They will be able to tune their application SQL in-house and then provide their clients with outlines for the applications SQL statements. This may help guarantee that the vendor's SQL statements run with the desired execution plans independent of the client's environment and the oracle optimizer

Summary

Oracle has finally realized that the Oracle optimizer does not always construct the best execution plan for a given SQL statement. Up until now, your choices were limited to running the Oracle database optimizer in either RULE or COST based mode. Those of us that have made the transition to the COST based optimizer have had to deal with the sensitivity of the Oracle optimizer to environmental changes and as a result, some performance issues. Usually we have ended up manually tuning our poor performing SQL statements with the addition of hard coded Oracle hints. The plan stability feature allows us to make the transition to the COST based optimizer in a more controlled way. It allows us to preserve the performance characteristics for critical SQL statements without having to code Oracle hints, while allowing the optimizer to make appropriate execution plan changes to improve performance for other SQL statements. Use of the plan stability feature can now allow DBAs to adjust different environment settings without fear of negatively effecting the performance of a production application's key SQL statements. Finally… Oracle plan stability.

Bibliography

"ORACLE8*i* SQL Reference Release 2 (8.1.6) December 1999"

"ORACLE8*i* Designing and Tuning for Performance Release 2 (8.1.6) December 1999"