EMC²
where information lives®

White Paper

Optimizing Oracle for EMC Documentum

Best Practices Planning

Abstract

This white paper provides guidance on optimizing Oracle for EMC® Documentum®. It includes some configuration recommendations, diagnosing and tuning methods, and comprehensive case studies.

December 2008

Part Number h5962

Table of Contents

Executive summary

Having many EMC® Documentum® deployments on an Oracle database puts a heavy workload on Oracle. In order to ensure Documentum has good performance, it's important to make sure Oracle is well optimized.

There are many approaches to optimize an Oracle database. This paper describes some of them, and provides some real case studies.

Introduction

The purpose of this white paper is to provide initial configuration recommendations and a tuning guide for Documentum deployments on the Oracle RDBMS.

Some recommendations are general for optimizing Oracle, while most of them are specific for Documentum. This paper is not Documentum version-specific, and should act as a recommendation for a starting point only. Please note that Oracle contents are mostly for 9i/10g or later.

This paper includes some specific case studies from the EMC/Documentum Performance Field Support team. They have had valuable experience resolving Oracle performance problems for Documentum.

Audience

This white paper is intended for database administrators and customer support personnel.

Configuration recommendations

Data tablespaces

Documentum creates two tablespaces when installing a repository on Oracle. One is for table data, another one is for index data. On Oracle 9i or later, the tablespaces should be created as a Local Managed Tablespace (LMT) with Automatic Segment Space Management (ASSM).

LMT

Locally Managed Tablespace (LMT) is a type of tablespace where extents are managed in the tablespace's header. It replaces Dictionary Managed Tablespace (DMT) because LMT reduces contention, eliminates undo, and provides less fragmentation. It is recommended by Oracle. LMT was introduced in Oracle 8i and becomes the default when you create tablespaces on 9i or later.

ASSM

Automatic Segment Space Management (ASSM) is a method used by Oracle to manage space inside data blocks. It traces spaces in a block using bitmap, instead of Freelist. It eliminates the need to specify parameters like PCTUSED, Freelists and Freelist groups for objects created in the tablespace. ASSM is introduced in Oracle 9i and is enabled by default in 10gr2.

Documentum includes dm_CreateTableSpace.sql to create the tablespaces, but this SQL script doesn't explicitly utilize LMT or ASSM. This script could be modified like the following statement, to enable these features when creating tablespaces.

```
CREATE TABLESPACE ts1
DATAFILE '/app/orafata/ts1.dbf' SIZE 10M
EXTENT MANAGEMENT LOCAL        -- Enable LMT
```

```
SEGMENT SPACE MANAGEMENT AUTO -- Enable ASSM
```

Undo tablespaces

Automatic Undo Management (AUM) is recommended in Oracle 9i or later. The advantage of using AUM is that it relieves the DBA of manually creating, sizing, and monitoring the rollback segments in the database.

To start using AUM you must create an UNDO-type tablespace and set some initialization parameters.

Creating an Undo Tablespace

```
CREATE UNDO TABLESPACE undots1
DATAFILE '/app/orafata/ts1.dbf' SIZE 100M AUTOEXTEND ON;
```

Parameters about Undo Management

- UNDO_MANAGEMENT (auto | manual) — The default is manual in Oracle 9i. It must be set to auto to enable AUM.
- UNDO_TABLESPACE — Specifies which undo tablespace to use.
- UNDO_RETENTION (time in seconds) — Specifies how much undo data in past seconds should be retained.

Dedicated connection

Oracle has two connection modes: dedicated and shared server. In dedicated mode, one server process is allocated per Oracle connection, while in shared server mode multiple connections may use one server process. Dedicated mode is recommended for Documentum, since Documentum in shared server mode does not perform as well as when it is in dedicated mode.

Dedicated mode is the default. Unsetting the SHARED_SERVERS parameter or setting it to 0 ensures Oracle in dedicated mode.

SGA_TARGET

SGA_TARGET is a database initialization parameter (introduced in Oracle 10g) that can be used for automatic SGA memory management. SGA_TARGET decides the total size of the SGA. You set it to a non-zero value, and then the individual pools (Shared Pool, Buffer Pool, Large Pool, Java Pool, etc.) within the SGA will be dynamically configured and tuned based on the workload.

SGA_TARGET is recommended for Documentum. To determine the appropriate setting of SGA_TARGET, the following steps are suggested.

1. Set SGA_TARGET to <Total Physical Memory> * 80%. Use 80% as a first estimate.

2. Tune SGA_TARGET using SGA Advisor, which is accessible at v$sga_target_advice or in Oracle Enterprise Manager.

In Oracle 9i, you manually set individual pool sizes, monitor their usage, and then tweak their size according to event statistics. Moreover, you can get separate advices to individual pools from vdb_cache_advice, vshared_pool_advice, and v$java_pool_advice.

PGA_AGGREGATE_TARGET

PGA_AGGREGRATE_TARGET specifies the target aggregate PGA memory available to all server processes attached to the instance. With WORKAREA_SIZE_POLICY=AUTO (which is the default), Oracle automatically and dynamically adjusts the sizes of memory areas, such as sort_area_size, hash_area_size, bitmap_merge_size, etc., in the user session.

PGA_AGGREGATE_TARGET was introduced in Oracle 9i.

The following steps are recommended to determine the appropriate PGA_AGGREGATE_TARGET.

1. Set PGA_AGGREGATE_TARGET to <Total Physical Memory> * 80% * 20% as a first estimate.

2. Tune PGA_AGGREGATE_TARGET using PGA Advisor, which is accessible at v$pga_target_advice or in Oracle Enterprise Manager.

To be able to set this appropriately (without underallocation or overallocation), get the PGA stats and PGA advice as follows (it is in the AWR report or the statspack report):

```
select name, value from v$pgastat;

select round (pga_target_for_estimate/1024/1024) Target,
estd_pga_cache_hit_percentage cache_hit_percentage,
estd_overalloc_count
from v$pga_target_advice;
```

The parameter should be set to a value where estd_overalloc_count is 0.

PROCESSES

The PROCESSES parameter specifies the maximum number of operating system user processes that can simultaneously connect to Oracle. Its value should allow for all background processes such as locks, job queue processes, and parallel execution processes.

Since each Content Server (DMCL) session may occupy one to two Oracle sessions in Documentum's architecture, it is recommended to set PROCESSORS to at least two times the concurrent_sessions value in Documentum's server.ini file.

OPTIMIZER_MODE

OPTIMIZER_MODE defines the behavior of the optimizer.

RULE, CHOOSE, and FIRST_ROWS modes are deprecated in Oracle 10gr2. You would only use FIRST_ROWS_n or ALL_ROWS modes.

The optimizer in RULE, CHOOSE, or FIRST_ROWS modes may use heuristics to find a best plan, while sometimes using heuristics leads the optimizer to pick a plan with a significantly large cost.

ALL_ROWS favors minimum resource usage for returning all rows while it often has a bit longer response time.

FIRST_ROWS_10 is recommended for Documentum. In this mode the optimizer uses a cost-based approach and optimizes with a goal of best response time to return the first 10 rows.

OPTIMIZER_INDEX_COST_ADJ

OPTIMIZER_INDEX_COST_ADJ lets you tune optimizer behavior for access path selection to be more or less index friendly — that is, to make the optimizer more or less prone to selecting an index access path over a full table scan.

The value range of this parameter is from 1 to 10000, and smaller means index access costs less.

The default setting of 100 is incorrect for most OLTP systems. It is recommended to set OPTIMIZER_INDEX_COST_ADJ to 5 for Documentum.

OPTIMIZER_INDEX_CACHING

OPTIMIZER_INDEX_CACHING indicates the percentage of the index blocks the optimizer should assume are in the cache. The range is between 0 and 100. The caching of the index parameter affects the cost of nested loops join or IN-list iterators using that index. Setting this parameter to a higher value makes nested loops joins and IN-list iterators look less expensive to the optimizer.

It is recommended to set OPTIMIZER_INDEX_CACHING to 95 for Documentum. The default setting of 0 is incorrect for most OLTP systems.

CURSOR_SHARING

FORCE is recommended for this parameter.

Setting CURSOR_SHARING=FORCE lets the optimizer treat the literals in every SQL statement as bind variables, so similar SQL statements can share the same SQL area and execution plan. It reduces hard parsing, library cache, or shared pool latches.

SESSION_CACHED_CURSORS

It is recommended to set this parameter to 50.

This parameter specifies the number of session cursors to cache. Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache. Subsequent parse calls find the cursor in the cache with no need to reopen the cursor. It even avoids soft parsing, so CPU usage, library cache, or shared pool latches will be reduced.

Init.ora parameters

Table 1 summarizes the recommended settings of init.ora parameters for Documentum.

Table 1. Init.ora parameters

Parameter name	Recommend settings
db_block_size	8192(default)
undo_management	AUTO
sga_target	Total physical memory * 80% * 80%
pga_aggregate_target	Total physical memory * 80% * 20%
processes	2 * concurrent_sessions in server.ini
sessions	processes * 1.1 + 5 (default)
optimizer_mode	FIRST_ROWS_10
optimizer_index_cost_adj	5
optimizer_index_caching	95
cursor_sharing	FORCE
session_cached_cursors	50

Statistics gathering

The Oracle Cost-Based Optimizer highly depends on good statistics to generate optimal plans of SQL statements. Ensure you have a good statistics-gathering mechanism.

dm_UpdateStats

Documentum provides a dm_UpdateStats job to gather database statistics. This job is run by the Agent Exec process once per week by default.

Basically dm_UpdateStats analyzes all tables in the repository with compute statistics. For good measure it also executes a script named custom_oracle_stat.sql under documentum/dba/config/<docbase>. This script includes a set of additional analyze statements, which add histograms for certain columns. Sometimes histograms and bind value peeking (especially intensified by CURSOR_SHARING=FORCE) might cause suboptimal execution plans. These cases should be treated carefully.

For most installations, running dm_UpdateStats once a week works well. If the repository changes very fast, statistics might be stale and suboptimal plans might be generated. If there are large batch jobs changing the repository, you should gather statistics immediately after the jobs finish.

Please note the compute statistics is a heavy operation, and you are advised to not run this job at the peak of database activities.

DBMS_STATS and GATHER_STATS_JOB

Since 9i, Oracle has provided a DBMS_STATS package to gather statistics. This package is recommended more by Oracle than the previous ANALYZE statement. Moreover, Oracle 10g introduced a default job named GATHER_STATS_JOB, which automatically gathers daily database statistics using DBMS_STATS.

The GATHER_STATS_JOB samples/estimates statistics with automatic size. It is adequate for most situations. It may also create a histogram for skewed data.

DBMS_STATS can be executed manually with various options. It helps gather good statistics for optimal execution plans.

DBMS_STATS.GATHER_SYSTEM_STATS gathers system I/O and CPU performance and utilization. It helps the Cost-Based Optimizer to calculate costs more accurately and generate more optimal execution plans.

STATSPACK and AWR

STATSPACK is a performance diagnosis tool, available since Oracle8i. Its report is very good for diagnosing instance-wide performance problems. It also supports application tuning activities by providing data that identifies high-load SQL statements. STATSPACK is used and maintained by a set of SQL scripts named as sp*.sql. The package STATSPACK contains its core functionalities.

AWR was introduced by Oracle 10g as STATSPACK's successor. It is more convenient and includes many new features. AWR tables that store snapshots are automatically created when creating a 10g database. An AWR snapshot is created automatically once an hour and lasts 7 days by default. AWR can be easily controlled though Oracle Enterprise Manager, and used through the DBMS_WORKLOAD_REPOSITORY package or a set of scripts named as awr*.sql.

The STATSPACK and AWR reports are pretty similar. The AWR/STATSPACK reports should be made for short-term intervals such as 15-30 minutes during a busy or peak time, when performance is at its worst. This will provide a very focused look at what was going wrong at that exact moment in time.

Remember to set TIMED_STATISTICS to true for your instance. Setting this parameter provides timing data that is essential for the following time-based tuning method.

Check Oracle documentation for more details on using AWR/STATSPACK.

Time-based tuning

Using AWR/STATSPACK, the Oracle instance's performance statistics can be clearly organized into a comprehensive report.

To know whether an instance is healthy, the best way is to look at the Top Timed Events. It shows the instance's bottleneck for such a time period. Starting from the most time-consuming wait events, you can do root-cause analysis or instance tuning. The following are some common wait events and some general tuning tips.

- DB File Scattered Read

  This generally indicates waits related to full table scans. Check SQL with the top physical reads.

- DB File Sequential Read

  This event generally indicates a single block read (an index read, for example). A large number of waits here indicates poor joining orders of tables, or unselective indexing. Check SQL with the top physical reads.

- Free Buffer

  This event is waiting for a buffer in memory, because none is currently available. It indicates slow DBWR or small buffer cache, if all SQLs are tuned.

- Buffer Busy

  This is a wait for a buffer that is being used in an unshareable way or is being read into the buffer cache. Find out what causes the hot blocks, then fix it accordingly.

- Latch Free

  This event means high contentions on various internal memory structures. It often indicates heavy hard parsing (library cache latch), redo generation issues (redo allocation latch), buffer cache contention (cache buffers LRU latch), hot blocks in the buffer cache (cache buffer chain), etc.

- Enqueue

  It indicates heavy locking in application or internal structures such as index, bitmap index, segment header, TLT slot, etc.

- Log Buffer Space

  This wait occurs because you are writing the log buffer faster than LGWR can write it to the redo logs, or because log switches are too slow. Consider increasing the log file size, or get faster disks.

- Log File Switch

  The "logfile switch (archiving needed)" message indicates ARCH is too slow. The "logfile switch (Checkpoint. Incomplete)" message indicates DBWR may be too slow because of I/O. Improve the I/O performance, or use more or larger redo logs.

- Log File Sync

  This indicates the database commits or rollbacks too fast, which LGWR does not handle. Try batch commits, or improve I/O performance. Do not use RAID 5, since it is very slow for applications that write a lot; potentially consider using file system direct I/O or raw devices, which are very fast at writing information.

Get an execution plan of top SQL

When you find the top SQL from the AWR/STATSPACK report, use the following script to check its plan statistics.

STATSPACK

```
SQL> @?/rdbms/admin/sprepsql.sql
Enter the plan hash value
```

AWR

```
SQL> @?/rdbms/admin/awrsqrpt.sql
Enter begin/end snapshot ids.
Enter the SQL id.
```

Also, DBMS_XPLAN.DISPLAY_AWR(SQL_ID) can display a history plan of SQL in AWR.

ADDM in Oracle 10g

The Automatic Database Diagnostic Monitor (ADDM) is a great tool to diagnose Oracle problems. ADDM analyzes AWR data on a regular basis, then locates the root causes of performance problems, provides recommendations for correcting any problems, and identifies non-problem areas of the system.

You can view ADDM analysis using Oracle Enterprise Manager or sqlplus. Oracle documentation provides more details.

SQL trace

How do you identify bad queries? AWR, STATSPACK, or ADDM can help, but they are mostly at the instance level, and some queries may not be discovered. SQL trace is more appropriate to troubleshoot a session with bad performance. The following are some ways to get SQL trace for Documentum.

SQL_TRACE of DMCL

Documentum can trace all SQL statements executed on the underlying RDBMS in a DMCL session. With the timestamps of trace lines, you can figure out which query elapses too much time.

Using the ‘trace’ server api in IAPI

```
API>trace,c,10,c:\api.log,SQL_TRACE
```

This command turns on SQL trace for the current session, with a severity level to 10. It generates a client API method called trace into c:\api.log, and also puts SQL trace into its session log, which is located in $DOCUMENTUM/dba/log/<hex_repository_id>/username.

Using the ‘apply’ server api in IAPI

```
API>apply,c,NULL,SQL_TRACE,LEVEL,I,1
```

This command puts SQL trace into the current session log.

Adding –osqltrace to Documentum's start command line

```
C:\Documentum\product\5.3\bin\documentum.exe -docbase_name docbase -
osqltrace
```

This turns on SQL_TRACE for all DMCL sessions connected to the repository.

SQL_TRACE (or 10046 Event Trace) of Oracle

Extended SQL tracing means turning on 10046 event trace with level 12, or SQL_TRACE with binds and waits. These two are equal.

The following is a basic sql trace, which does not include either waits or binds.

alter session set sql_trace = true

Extended SQL tracing provides the most detail for SQL execution in an Oracle session. It is recommended.

SQL trace a specific oracle session

Log in to Documentum Administrator, and go to Administration > Sessions. Find out the Oracle session id of the session you want to trace. Then go to sqlplus.

```
SQL> select sid, serial#, username, process, machine, terminal, program
from v$session where sid = <sid>
```

In Oracle 10g:

```
SQL> exec dbms_monitor.session_trace_enable(session_id=><sid>,
serial_num=><serial#>, waits=>TRUE, binds=>TRUE);
```

In Oracle 9i:

```
SQL> exec dbms_system.set_ev(<sid>,<serial#>,EV=>10046,LE=>12,NM=>'');
```

To turn off extended SQL trace:

```
SQL> exec dbms_monitor.session_trace_disable(session_id=><sid>,
serial_num=><serial#>)
```

Or:

```
SQL> exec dbms_system.set_ev(<sid>,<serial#>,EV=>10046,LE=>0,NM=>'');
```

SQL trace a current session in sqlplus

```
SQL> alter session set events '10046 trace name context forever, level
12';
```

To turn off:

```
SQL> alter session set events '10046 trace name context off';
```

SQL trace all docbase sessions using trigger

The following is a database logon trigger created for tracing all sessions connect to the 'DOCREPO' repository.

```
CREATE OR REPLACE TRIGGER LOGON_TRACE_DCTM
AFTER LOGON ON DATABASE
BEGIN
 if user = 'DOCREPO' then
  execute immediate 'alter session set timed_statistics=true';
  dbms_session.set_identifier('docrepo');
  execute immediate 'alter session set events ''10046 trace name context
forever, level 12''';
 end if;
END;
```

To disable/enable the trigger:

```
SQL> alter trigger LOGON_TRACE_DCTM disable/enable;
```

TRCSESS and TKPROF

Oracle generates a sql trace file (with a .trc extension) under a folder that is specified in the USER_DUMP_DEST parameter. It is $ORACLE_HOME/admin/<oid>/udump by default.

Oracle trace files are not easy to read. TKPROF analyzes a raw trace file and generate a human-readable summary report. TRCSESS aggregates selected data from multiple trace files into one file. It is very useful in a connection pool situation.

For example, the following command aggregates all sessions with the client identifier set to 'docrepo'.

```
trcsess output=docrepo.trc clientid=docrepo *.trc
```

Then you can get a report from a trace using the following command:

```
tkprof docrepo.trc docrepo.prf sys=no sort=fchela,exeela,prsela
```

The docrepo.prf has no recursive-calls, and the SQL statements were sorted by elapsed times for fetching, executing, and parsing.

Check the *Oracle Database Performance Tuning Guide 10g Release 2 (10.2)* for more detail.

SQL tuning

After we find the sql that has bad performance from AWR/STACKPACK/ADDM or SQL traces, it is tuning time. The following are some tools to tune bad sql.

AUTOTRACE

AUTOTRACE is provided by sqlplus. Turn on autotrace and execute the bad sql. After the sql is executed, sqlplus reports an execution plan and statistics of the sql.

The following is its usage:

```
SQL> set autotrace on | traceonly
SQL> just run the bad sql
```

Traceonly indicates sqlplus is not to display the result after executing the sql statement.

The following is a sample output:

```
Execution Plan
----------------------------------------------------------
Plan hash value: 272002086

--------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)|Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |     2 |     2   (0)|00:00:01|
|   1 |  TABLE ACCESS FULL| DUAL |     1 |     2 |     2   (0)|00:00:01|
--------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
         24  recursive calls
          0  db block gets
          6  consistent gets
          4  physical reads
          0  redo size
        407  bytes sent via SQL*Net to client
        396  bytes received via SQL*Net from client
```

```
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)

          0  sorts (disk)
          1  rows processed
```

You can also execute an autotrace in Oracle SQL Developer, which is a free GUI tool provide by Oracle.

DBMS_XPLAN

This is the package to output a formatted execution plan. It can display the execution plans in the plan table, plans of a cursor in memory, or plans in AWR since Oracle 10g. The following are examples of DBMS_XPLAN usage.

Display the execution plan in the plan table:

```
SQL> explain plan for <a query>;
SQL> select * from table(dbms_xplan.display);
```

Display the execution plan of a cursor in memory:

```
SQL> select * from
table(dbms_xplan.display_cursor(<sql_id>,<cursor_child_no>,null));
```

Setting <sql_id>, <cursor_child_no> to null provides the plan for the last statement you executed. The third parameter (null) means default format. v$sql contains information on the cursors in memory.

Display the actual execution stats alongside estimated plan stats for a query:

```
SQL> select /*+ gather_plan_statistics */ ...

SQL> select * from table(dbms_xplan.display_cursor(NULL,NULL,'ALLSTATS'));
```

If you use a /*+ gather_plan_statistics */ hint with a statement or set the parameter statistics_level = all, the cursor includes actual execution stats in memory. The 'ALLSTATS' format displays estimated stats and actual stats. It is useful to find stale statistics or an optimizer problem.

Display the execution plan of a history sql in AWR:

```
SQL> select * from
table(dbms_xplan.display_cursor(<sql_id>,<plan_hash_value>,<db_id>, <format>);
```

The history SQL plans are sampled into a DBA_HIST_SQL_PLAN table. The parameter plan_hash_value, db_id, and format can be null.

Understanding an execution plan

An execution plan is a set of steps that are executed in order to access or modify the data. All SQL statements are executed according to an execution plan. An execution plan can tell you why SQL has bad or good performance.

An execution plan is often outputted as a plan table, which contains the detailed steps plus each step's estimated statistics calculated by the optimizer.

The following is a sample plan output by dbms_xplan.display.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
Plan hash value: 1144560532

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |     1 |    93 |     4  (25)| 00:00:01 |
|*  1 |  FILTER                        |                     |       |       |            |          |
|   2 |   NESTED LOOPS                 |                     |     1 |    93 |     4  (25)| 00:00:01 |
|   3 |    NESTED LOOPS                |                     |     1 |    76 |     3  (34)| 00:00:01 |
|   4 |     SORT UNIQUE                |                     |     1 |    28 |     1   (0)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID| DM_FOLDER_R         |     1 |    28 |     1   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN         | D_1F20E68180000015  |     1 |       |     1   (0)| 00:00:01 |
|*  7 |     TABLE ACCESS BY INDEX ROWID| DM_SYSOBJECT_S      |     1 |    48 |     1   (0)| 00:00:01 |
|*  8 |      INDEX UNIQUE SCAN         | D_1F20E6818000010A  |     1 |       |     1   (0)| 00:00:01 |
|*  9 |    INDEX UNIQUE SCAN           | D_1F20E68180000144  |     1 |    17 |     1   (0)| 00:00:01 |
|* 10 |   TABLE ACCESS BY INDEX ROWID  | DM_ACL_R            |     1 |    34 |     1   (0)| 00:00:01 |
|  11 |    NESTED LOOPS                |                     |     1 |    83 |     2   (0)| 00:00:01 |
|* 12 |     INDEX RANGE SCAN           | D_1F20E68180000104  |     1 |    49 |     1   (0)| 00:00:01 |
|* 13 |     INDEX RANGE SCAN           | D_1F20E68180000102  |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("JK_"."OWNER_NAME"=:SYS_B_03 OR "JK_"."OWNER_NAME"=:SYS_B_04 OR  EXISTS (SELECT
             /*+ */ 0 FROM "DM_ACL_R" "ACL_R","DM_ACL_S" "ACL_S0" WHERE "ACL_S0"."OBJECT_NAME"=:B1 AND
             "ACL_S0"."OWNER_NAME"=:B2 AND "ACL_S0"."R_OBJECT_ID"="ACL_R"."R_OBJECT_ID" AND
             "ACL_R"."R_ACCESSOR_PERMIT">=TO_NUMBER(:SYS_B_12) AND (("ACL_R"."R_ACCESSOR_NAME"=:SYS_B_06
             OR "ACL_R"."R_ACCESSOR_NAME"=:SYS_B_07) OR ("ACL_R"."R_ACCESSOR_NAME"=:SYS_B_09 OR
             "ACL_R"."R_ACCESSOR_NAME"=:SYS_B_10) AND "ACL_R"."R_IS_GROUP"=TO_NUMBER(:SYS_B_08)) AND
             ("ACL_R"."R_PERMIT_TYPE"=TO_NUMBER(:SYS_B_11) OR "ACL_R"."R_PERMIT_TYPE" IS NULL)))
   6 - access("R_FOLDER_PATH"=:SYS_B_00)
   7 - filter("JK_"."I_HAS_FOLDER"=TO_NUMBER(:SYS_B_01) AND
             "JK_"."I_IS_DELETED"=TO_NUMBER(:SYS_B_02))
   8 - access("JK_"."R_OBJECT_ID"="R_OBJECT_ID")
   9 - access("JK_"."R_OBJECT_ID"="SLB_"."R_OBJECT_ID")
  10 - filter("ACL_R"."R_ACCESSOR_PERMIT">=TO_NUMBER(:SYS_B_12) AND
             (("ACL_R"."R_ACCESSOR_NAME"=:SYS_B_06 OR "ACL_R"."R_ACCESSOR_NAME"=:SYS_B_07) OR
             ("ACL_R"."R_ACCESSOR_NAME"=:SYS_B_09 OR "ACL_R"."R_ACCESSOR_NAME"=:SYS_B_10) AND
             "ACL_R"."R_IS_GROUP"=TO_NUMBER(:SYS_B_08)) AND ("ACL_R"."R_PERMIT_TYPE"=TO_NUMBER(:SYS_B_11)
             OR "ACL_R"."R_PERMIT_TYPE" IS NULL))
  12 - access("ACL_S0"."OWNER_NAME"=:B1 AND "ACL_S0"."OBJECT_NAME"=:B2)
  13 - access("ACL_S0"."R_OBJECT_ID"="ACL_R"."R_OBJECT_ID")
```

Each row represents a single step in the execution plan. Look at the columns in the plan table.

- **Id** is just the identifier of the step. It correlates with the Predicate Information.
- **Operation** indicates the operation to execute. It can be scanning a table, accessing rows from a table by using an index, joining two tables together, sorting a row set, and so on.
- **Name** is the data object to access. It can be a table or index.
- **Rows** is the estimated number of rows to return from the operation. It is also called cardinality.
- **Bytes** is the estimated size of rows to return.
- **Cost** represents units of work or resource used. It is calculated by the Cost-Based Optimizer, to determine a better plan. (%CPU) means the percentage of CPU cost in total cost.
- **Time** is the estimated timing to execute the operation.

Operations in a plan table are formatted in a hierarchy structure, which indicates the execution order. Interpret the hierarchy structure as a tree using the indentation to identify parent/child relationships. An operation is the parent of the right-indented operations at the nearest distance. A parent has at most two

children. In the previous sample, Step 1 FILTER is the parent of Step 2 NESTED LOOPS and Step 10 TABLE ACCESS BY INDEX ROWID.

Oracle traverses the binary tree of operations in postorder, to execute the plan. The basic rules are:

- The upper child is executed before the lower child.
- Children are executed before their parent.

For instance, the order of the previous sample is:

6->5->4->8->7->3->9->2->12->13->11->10->1->0.

In Documentum, watch for some operations that often have high costs:

- TABLE ACCESS FULL
- INDEX FULL SCAN / INDEX FAST FULL SCAN
- INDEX RANCE SCAN with large Rows
- HASH JOIN
- MERGE JOIN
- NEST LOOPS with large Rows of the upper (first) child
- SORT

Please note the previous described statistics (Rows, Bytes, Cost, Time) of operation are all estimated. To get the actual statistics in execution, use the gather_plan_statistics hint or set statistics_level=all, and call dbms_xplan.display_cursor with 'ALLSTATS'. Extra columns of statistics are added in the plan output. This is a great way to detect issues where the estimates look good but are actually bad, for instance, stale stats and correlated columns.

SQL Profile and SQL Tuning Advisor

SQL Profile was introduced in Oracle 10g. The optimizer provides a *tuning* mode besides the *normal* mode. Normally the optimizer operates in a very short time interval to determine an execution plan. In tuning mode, the optimizer performs additional analysis to check whether the execution plan that was produced under normal mode can be further improved. The output of the optimizer is not an execution plan, but a series of actions, along with their rationale and the expected benefit for producing a significantly superior plan.

SQL Tuning Advisor runs sql statements in tuning mode. If the optimizer found useful info such as data distribution or correlation to help it generate a superior plan, it stores this info as a SQL Profile. Then in normal mode, the optimizer can quickly generate a superior plan using the profile.

SQL Tuning Advisor is easy to use though Oracle Enterprise Manager. Often ADDM brings you to SQL Tuning Advisor. The document *Tuning poor performing SQL's Using Oracle 10g Enterprise Manager's Automatic SQL Tuning Advisor* provides more information on how to use the SQL Tuning Advisor.

It takes some resources to run a SQL Tuning Advisor task. It is best to run it during periods of low activity.

SQL Tuning Advisor and SQL Profile can also be managed using the dbms_sqltune package. SQL Profiles can be found in DBA_SQL_PROFILES.

SQL Tuning Advisor and SQL Profile are recommended for tuning SQL statements without modifying SQL statements.

Stored outline

Stored outline preserves the execution plan by using a set of hints. It was introduced in Oracle 8i. Normally, if you have a bad SQL statement, you tune it with different hints/parameters to find the best plan. Once you find the best plan, create an outline for it. The outline stores many hints that can stabilize the execution path. Then optimizer can use it when running the statement later.

Create an outline using the SQL command, for example:

```
CREATE OUTLINE emp_dept FOR CATEGORY scott_outlines

ON SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno =
d.deptno;
```

Create an outline using the **dbms_outln** package:

```
BEGIN
  dbms_outln.create_outline(
    hash_value    => 3909283366,
    child_number  => 0,
    category      => 'SCOTT_OUTLINES');
END;
/
```

To enable the outlines, enable query rewrites and indicate which outline category the instance or session should use:

```
ALTER SESSION SET query_rewrite_enabled=TRUE;
ALTER SESSION SET use_stored_outlines=SCOTT_OUTLINES;
```

You can monitor outlines from user_outlines and user_outline_hints.

An outline is another way to gain better performance without modifying the SQL statement.

The difference between outline and sql profile is the outline fixes the plan no matter if any data changes; sql profile is a bit more flexible and works well when data volume changes, and if data distributions do not change.

Case studies

In this section, we describe some customer cases that were related to database performance.

Database performance issues manifest in many ways in customer environments: poor application performance, high memory/CPU utilization, stuck threads on application servers, and so on. A careful study of some of the following information helps in identifying performance issues at the database level.

- Database environment details – new or upgraded database, physical memory, processor/CPU, etc.
- Oracle initialization parameters
- Database statistics: how are statistics updated
- Queries generated from the poor performing operation and the volume of data involved in those queries
- Query execution context (such as job, method, user's profile, user's group memberships, etc.)
- Trace reports and query execution plans

- AWR/Statspack reports on the database
- Schema details/dump

Subsequently, an appropriate tuning option(s) can be selected, some of which follow:

- Modify Oracle Parameters as appropriate
  - Increase pga_aggregate_target, if required, based on PGA stats and PGA advisory (without underallocation or overallocation)
  - Increase sga_max_size and sga_target as necessary
  - Cursor_sharing (recommended value: FORCE or SIMILAR)
  - Optimizer_index_caching (recommended setting of 95)
  - Optimizer_index_cost_adj (recommended setting of 5, to make indexes look cheaper to the optimizer than full table scans)
  - Update database statistics
- Avoid costly full table scans with indexes
- Rewrite/Tune custom DQLs for better SQL generation
- Use DQL hints and DQL passthrough for SQL hints
- Use Materialized view (via registered tables), if there are large views in the query
- Use index partitioning for very large indexes incurring a high cost of index range scan

We present some customer case studies here, showing specific tuning solutions. It is a small set of varied cases, and we will add more cases in the future to cover a broader scenario space.

Case No.	*Problem Description*
1	A query that is being run in all sessions is causing extremely high database activity and severe performance impacts to end users
2	Query timeouts in repository - a custom query returned only 100 rows but it was taking over 4 minutes to get the results back
3	Custom DQL taking a long time for execution (4 min+), in a query using document parent-child relationships, impacting application performance
4	Performance issue on login to a DCM application for non-superusers
5	Errors encountered while running dm_DBWarning job

Case 1 (Application impact from slow queries)

Case environment: The customer was using CS 6.0 SP1 on Solaris 10, Oracle Database 10.2.0.3, and BEA WebLogic Application Server 9.2 mp2.

A query that was being run in many sessions was causing extremely high database activity and severe performance impacts to end users. The query runs were triggered by a customized import operation that applied the lifecycle to an imported document, setting some attributes in a lifecycle state, and also applying retention to the documents. In a multi-user situation with frequent uploads, the AWR report generated by the customer showed that the queries were in fact executed at a rate of 9,000/hour, each in a DFC session of its own. There were three DFC sessions created even in a simple act of stamping an attribute on an object when it enters a lifecycle state. The optimizer used some indexes and brought the query execution time

from 2+ seconds to about .25, but at a rate of 9,000 executions/hour, it added up, and the system was rendered unusable.

Also, the customer had a high number of groups (51,369 groups_s, 1,195,267 group_r) and typical users are part of a deep group hierarchy. Most of the groups were also defined to be dynamic, though no business logic required them to be defined this way.

The queries in question:

```
select distinct gs.group_name, gs.is_dynamic, gs.is_dynamic_default, gs.is_protected, gs.is_module_only
from dm_group_s gs, dm_group_r gr1, dm_group_r gr2
where gs.group_name = gr1.i_nondyn_supergroups_names and gr1.r_object_id = gr2.r_object_id
and (gr2.users_names = :P0 or gr2.groups_names = 'dm_world')
and gr1.i_nondyn_supergroups_names IS NOT NULL

select distinct gs.group_name, gs.is_dynamic, gs.is_dynamic_default, gs.is_protected, gs.is_module_only
from dm_group_s gs, dm_group_r gr
where gs.r_object_id = gr.r_object_id and gs.is_dynamic = :"SYS_B_0" and gs.is_dynamic_default = :"SYS_B_1"
and (gr.users_names = :P0 or gr.groups_names = :"SYS_B_2" or gr.groups_names in
(select distinct gs.group_name from dm_group_s gs, dm_group_r gr1, dm_group_r gr2
where gs.group_name = gr1.i_nondyn_supergroups_names and gr1.r_object_id = gr2.r_object_id
and (gr2.users_names = :P1 or gr2.groups_names = :"SYS_B_3")
and gr1.i_nondyn_supergroups_names IS NOT NULL))
```

Execution plan:

```
Plan hash value: 696337474
| Id  | Operation                           | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
|   0 | SELECT STATEMENT                    |                     |     3 |   186 |   407   (5)| 00:00:05 |
|   1 |  HASH UNIQUE                        |                     |     3 |   186 |   407   (5)| 00:00:05 |
|*  2 |   FILTER                            |                     |       |       |            |          |
|   3 |    TABLE ACCESS BY INDEX ROWID      | DM_GROUP_R          |  1323 | 23814 |    22   (0)| 00:00:01 |
|   4 |     NESTED LOOPS                    |                     |  3698 |  223K |   406   (4)| 00:00:05 |
|*  5 |      TABLE ACCESS FULL              | DM_GROUP_S          |     3 |   132 |   340   (5)| 00:00:05 |
|*  6 |      INDEX RANGE SCAN               | D_1F0163BC80000108  |  1323 |       |     1   (0)| 00:00:01 |
|*  7 |    HASH JOIN                        |                     |     1 |    70 |     5   (0)| 00:00:01 |
|   8 |     TABLE ACCESS BY INDEX ROWID     | DM_GROUP_R          |     1 |    35 |     1   (0)| 00:00:01 |
|   9 |      NESTED LOOPS                   |                     |     1 |    52 |     2   (0)| 00:00:01 |
|* 10 |       INDEX RANGE SCAN              | D_1F0163BC80000018  |     1 |    17 |     1   (0)| 00:00:01 |
|* 11 |       INDEX RANGE SCAN              | D_1F0163BC80000056  |     1 |       |     1   (0)| 00:00:01 |
|* 12 |     TABLE ACCESS BY INDEX ROWID     | DM_GROUP_R          |    65 |  1170 |     3   (0)| 00:00:01 |
|  13 |      BITMAP CONVERSION TO ROWIDS    |                     |       |       |            |          |
|  14 |       BITMAP OR                     |                     |       |       |            |          |
|  15 |        BITMAP CONVERSION FROM ROWIDS|                     |       |       |            |          |
|* 16 |         INDEX RANGE SCAN            | D_1F0163BC80000045  |       |       |     1   (0)| 00:00:01 |
|  17 |        BITMAP CONVERSION FROM ROWIDS|                     |       |       |            |          |
|* 18 |         INDEX RANGE SCAN            | D_1F0163BC80000044  |       |       |     1   (0)| 00:00:01 |
```

Resolution history:

The following recommendations were given to the customer:

1. The AWR report (the PGA Advisory part) revealed a low pga_aggregate target; so increase the pga_aggregate_target from a low value of 100m to a value where estd_overalloc_count is 0 (which was 800mb).

2. The customer was using a low value of SGA_MAX_SIZE and SGA_TARGET; hence, increase SGA_MAX_SIZE to a high value (2gb), and increase the SGA_TARGET close to SGA_MAX_SIZE.

3. The execution plans showed full table scans on high volume tables, so create indexes:
   INDEX1 : DM_GROUP_R(R_OBJECT_ID, USERS_NAMES, GROUPS_NAMES, I_NONDYN_SUPERGROUPS_NAMES)
   INDEX2 : DM_GROUP_S(IS_DYNAMIC, IS_DYNAMIC_DEFAULT, R_OBJECT_ID)

   INDEX 3: DM_GROUP_R : users_names, i_supergroups_names, r_object_id
   INDEX 4: DM_GROUP_S: r_object_id, group_class, group_native_room_id, group_name)

4. Set optimizer_dynamic_sampling=0, which switches off dynamic sampling.

5. For data modifications, make all groups non-dynamic (if there was no need for the groups to be dynamic).

Recommendations 1 to 4 did not have much effect, and recommendation 5 had a dramatic effect on performance of the query.

Case 2 (Query timeouts in the repository)

The customer faced query timeouts in the repository. A custom query, which follows, returned only 100 rows but it was taking over 4 minutes to get the results back.

Query:

```
SELECT DISTINCT benefit_state, claimant_last_name, claim_num, r_creation_date, ecm_doc_control_number,
  ecm_doc_category, doc_code,  doc_type,  adjuster_id, insured_name, nurse_extension,  nurse_last_name,
  r_object_id, object_name, packet_control_number, restricted_group, ecm_scan_datetime, service_from_date,
transmission_file_name,  subject, bill_control_number
FROM    wcmedbill_doc
WHERE   (   (  adjuster_id='7753'
               AND ( ecm_scan_datetime>=DATE('07/17/2008') AND ecm_scan_datetime <DATE('07/23/2008') )
               AND ecm_doc_category='Other Documents'
           )
           AND i_cabinet_id <> '0c009ef28000092a'
       )
ENABLE (RETURN_TOP 1000)
```

SQL query generated:

```
SELECT ALL wcmedbill_doc.benefit_state,
 wcmedbill_doc.claimant_last_name     ,
 wcmedbill_doc.claim_num              ,
 wcmedbill_doc.r_creation_date        ,
 wcmedbill_doc.ecm_doc_control_number ,
 wcmedbill_doc.ecm_doc_category       ,
 wcmedbill_doc.doc_code               ,
 wcmedbill_doc.doc_type               ,
 wcmedbill_doc.adjuster_id            ,
 wcmedbill_doc.insured_name           ,
 wcmedbill_doc.nurse_extension        ,
 wcmedbill_doc.nurse_last_name        ,
 wcmedbill_doc.r_object_id            ,
 wcmedbill_doc.object_name            ,
 wcmedbill_doc.packet_control_number  ,
 wcmedbill_doc.restricted_group       ,
 wcmedbill_doc.ecm_scan_datetime      ,
 wcmedbill_doc.service_from_date      ,
 wcmedbill_doc.transmission_file_name ,
 wcmedbill_doc.subject                ,
 wcmedbill_doc.bill_control_number
  FROM wcmedbill_doc_sp wcmedbill_doc
 WHERE ( (wcmedbill_doc.ecm_doc_category='Other Documents'
AND wcmedbill_doc.i_cabinet_id!          ='0c009ef28000092a'
AND wcmedbill_doc.adjuster_id            ='7365')
AND (wcmedbill_doc.ecm_scan_datetime    >= to_date('17/07/2008 0:0:0', 'DD/MM/YYYY HH24:MI:SS')
AND wcmedbill_doc.ecm_scan_datetime      < to_date('23/07/2008 0:0:0', 'DD/MM/YYYY HH24:MI:SS')) )
AND (wcmedbill_doc.i_has_folder          = 1
AND wcmedbill_doc.i_is_deleted           = 0)
```

Execution plan:

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |     1 |   513 |     4   (0)|
|   1 |  NESTED LOOPS                 |                     |     1 |   513 |     4   (0)|
|   2 |   NESTED LOOPS                |                     |     1 |   455 |     1   (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| WCMEDBILL_DOC_S     |     1 |   173 |     1   (0)|
|*  4 |     INDEX RANGE SCAN          | D_1F009EF280000504  |     1 |       |     1   (0)|
|*  5 |    TABLE ACCESS BY INDEX ROWID| DM_SYSOBJECT_S      |     1 |   282 |     0   (0)|
|*  6 |     INDEX UNIQUE SCAN         | D_1F009EF280000108  |     1 |       |     0   (0)|
|*  7 |   TABLE ACCESS BY INDEX ROWID | ECM_PNC_DOC_S       |     1 |    58 |     3   (0)|
|*  8 |    INDEX UNIQUE SCAN          | D_1F009EF280000500  |     1 |       |     2   (0)|
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("FI_"."ADJUSTER_ID"='7365')
   5 - filter("TE_"."I_HAS_FOLDER"=1 AND "TE_"."I_IS_DELETED"=0 AND
              "TE_"."I_CABINET_ID"<>'0c009ef28000092a')
   6 - access("TE_"."R_OBJECT_ID"="FI_"."R_OBJECT_ID")
   7 - filter("CI_"."ECM_SCAN_DATETIME">=TO_DATE('2008-07-17 00:00:00',
              'yyyy-mm-dd hh24:mi:ss') AND "CI_"."ECM_DOC_CATEGORY"='Other Documents' AND
              "CI_"."ECM_SCAN_DATETIME"<TO_DATE('2008-07-23 00:00:00', 'yyyy-mm-dd
              hh24:mi:ss'))
   8 - access("CI_"."R_OBJECT_ID"="FI_"."R_OBJECT_ID")
       filter("TE_"."R_OBJECT_ID"="CI_"."R_OBJECT_ID")
```

Execution plan from trace:

```
Rows     Row Source Operation
-------  ---------------------------------------------------
     11  SORT UNIQUE
     11   FILTER
     11    NESTED LOOPS
     11     NESTED LOOPS
  29805      TABLE ACCESS BY INDEX ROWID WCMEDBILL_DOC_S
  29805       INDEX RANGE SCAN D_1F009EF280000504 (object id 11342)
     11      TABLE ACCESS BY INDEX ROWID ECM_PNC_DOC_S
  29805       INDEX UNIQUE SCAN D_1F009EF280000500 (object id 11169)
     11     TABLE ACCESS BY INDEX ROWID DM_SYSOBJECT_S
     11      INDEX UNIQUE SCAN D_1F009EF280000108 (object id 7987)
```

Resolution history:

The query dealt with a high volume of data as revealed by the following:

SQL> select count(*) from wcmb01p.dm_sysobject_s => 6277270

SQL> select count(*) from wcmb01p.ecm_pnc_doc_s => 6222575

Recommendations:

1. Row selectivity is suspect, and hence new indexes should be added:

   dm_sysobject_s (i_cabinet_id, i_has_folder, i_is_deleted)
   ecm_pnc_doc_s (ecm_scan_datetime, ecm_doc_category)

2. It was observed that the following parameters were not in line with Documentum recommendations. The PGA advisory (v$pga_target_advice) recommended a setting of 1152mb for pga_aggregate_target; hence set the following for the Oracle instance:

CURSOR_SHARING = FORCE
PGA_AGGREGATE_TARGET > 900mb

Recommendations 1 and 2 resulted in the following plan:

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |     1 |   518 |     1 (100)|
|   1 |  HASH UNIQUE                   |                     |     1 |   518 |     1 (100)|
|*  2 |   TABLE ACCESS BY INDEX ROWID  | WCMEDBILL_DOC_S     |     1 |   173 |     0   (0)|
|   3 |    NESTED LOOPS                |                     |     1 |   518 |     0   (0)|
|   4 |     NESTED LOOPS               |                     |     1 |   345 |     0   (0)|
|   5 |      TABLE ACCESS BY INDEX ROWID| DM_SYSOBJECT_S      |     1 |   282 |     0   (0)|
|*  6 |       INDEX SKIP SCAN          | INDEX2              |     1 |       |     0   (0)|
|*  7 |      TABLE ACCESS BY INDEX ROWID| ECM_PNC_DOC_S       |     1 |    63 |     0   (0)|
|*  8 |       INDEX RANGE SCAN         | INDEX1              |     1 |       |     0   (0)|
|*  9 |     INDEX RANGE SCAN           | D_1F009EF280000504  |     1 |       |     0   (0)|
-----------------------------------------------------------------------------------------------
 </p>
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("TE_"."R_OBJECT_ID"="FI_"."R_OBJECT_ID" AND
              "CI_"."R_OBJECT_ID"="FI_"."R_OBJECT_ID")
   6 - access("TE_"."I_HAS_FOLDER"=1 AND "TE_"."I_IS_DELETED"=0)
       filter("TE_"."I_HAS_FOLDER"=1 AND "TE_"."I_IS_DELETED"=0 AND
              "TE_"."I_CABINET_ID"<>'0c009ef28000092a')
   7 - filter("TE_"."R_OBJECT_ID"="CI_"."R_OBJECT_ID")
   8 - access("CI_"."ECM_SCAN_DATETIME">=TO_DATE('2008-07-17 00:00:00',
              'YYYY-mm-dd hh24:mi:ss') AND "CI_"."ECM_DOC_CATEGORY"='Other Documents' AND
              "CI_"."ECM_SCAN_DATETIME"<TO_DATE('2008-07-23 00:00:00', 'YYYY-mm-dd hh24:mi:ss'))
       filter("CI_"."ECM_DOC_CATEGORY"='Other Documents')
   9 - access("FI_"."ADJUSTER_ID"='9825')
```

However, customers did not see substantial query performance improvement.

Further recommendations:

1. Use dbms_stats package instead of the Documentum dm_updatestats job and verify (as the stats generated by dm_updateStats job may not have helped the customer's case).

2. Use OPTIMIZE_TOP DQL hint (in addition to the RETURN_TOP hint used by the query) and modify the custom query as shown below:

   select <*...rest of the query..*> ENABLE (RETURN_TOP 1000, OPTIMIZE_TOP 1000)

   The OPTIMIZE_TOP N hint directs the database server to return the first N rows returned by a query quickly. The remaining rows are returned at the normal speed. Use it under the following conditions:
   - When you want to return all the results but want the first few rows more quickly
   - With the RETURN_TOP hint, to optimize the return of specified rows
   - When the execution plan chosen for the query is a bad plan and there is no obvious solution

3. Use materialized views for wcmedbill_doc_sp view, register the materialized view, and create an index (adjuster_id, ecm_scan_datetime, ecm_doc_category, i_cabinet_id).

Case 3 (Custom DQL taking long time for execution [4 min+], in a query using document parent-child relationships)

Case environment: The customer was using Documentum Content Server 5.2.5 SP2 and Oracle 9.2.0.8.

Inside a custom method invoked by an application (Webtop), the customer had an update DQL query that was taking a long time to execute. The query used document parent-child relationships. The query lasted

for about 10 minutes before listing any results, using CPU at 100% and impacting application performance overall.

This server had 8 GB memory and four CPUs; Documentum Server was running on a server with four CPUs but only two CPUs had been enabled. These two CPUs got used at 100%, resulting in some process failures too.

Query:

The update query in question is shown below:

```
qry = "update arc_facility_doc object " & _
   "set soc_date = date ('" & format$(now(), "mm/dd/yyyy") & "') " & _
   "where r_object_id in " & _
   "(select doc.r_object_id from arc_facility_doc doc, dm_relation rel " & _
   "where rel.relation_name ='Facility XMTL Attachment' " & _
   "and rel.parent_id='" & ObjectId & "' and " & _
   "((rel.child_id = doc.r_object_id and rel.child_label is nullstring) or " & _
   " (rel.child_id = doc.i_chronicle_id and any doc.r_version_label = rel.child_label)))"

rvi = dmAPIExec ("execquery," & SessionId & ",F," & qry)
 coll = dmAPIGet("getlastcoll," & SessionId)
 while (dmAPIExec("next," & SessionID & "," & coll) <> 0)
.....
```

The select clause in the query above was found to be very slow (2 to 4 minutes).

```
SELECT  ALL doc.r_object_id FROM  arc_facility_doc_sp doc, dm_relation_sp rel
WHERE
      (   rel.parent_id='09010da380079175'
          AND (
                      (
                            rel.child_label = ' '
                        AND rel.child_id    =doc.r_object_id
                      )
                  OR (
                            EXISTS (
                                    SELECT  r_object_id FROM    dm_sysobject_r
                                    WHERE   doc.r_object_id = r_object_id
                                    AND     r_version_label =rel.child_label
                            )
                        AND rel.child_id=doc.i_chronicle_id
                      )
              )
          AND rel.relation_name='Facility XMTL Obsoletes Doc'
      )
  AND ( doc.i_has_folder = 1 AND doc.i_is_deleted = 0 );
```

Additional data for the case is follows:

1. The type hierarchy for arc_facility_doc type (used in the query) was three levels deep:

   dm_document->arc_document->arc_sop_doc->arc_facility_doc

2. The Oracle initialization parameters used by the customer were:

NAME	TYPE	VALUE
cursor_sharing	string	FORCE
pga_aggregate_target	big integer	25165824
optimizer_dynamic_sampling	integer	1
optimizer_features_enable	string	9.2.0

optimizer_index_caching	integer	95
optimizer_index_cost_adj	integer	5
optimizer_max_permutations	integer	2000
optimizer_mode	string	CHOOSE

3. Statspack was installed in the database and ran every hour.

Query execution plan:

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------
| Id  | Operation                           |  Name                | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                      |     1 |   147 |     7 |
|*  1 |  FILTER                             |                      |       |       |       |
|   2 |   NESTED LOOPS                      |                      |     1 |   147 |     7 |
|   3 |    NESTED LOOPS                     |                      |     1 |   111 |     6 |
|   4 |     NESTED LOOPS                    |                      | 84649 |  7853K|     5 |
|   5 |      MERGE JOIN CARTESIAN           |                      | 84649 |  6530K|     4 |
|*  6 |       TABLE ACCESS BY INDEX ROWID   | DM_RELATION_S        |     1 |    63 |     1 |
|*  7 |        INDEX RANGE SCAN             | D_1F010DA380000024   |     5 |       |     3 |
|   8 |       BUFFER SORT                   |                      |  246K |  3847K|     3 |
|   9 |        INDEX FULL SCAN              | D_1F010DA3800001D2   |  246K |  3847K|    49 |
|* 10 |      INDEX UNIQUE SCAN              | D_1F010DA3800001D0   |     1 |    16 |       |
|* 11 |     INDEX UNIQUE SCAN               | D_1F010DA3800001CE   |     1 |    16 |       |
|* 12 |    TABLE ACCESS BY INDEX ROWID      | DM_SYSOBJECT_S       |     1 |    36 |     1 |
|* 13 |     INDEX UNIQUE SCAN               | D_1F010DA380000108   |     1 |       |       |
|* 14 |   TABLE ACCESS BY INDEX ROWID       | DM_SYSOBJECT_R       |     1 |    22 |     1 |
|* 15 |    INDEX RANGE SCAN                 | D_1F010DA380000109   |     3 |       |     3 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

   1 - filter("SYS_ALIAS_2"."CHILD_LABEL"=' ' AND "SYS_ALIAS_2"."CHILD_ID"="SYS_ALIAS_1"."R_OBJECT_ID" OR
              "SYS_ALIAS_2"."CHILD_ID"="SYS_ALIAS_1"."I_CHRONICLE_ID" AND  EXISTS (SELECT /*+ */ 0
              FROM "EDOCSPRO"."DM_SYSOBJECT_R" "DM_SYSOBJECT_R" WHERE
              "DM_SYSOBJECT_R"."R_OBJECT_ID"=:B1 AND "DM_SYSOBJECT_R"."R_VERSION_LABEL"=:B2))
   6 - filter("SYS_ALIAS_2"."RELATION_NAME"='Facility XMTL Obsoletes Doc')
   7 - access("SYS_ALIAS_2"."PARENT_ID"='09010da380079175')
  10 - access("SJC_"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID")
  11 - access("UFC_"."R_OBJECT_ID"="SJC_"."R_OBJECT_ID")
       filter("UFC_"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID")
  12 - filter("SYS_ALIAS_1"."I_HAS_FOLDER"=1 AND "SYS_ALIAS_1"."I_IS_DELETED"=0)
  13 - access("SYS_ALIAS_1"."R_OBJECT_ID"="SJC_"."R_OBJECT_ID")
       filter("SYS_ALIAS_1"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID" AND
              "SYS_ALIAS_1"."R_OBJECT_ID"="UFC_"."R_OBJECT_ID")
  14 - filter("DM_SYSOBJECT_R"."R_VERSION_LABEL"=:B1)
  15 - access("DM_SYSOBJECT_R"."R_OBJECT_ID"=:B1)

Note: cpu costing is off
```

Resolution history:

1. Increase pga_aggregate_target to a higher value (700 MB or a higher value as your pga advisory may indicate) as the current value set (25mb) is too low.

2. Add a composite index for the DM_RELATION_S columns as specified below in the same order:

DM_RELATION_S_IDX1 ("PARENT_ID", "RELATION_NAME", "CHILD_LABEL", "CHILD_ID")

Execution plan after adding indexes:

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------
| Id  | Operation                          |  Name                | Rows     | Bytes    | Cost  |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                      |      1   |    158   |      7 |
|*  1 |  FILTER                            |                      |          |          |        |
|   2 |   NESTED LOOPS                     |                      |      1   |    158   |      7 |
|   3 |    NESTED LOOPS                    |                      |      1   |    118   |      6 |
|   4 |     NESTED LOOPS                   |                      |  91146   |   8989K  |      5 |
|   5 |      MERGE JOIN CARTESIAN          |                      |  91146   |   7476K  |      4 |
|*  6 |       INDEX RANGE SCAN             | DM_RELATION_S_IDX1   |      1   |     67   |      3 |
|   7 |       BUFFER SORT                  |                      |  246K    |   4089K  |      3 |
|   8 |        INDEX FULL SCAN             | D_1F010DA3800001D2   |  246K    |   4089K  |     49 |
|*  9 |      INDEX UNIQUE SCAN             | D_1F010DA3800001D0   |      1   |     17   |        |
|* 10 |     INDEX UNIQUE SCAN              | D_1F010DA3800001CE   |      1   |     17   |        |
|* 11 |    TABLE ACCESS BY INDEX ROWID     | DM_SYSOBJECT_S       |      1   |     40   |      1 |
|* 12 |     INDEX UNIQUE SCAN              | D_1F010DA380000108   |      1   |          |        |
|* 13 |   TABLE ACCESS BY INDEX ROWID      | DM_SYSOBJECT_R       |      1   |     23   |      1 |
|* 14 |    INDEX RANGE SCAN                | D_1F010DA380000109   |      3   |          |      3 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

   1 - filter("SYS_ALIAS_2"."CHILD_LABEL"=' ' AND
              "SYS_ALIAS_2"."CHILD_ID"="SYS_ALIAS_1"."R_OBJECT_ID" OR
              "SYS_ALIAS_2"."CHILD_ID"="SYS_ALIAS_1"."I_CHRONICLE_ID" AND  EXISTS (SELECT /*+ */ 0
              FROM "EDOCSPRO"."DM_SYSOBJECT_R" "DM_SYSOBJECT_R" WHERE
              "DM_SYSOBJECT_R"."R_OBJECT_ID"=:B1 AND "DM_SYSOBJECT_R"."R_VERSION_LABEL"=:B2))
   6 - access("SYS_ALIAS_2"."PARENT_ID"='09010da380079175' AND
              "SYS_ALIAS_2"."RELATION_NAME"='Facility XMTL Obsoletes Doc')
   9 - access("SJC_"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID")
  10 - access("UFC_"."R_OBJECT_ID"="SJC_"."R_OBJECT_ID")
       filter("UFC_"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID")
  11 - filter("SYS_ALIAS_1"."I_HAS_FOLDER"=1 AND "SYS_ALIAS_1"."I_IS_DELETED"=0)
  12 - access("SYS_ALIAS_1"."R_OBJECT_ID"="SJC_"."R_OBJECT_ID")
       filter("SYS_ALIAS_1"."R_OBJECT_ID"="QRC_"."R_OBJECT_ID" AND
              "SYS_ALIAS_1"."R_OBJECT_ID"="UFC_"."R_OBJECT_ID")
  13 - filter("DM_SYSOBJECT_R"."R_VERSION_LABEL"=:B1)
  14 - access("DM_SYSOBJECT_R"."R_OBJECT_ID"=:B1)

Note: cpu costing is off
```

From this plan, it can be seen that neither the pga_aggregate_target increase nor the recommended indexes helped much. The query, due to its nature, was not significantly affected by indexes.

Further recommendation:

Rewrite SQL query as:

```
SELECT ALL doc.r_object_id
FROM arc_facility_doc_sp doc , dm_relation_sp rel
WHERE
   rel.parent_id ='09010da380079175'
      AND rel.relation_name ='Facility XMTL Obsoletes Doc'
     AND (doc.i_has_folder = 1 AND doc.i_is_deleted  = 0)
     AND (rel.child_id = doc.r_object_id and rel.child_label = ' ')
UNION
SELECT ALL doc.r_object_id
FROM arc_facility_doc_sp doc , dm_relation_sp rel
WHERE
   rel.parent_id ='09010da380079175'
   AND rel.relation_name ='Facility XMTL Obsoletes Doc'
   AND (doc.i_has_folder = 1 AND doc.i_is_deleted  = 0)
   AND  ( EXISTS
        (SELECT r_object_id FROM dm_sysobject_r
        WHERE doc.r_object_id = r_object_id AND r_version_label =rel.child_label
         )
         AND rel.child_id      =doc.i_chronicle_id
   )
```

This avoids the "OR" clause in the old query, making it less expensive. Taking this structural change to DQL, the customer was given two choices:

- Modify the DQL query used in the custom scripts to use UNION in the custom docbasic code:

```
qry = "update arc_facility_doc object " & _
  "set soc_date = date ('" & format$(now(), "mm/dd/yyyy") & "') " & _
  "where r_object_id in " & _
  "(select doc.r_object_id from arc_facility_doc doc, dm_relation rel " & _
  "where rel.relation_name ='Facility XMTL Attachment' " & _
  "and rel.parent_id='" & ObjectId & "' and " & _
  "((rel.child_id = doc.r_object_id and rel.child_label is nullstring) or " & _
  " (rel.child_id = doc.i_chronicle_id and any doc.r_version_label = rel.child_label)))"

rvi = dmAPIExec ("execquery," & SessionId & ",F," & qry)
coll = dmAPIGet("getlastcoll," & SessionId)
while (dmAPIExec("next," & SessionID & "," & coll) <> 0)
```

- Modify the docbasic script invoked by the application to use two updates instead of one:

```
------------ New Split Code ----------------
qry = "update " & RelatedDocType & " object set obsoleted_by = 'TS " & transmittal_nbr & "' " & _
  " where r_object_id in (" & _
  "select doc.r_object_id from " & RelatedDocType & " (all) doc, dm_relation rel " & _
  "where rel.parent_id = '" & ObjectId & "' and rel.relation_name = '" & RelationName & "'" &_
      "(rel.child_label is nullstring and rel.child_id = doc.r_object_id))"

rvi = dmAPIExec ("execquery," & SessionId & ",F," & qry)
coll = dmAPIGet("getlastcoll," & SessionId)
while (dmAPIExec("next," & SessionID & "," & coll) <> 0)
        call printLog ("SetObsoletedBy. objects_updated = " & _
            dmAPIGet("get," & SessionId & "," & coll & ",objects_updated"))
wend

if (coll <> "") then
    rvi = dmAPIExec("close," & SessionId & "," & coll)
end if

qry = "update " & RelatedDocType & " object set obsoleted_by = 'TS " & transmittal_nbr & "' " & _
  " where r_object_id in (" & _
      "select doc.r_object_id from " & RelatedDocType & " (all) doc, dm_relation rel " & _
          "where rel.parent_id = '" & ObjectId & "' and rel.relation_name = '" & RelationName & "'" &_
              "(any doc.r_version_label = rel.child_label and rel.child_id =
doc.i_chronicle_id))"

rvi = dmAPIExec ("execquery," & SessionId & ",F," & qry)
coll = dmAPIGet("getlastcoll," & SessionId)
while (dmAPIExec("next," & SessionID & "," & coll) <> 0)
  call printLog ("SetObsoletedBy. objects_updated = " & _
    dmAPIGet("get," & SessionId & "," & coll & ",objects_updated"))
wend
----------End of New Split Code ----------
```

The customer chose the multiple update option, and update performance improved dramatically.

Case 4 (Performance issue on login to a DCM application for non-superusers)

Case environment: The customer was using DCM 5.3 sp4, CS 5.3 sp4 on HP-UX 10.x with Oracle 9.2.0.5, and WebLogic 10 as the application server.

The customer had a performance issue on DCM login for non-superusers. A superuser could log in to the application much faster that a non-superuser. For instance, the following query (related to dcm_signoff_relation) took 1-2 seconds for the superuser while it took 16+ seconds for others. It showed up in the Statspack reports as a very high I/O consuming query.

The query involved a reasonable volume of data; the customer had 148,000 rows in the dm_sysobject_s table, and had 22,631 ACLs.

Query:

```
select all dm_document.r_object_id, dm_document.object_name, dm_repeating.r_version_label,
dm_document.owner_name,
dm_document.r_modify_date, dm_document.r_object_type, dm_document.a_content_type,
dm_document.r_lock_owner,
dm_document.r_content_size, dm_document.r_link_cnt, dm_document.r_is_virtual_doc
from gdmsprod.dm_document_sp dm_document, gdmsprod.dm_document_rp dm_repeating
where (dm_document.r_object_id in (select all appr.parent_id from gdmsprod.dcm_signoff_relation_sp appr,
gdmsprod.dm_document_sp doc
where ((doc.r_object_id=appr.parent_id) and (doc.r_current_state=appr.state_no) and
(appr.relation_name='dcm_approval') and (appr.status='pending') and ((appr.child_id='110027ed80133e75')
or (appr.acquiring_user_id='110027ed80133e75') or ((appr.acquiring_user_id='0000000000000000') and appr.child_id
in (select distinct dm_group.r_object_id from gdmsprod.dm_group_sp dm_group, (select distinct gr1.users_names as
i_all_users_names, gr2.i_supergroups_names as group_name from gdmsprod.dm_group_r gr1,
gdmsprod.dm_group_r gr2 where gr1.r_object_id = gr2.r_object_id and gr1.users_names is not NULL) gr3 where
((gr3.i_all_users_names='DocControl VC')) and gr3.group_name = dm_group.group_name )) ) ) and (doc.i_has_folder
= 1 and doc.i_is_deleted = 0) and ( ( doc.owner_name in ('DocControl
VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager')) or (exists (select 1 from gdmsprod.dm_acl_s ACL_S, gdmsprod.dm_acl_r
ACL_R where ACL_S.r_object_id = ACL_R.r_object_id and doc.acl_domain = ACL_S.owner_name and doc.acl_name
= ACL_S.object_name and ((ACL_R.r_accessor_name in ('DocControl VC','dm_world') or (ACL_R.r_is_group = 1 and
(ACL_R.r_accessor_name in ('DocControl
VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager')))) and ((ACL_R.r_permit_type = 0 or ACL_R.r_permit_type is null) and
(((ACL_R.r_accessor_permit >= 2) and ((ACL_S.i_has_access_restrictions = 0 or ACL_S.i_has_access_restrictions is
null) or (not exists (select 1 from gdmsprod.dm_acl_r ACL_R2 where ACL_R2.r_object_id = ACL_S.r_object_id and
ACL_R2.r_permit_type = 3 and (ACL_R2.r_accessor_name in ('DocControl
VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager')) and (ACL_R2.r_accessor_permit >= 1 and ACL_R2.r_accessor_permit <=
2)))))) and ((ACL_S.i_has_required_groups = 0 or ACL_S.i_has_required_groups is null) or (not exists (select 1
from gdmsprod.dm_acl_r ACL_R2 where ACL_R2.r_object_id = ACL_S.r_object_id and ACL_R2.r_permit_type = 6
and (ACL_R2.r_accessor_name not in ('DocControl
VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager'))))) and ((ACL_S.i_has_required_group_set = 0 or
ACL_S.i_has_required_group_set is null) or (exists (select 1 from gdmsprod.dm_acl_r ACL_R2
where ACL_R2.r_object_id = ACL_S.r_object_id and ACL_R2.r_permit_type = 7 and (ACL_R2.r_accessor_name in
('DocControl VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager'))))))))) )) and (dm_document.i_has_folder = 1 and dm_document.i_is_deleted =
0) and ( ( dm_document.owner_name in ('DocControl
VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-
drawings','coordinator','bl_vc_manager')) or (exists (select 1 from gdmsprod.dm_acl_s ACL_S, gdmsprod.dm_acl_r
```

ACL_R where ACL_S.r_object_id = ACL_R.r_object_id and dm_document.acl_domain = ACL_S.owner_name and dm_document.acl_name = ACL_S.object_name and ((ACL_R.r_accessor_name in ('DocControl VC','dm_world') or (ACL_R.r_is_group = 1 and (ACL_R.r_accessor_name in ('DocControl VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-drawings','coordinator','bl_vc_manager')))) and ((ACL_R.r_permit_type = 0 or ACL_R.r_permit_type is null) and (((ACL_R.r_accessor_permit >= 2) and ((ACL_S.i_has_access_restrictions = 0 or ACL_S.i_has_access_restrictions is null) or (not exists (select 1 from gdmsprod.dm_acl_r ACL_R2 where ACL_R2.r_object_id = ACL_S.r_object_id and ACL_R2.r_permit_type = 3 and (ACL_R2.r_accessor_name in ('DocControl VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-drawings','coordinator','bl_vc_manager')) and (ACL_R2.r_accessor_permit >= 1 and ACL_R2.r_accessor_permit <= 2)))))) and ((ACL_S.i_has_required_groups = 0 or ACL_S.i_has_required_groups is null) or (not exists (select 1 from gdmsprod.dm_acl_r ACL_R2 where ACL_R2.r_object_id = ACL_S.r_object_id and ACL_R2.r_permit_type = 6 and (ACL_R2.r_accessor_name not in ('DocControl VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-drawings','coordinator','bl_vc_manager'))))) and ((ACL_S.i_has_required_group_set = 0 or ACL_S.i_has_required_group_set is null) or (exists (select 1 from gdmsprod.dm_acl_r ACL_R2 where ACL_R2.r_object_id = ACL_S.r_object_id and ACL_R2.r_permit_type = 7 and (ACL_R2.r_accessor_name in ('DocControl VC','dm_world','contributor','doccontrol','roc_sol_doccontrol','roc_lens_doccontrol','consumer','tbr-sol-drawings','coordinator','bl_vc_manager')))))))) and dm_repeating.r_object_id=dm_document.r_object_id order by 2, 3

Resolution history:

The customers were asked for some specific data:

- Number of index records. There were 489 dmi_index records.

- A dump of the index D_1F0027ED800001E1 from the dmi_index table.

```
retrieve,c,dmi_index where name='D_1F0027ED800001E1'
dump,c,l

USER ATTRIBUTES
  index_type              : 030027ed8000010f
  is_unique               : T
  name                    : d_1f0027ed800001e1
  attr_count              : 0
  repeating               : F
  use_id_col              : T
  use_pos_col             : F
  attribute             []: <none>
  data_space              :
  rebuilding              : F
  use_tag               []: <none>
SYSTEM ATTRIBUTES
  r_object_id             : 1f0027ed800001e1
APPLICATION ATTRIBUTES
INTERNAL ATTRIBUTES
  i_vstamp                : 0
```

- A data dump of the following tables to investigate the query further:

```
dm_group_s, dm_group_r
dm_acl_s  , dm_acl_r
dm_sysobject_s, dm_sysobject_r
dm_document_s, dm_document_r
```

- A view definition for the dcm_signoff_relation_sp view.

Recommendation:

Add the following indexes, and update statistics:

1. Table: dm_acl_r: Columns: r_object_id, r_permit_type, r_accessor_name, r_accessor_permit
2. Table: dcm_process_relation_s: Column: status

After implementing this recommendation, the customer saw a dramatic improvement in performance, reduction in I/O, and cost of the sql executed.

Case 5 (Errors while running dm_DBWarning job)

Case environment: The customer was using Documentum Content Server 5.3 SP5 and Oracle 10g.

The customer encountered errors while running the dm_DBWarning job. An Oracle bug manifested itself while running dm_DBWarning job (Oracle Bug 6378440 QUERYING DBA_EXTENTS NEVER COMPLETES) and prevented dm_DBWarning from completing successfully.

Resolution history:
Oracle recommended turning off the new Oracle 10g hidden parameter _optimizer_cost_based_transformation:

```
_optimizer_cost_based_transformation='off'
```

According to Oracle development, setting this parameter makes the query run faster, as the parameter setting disables the cost-based *transformation*, which disables costing for features like unnesting, complex view merging, etc.

The dm_DBWarning job ran successfully after turning off the parameter.

There are no known effects on repository queries as a result of turning off this parameter. In general, Oracle seems to recommend setting _optimizer_cost_based_transformation='off' as a workaround when database upgrades lead to performance problems.

Conclusion

This paper describes some general methods to optimize an Oracle database for Documentum, and shows several particular support cases. You can also use other optimizing methods from other Oracle tuning documents, but tuning in the context of Documentum may be different.

References

The following are helpful documents about Oracle tuning for Documentum:

- *Tuning poor performing SQL's Using Oracle 10g Enterprise Manager's Automatic SQL Tuning Advisor* (FAQ)
  http://developer.emc.com/developer/downloads/FAQ_Perf_Tuning_With_Oracle_Tuning_Advisor.pdf
- *Oracle Configuration Recommendations for the Documentum Content Server*
  http://developer.emc.com/developer/downloads/oracleconfiguration.pdf
- *SQL Tuning on Oracle* (password required)
  http://beauty2.documentum.com/webtop/drl/objectId/090000018063800d
- *Oracle Database Performance Tuning Guide 10g Release 2 (10.2)*
  http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/toc.htm