Page 1
October 2005 Volume I Issue 1
ORAtipsJournal

Oracle® Application Database Management, Part 1:

Take Database Monitoring in Oracle® Applications Manager to the Next Level

By Natalka Roshak

Editor's Note: In this first of a series of ORAtips articles on Oracle Application Database management, Natalka Roshak discusses monitoring the Oracle Applications environment using the Oracle Applications Manager for SQL activity. You'll learn different approaches for executing, understanding, and using a SQL Activity Report, including using hash values and script modification. This could be very important information as not monitoring can result in runaway scripts and concurrent processes which in turn decrease application performance.

Introduction

Oracle Applications Manager (OAM) lets you monitor components of your Oracle Applications instance. This series of articles will show you how to extend Oracle Applications Manager's database monitoring capability with custom SQL queries, which can then be integrated with OAM using SQL extensions.

This article will focus on one aspect of OAM's database monitoring capability, SQL Activity. Future articles will drill down on the other aspects of OAM's database monitoring.

Why Monitor the Database?

Database performance problems are Oracle Applications performance problems. Thus, Oracle Applications Manager provides some database monitoring capabilities to let Application DBAs focus on potential trouble spots in the database. OAM lets Application DBAs monitor the following database areas:

- SQL Activity
- Runaway sessions
- Session information

OAM Database Monitoring Capability

Let's begin by locating the OAM monitoring screens that focus on the database. From the Dashboard, navigate to the Site Map. Some database monitoring screens are found under the Performance heading of the Monitoring sub-tab. From here, you can pull up reports on:

- SQL Activity
- Concurrent Request runaways

And some database monitoring screens are found under the Activity Monitors (Navigation Path: Site Map > Activity > Activity Monitors). From here, you can see some information on:

- Database sessions
- Concurrent Requests

Database performance problems are Oracle Applications performance problems.

You can also find more detailed Database Session information under the Forms Sessions tab (Navigation Path: Site Map > Monitoring subtab > Current Activity > Forms Sessions > (B) Session Details).

Focus: SQL Monitoring

This article will explain OAM's SQL report and extend it by querying the database directly.

SQL Activity Report

The Oracle Applications Monitor provides some basic information on SQL activity. To pull up the SQL Activity report, navigate from the Dashboard to the Site Map, choose the Monitoring tab, then the Performance heading. Clicking on the SQL Activity link will pull up the SQL Activity Report.

Explanation of the SQL Activity Report

The SQL Activity report has the following columns:

- SQL_HASH
- Physical Reads
- Logical Reads
- Total Sorts
- Execs
- Total Loads
- Loads

To make sense of this report, it helps to know something about how the Oracle database stores SQL statements. It takes CPU cycles to parse a SQL statement, so Oracle caches already-parsed SQL statements in memory so that the parsed version can be retrieved if the statement is reissued. For example, if a user issues an identical report request every hour, the SQL for that report will only be parsed the first time the user issues the request; on every subsequent call, the RDBMS will look up the SQL statement, find the parsed version, and use that instead of reparsing.

Oracle does this lookup using a hash of the SQL statement. The entire SQL statement is passed, as an unedited string, to a hashing function that outputs a short number. The SQL_HASH column of the SQL Activity Report shows this hash value. Two important facts about this hash value:

1. The SQL_HASH value that shows up in the SQL Activity Report is the same one used by Oracle in its data dictionary tables. Thus, we can use this value to drill down in the database for more information on the SQL statement behind that hash value.

2. When Oracle hashes a SQL statement, it uses the whole text of the SQL statement, including spaces, capitals, and literals. Thus, if a SQL statement is issued twice with different parameters, it will have two different hash values – meaning it is re-parsed, wasting CPU cycles, and will be re-cached under the new hash value, wasting memory.

Before we move into the drill-downs we can do in the database using this SQL_HASH value, let's cover the other columns available in the report.

Execs: The number of times the SQL statement represented by this SQL_HASH has been executed. This includes the executions of any child cursors required to execute this statement.

Physical Reads: The total number of disk reads the execution of this query has required, over all the times it has been executed

When Oracle hashes a SQL statement, it uses the whole text of the SQL statement, including spaces, capitals, and literals.

Logical Reads: The total number of reads from memory the execution of this query has required, over all the times it has been executed

Total Sorts: The total number of sort operations the execution of this query has required, over all the times it has been executed

Total Loads: The total number of times this query has been loaded or reloaded from memory

Finding the SQL Text

More information on the SQL statement, including the SQL text, can sometimes be viewed by clicking on the SQL_HASH value. However, the SQL text can always be extracted from the database. We can use a simple database query to duplicate the aforementioned SQL activity report, plus the text of the SQL statements.

The first step is to log in to SQL*Plus, or iSQL*Plus, using a username & password that can view the data dictionary (Oracle internal tables), as shown in Figure 6. If you've never used it, SQL*Plus is found under the "bin" directory of the Oracle Home on your machine.

Note: If you have access to a database tool like TOAD, of course, use that instead of SQL*Plus.

Once we have a SQL*Plus session open, we can query the data dictionary to replicate the OAM report, with the addition of the SQL text. We'll query the v$sqlarea view, a dynamic performance view that provides a window onto the SQL area, i.e., the area of memory where Oracle has cached the previously parsed SQL statements. The v$sqlarea view has

```
[oracle@mymachine] $ sqlplus

SQL*Plus: Release 10.2.0.1.0 - Production on Sun Aug 21 22:35:00 2005

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Enter user-name: system@apps01.world
Enter password:

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

SQL>
```

Figure 6 – Oracle SQL*Plus Login

all the information we need for this query.

Create a file with the following script, as shown in Figure 7, in it and save it as sqlarea1.sql :

```
set lines 120
set pages 1000
col sql_hash for 9999999999
col phyrd for 99999999
col logrd for 99999999
col sorts for 99999999
col execs for 99999999
col loads for 99999999
col sqltxt for a55
select * from (
select hash_value as sql_hash, disk_reads as phyrd, buffer_gets as logrd,
sorts, executions as execs, loads,
substr(sql_text,1,55) as sqltxt
from v$sqlarea
order by execs desc)
where rownum <= 15
/
```

Figure 7 – sqlarea1.sql

Now run the script in SQL*Plus. The results are shown in Figure 8.

Let's look at the script sqlarea1.sql in a bit more detail.

- The first nine lines are formatting statements to make the output easily readable in SQL*Plus; they can be omitted in a GUI SQL client such as TOAD.
- The results are sorted by the number of executions; change the "order by" clause in this query (see Figure 7) if you want to sort by another column in the report.
- Only the first 15 rows are displayed, for readability.
- Only the first few words of the SQL statement are displayed, for readability. You can replace the "substr(sql_text,1,55)" with simply "sql_text" if using a GUI client (see Figure 7).

Using This Report

What's the use of such a script? Why did Oracle include a similar report in OAM? What does it tell the Application DBA about this Oracle Applications instance?

There are several important pieces of information here. First, there are two important ways to look at the Executions column: top-down and bottom-up.

Sorted in descending order, as shown in Figure 8, this script provides an instant snapshot of which SQL statements are most frequently issued.

If the most frequently issued statements have a lot of physical reads, you're wasting time going to disk, and your Oracle Applications instance could benefit from increasing your database buffer cache.

If you're seeing high values in the LOADS column for the most

```
SQL> @sqlarea1.sql

  SQL_HASH  PHYRD    LOGRD SORTS   EXECS LOADS SQLTXT
_____ ___ ____ ___ ____ ___ _____________
 1053795750      0  2432257     0 3648977     2 COMMIT
 2525858779     12  4068516     0 1363654     5 SELECT COMP_FREQ FROM REF_JOB_
 3620777859      0  2929383     0  976461     4 SELECT (TO_CHAR(:B1, 'YYYY'))
  883532485      0  2322999     0  774327     4 SELECT (TO_CHAR(:B1, 'YYYY')-1
 2886384431   5124  4199746     0  624528     5 SELECT SALARY_NO FROM HR_DATA.
 1483828792    553  1640505     0  527922     5 SELECT MIN(EFF_DT) FROM HR_DAT
 4060129601  15750  2354645     0  323041     3 SELECT MAX(EFF_DATE), MIN(EFF_
 2830797694   1552  3422741     0  298323     3 SELECT EMPL_STATUS_CD FROM LAT
  492675944      3   818131     0  272710     1 SELECT GREATEST (:B4 , :B3 ),
  403132514      9   741159     0  247053     5 SELECT HOURS_PER_YEAR FROM REF
 1871132240      0  2235792     0  217636     5 UPDATE HR_WORK.POSITION SET US
 3749286049  28112  2582968     0  189875    12 SELECT NVL(A.ADT, B.ADT) FROM
 3773735564   4685 52115689  2320  189180     3 SELECT COUNT(*) FROM ( (SELECT
 3742653144     66   502748     0  167581     1 select sysdate from dual
 2734578310      0        0     0  131676     1 declare nlns number;  buf_t va

15 rows selected.

SQL>
```

Figure 8 – sqlarea1.sql Results

frequently used SQL statements, your performance may benefit from increasing the size of your database shared pool. The shared pool is where the SQL area is found. Check the GETHITRATIO column of the v$librarycache view – a well-tuned OLTP system should have a GETHITRATIO of .95 or higher for the SQL Area.

With the SQL text, this report can give you a quick feel for what's going on in the database.

Sorted in ascending order, expect to see a lot of statements with one execution. This is not normally a problem; however, recall from our discussion of hash values that two almost-identical SQL statements will have different hash values. This can be a problem if a lot of very similar SQL statements are passed – e.g., the same report is run frequently with different parameters.

(Tom Kyte, of Oracle Magazine's "Ask Tom" column, provides a very clear explanation of why this can be a major hit on database performance in his posting: http://asktom.oracle.com/pls/ask/f?p=4950:8:12818666420225533154::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:1516005546092).

Oracle provides a simple way to make two similar statements with different parameters have the same hash value. It's done by using bind variables. Briefly, instead of writing

```
select count(*)
 from emp
where ename='Smith' ;
```

the application developer would write:

```
select count(*)
 from emp
where ename=':b1' ;
```

```
set lines 120
set pages 1000
col sqltext for a30
col instances for 99999999
select * from (
select substr(sql_text,1,30) as sqltxt,
count(*) as instances
from v$sqlarea
group by substr(sql_text,1,30)
order by count(*) desc )
where rownum <= 15
/
```

Figure 9 – sqlarea2.sql

```
SQL> @sqlarea2.sql

SQLTXT                          INSTANCES
------------------------------ ----------
DECLARE job BINARY_INTEGER :=        2398
DECLARE var_val owa.vc_arr;          2347
select  /*+ cursor_sharing_exa       1583
declare    type ref_cur is ref       1057
SELECT /*+ PIV_SSF */ SYS_OP_M        981
SELECT /*+ TIV_SSF */ SYS_OP_M        975
SELECT * FROM HR_DATA.PERSON W        766
select /*+ cursor_sharing_exac        724
declare begin utility.applicat        125
INSERT INTO HR_WORK.PERSON ( I        117
select * from "ADHOC_USER"."TM         97
select * from "REPOMAN2"."SMP_         96
 select count(*)  from bulkloa         91
SELECT /*+NESTED_TABLE_GET_REF         60
select * from "BULKLOAD"."LATE         57

15 rows selected.

SQL>
```

Figure 10 – sqlarea2.sql Results

and then pass in a value for the bind variable, "b1", when the report is called. It's easy to check your system to make sure that your application SQL is using bind variables; simply count the number of similar SQL statements, using the script shown in Figure 9.

The output will resemble that shown in Figure 10.

As you can see from the listing, we've approximated "similar" statements with "statements whose first 30 characters match". If you see an Oracle Applications statement, or other application statement, with a high value of "INSTANCES", then it's worth drilling down on that statement to see whether or not there really are thousands of similar statements taking up memory and time.

ORAtips

The simplest way to do this is to look at the SQL statements whose first 30 characters match the listing above, and see if they are duplicates of each other that vary only by literals. To do this properly, we'll look at another dynamic performance view, v$sqltext.

v$sqltext contains only the hash value, the address at which the SQL is stored in memory, the command type, and the full text of each SQL statement, broken (arbitrarily) into lines. You can use this view (as shown in Figure 11) to look up the full text for any hash value in the SQL Activity report, or in our results shown in Figure 10.

In order to drill down on the suspect SQLs revealed by the script in Figure 11, we'll want to query on the SQL text instead of the hash value. In Figure 12, I've chosen to drill down on the first query in the sample output for Figure 11, i.e., 'DECLARE job BINARY_INTEGER :='

From the output of this script (Figure 13), we can clearly see that this SQL is being run repeatedly with literals instead of bind variables:

We have looked at the simplest possible way to check for almost-identical SQL statements that should use bind variables.

Tom Kyte provides a more sophisticated script that loops through the text of all the SQL in the shared pool, removes the literals, and then groups the statements to check for matches. The script can be found at http://asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:1163635055580

```
SQL> select piece, sql_text from v$sqltext
  2  where hash_value=1916299250
  3  order by address, piece ;

     PIECE SQL_TEXT
---------- ----------------------------------------------------------------
         0 SELECT SUN + MON + TUES + WEDS + THUR + FRI + SAT + SHIFT_SUN +
         1 SHIFT_MON + SHIFT_TUES + SHIFT_WEDS + SHIFT_THUR + SHIFT_FRI + S
         2 HIFT_SAT FROM SCHEDULES WHERE SCHEDULE_NO = :B1
SQL>
```

Figure 11 – v$sqltext.sql results

```
set lines 70
set pages 1000
select sql_text from v$sqltext s1
where (s1.hash_value, s1.address) in
 (select hash_value, address from v$sqltext s2
 where substr(sql_text,1,30) = 'DECLARE job BINARY_INTEGER := ')
order by hash_value, address, piece
/
```

Figure 12 – sqlarea3.sql

```
SQL> @sqlarea3.sql

SQL_TEXT
----------------------------------------------------------------
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('1'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('2'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('3'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('4'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('5'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('6'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
[....]
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN exec myproc('2398'); commit; end;
:mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;
2398 rows selected.
SQL>
```

Figure 13 – sqlarea3.sql Results

ORAtips

on Database - Monitoring

Sample output from this script is shown as Figure 14.

Note: I've truncated the values of SQL_TEXT_WO_CONSTANTS in this example for readability. The full SQL text will display when you run the script.

Performance Hint: CURSOR_SHARING

So what's a DBA to do if the above scripts reveal a lot of shareable SQL that's not being shared? Fortunately, there's a database initialization parameter that tells Oracle to substitute bind variables for literals whenever it is passed a SQL statement with literals. This initialization parameter, CURSOR_SHARING, is available in database versions 8.1.6 and above. In 8i releases 8.1.6 and above, you can set CURSOR_SHARING=FORCE at session or system level, and Oracle will replace all literals with bind variables.

This can have performance disadvantages as well as advantages. Inappropriate variable substitutions can cause the CBO to choose sub-optimal query plans. So CURSOR_SHARING=FORCE should only be implemented if you are seeing real problems with bind variable underuse in your system.

If you see a SQL statement that is long running, with thousands of executions, it's a good bet that a session is currently running it, or has run it recently.

```
SQL> set lines 70
SQL> set pages 1000
SQL> col sql_text_wo_constants for a55
SQL> col cnt for 999999
SQL> set lines 65
SQL> @remove_constants

Table created.


Table altered.


Function created.


8473 rows updated.


SQL_TEXT_WO_CONSTANTS                                    CNT
____________________ __
DECLARE VAR_VAL OWA.VC_ARR; VAR_NAME OWA.VC_ARR; DUMMY_  829
DECLARE VAR_VAL OWA.VC_ARR; VAR_NAME OWA.VC_ARR; DUMMY_  792
DECLARE VAR_VAL OWA.VC_ARR; VAR_NAME OWA.VC_ARR; DUMMY_  285
DECLARE VAR_VAL OWA.VC_ARR; VAR_NAME OWA.VC_ARR; DUMMY_  146
DECLARE VAR_VAL OWA.VC_ARR; VAR_NAME OWA.VC_ARR; DUMMY_  133

11 rows selected.

SQL>
```

Figure 14 – Tom Kyte's "remove_constants" Results

Database versions 9iR2 and above provide a better option which can be implemented with no performance tradeoff: you can set CURSOR_SHARING=SIMILAR with the result that bind variables are only substituted for literals when it won't have a negative effect on the query plan.

Who's Been Running This SQL?

Let's return to the SQL Activity Report and take a look at another drilldown. If you see a SQL statement that is long running, with thousands of executions, it's a good bet that a session is currently running it, or has run it recently. This drilldown will tell you which Oracle account has been running the SQL in question.

Start with a hash value from the SQL Activity report, or from our SQL-text-enhanced version, sqlarea1.sql. This hash value is present in a dynamic performance view, v$session, which contains information on all current database sessions. So it's simple to see if anyone is currently running that statement.

```
set lines 120
set pages 1000
col sql_hash for 9999999999
col phyrd for 99999
col logrd for 9999999
col sorts for 999
col execs for 999999
col cputime for 999999999
col cpu_per_exec for 999999
col sqltxt for a30
select * from (
select hash_value as sql_hash, disk_reads as phyrd, buffer_gets as logrd,
sorts, executions as execs, cpu_time as cputime,
decode(executions,0,0,cpu_time/executions) as cpu_per_exec,
substr(sql_text,1,30) as sqltxt
from v$sqlarea
order by cpu_time desc)
where rownum <= 15
/
```

Figure 15 – sqlarea1.sql Modified

We'll start by altering sqlarea1.sql (Figure 15) to focus on long-running queries rather than the most frequently executed queries.

Executing, we see a different set of queries (Figure 16) – the queries that have been the most CPU-intensive on our system:

(Again, I have truncated the SQL text in Figure 16 for readability.) Now, let's see if any of these resource hogs are running. Let's look at the SQL statement with hash value 1861958696 (Figure 17):

Running this script shows me (Figure 18) which Oracle user(s) is (are) executing this SQL currently, and how many sessions of each user are executing it:

```
SQL> @sqlarea1.sql

 SQL_HASH  PHYRD    LOGRD SORTS   EXECS    CPUTIME CPU_PER_EXEC SQLTXT
---------- ------ -------- ----- ------- ---------- ------------ ------
 826635222 ###### ########     0     569 ##########      ####### declare begin
1090992093 ###### ########     0     173 ##########      ####### declare begin
1758711393     65 ########   912     912 ##########      ####### DELETE FROM
4162728552 ###### ########     0    1548 ##########      ####### declare begin
1861958696  12537 ########  ####   46846 ##########        50795 select null
1483738633 ###### ########     0     290 ##########      ####### declare segs
 538521295  42037    11879    26      13 ##########      ####### SELECT vdq.
1966261648 ###### ########  ####    2068 ##########       519917 select ccid,
1997068893   2052 ########  ####   56866 ##########        18387 select null
1436358176 ###### ########     0       1 ##########      ####### DECLARE job
3282269111  31440 ########     0     152  920718750      ####### DELETE FROM
2886384431  30196 ########  #### #######  877937500          454 SELECT SALAR
 819838265 ###### ########     0     164  841234375      ####### declare begin
3591315707 ######  2717431   913     913  822718750       901116 INSERT INTO
4068684737 ###### ########     0     912  685984375       752176 DELETE FROM

15 rows selected.

SQL>
```

Figure 16 – sqlarea1.sql (modified) Results

ORAtips on Database - Monitoring

```
col username for a30
select username, count(*) cnt_executing_sessions
from v$session sess
where sess.sql_hash_value=1861958696
or sess.prev_hash_value=1861958696
group by username
/
```

Figure 17 – sqlsession.sql

```
SQL> @sqlsession

USERNAME                    CNT_EXECUTING_SESSIONS
__________ ________
WEBSERVER                                  2

SQL>
```

Figure 18 – sqlsession.sql Results

Extending OAM With Your New Scripts

OAM can be extended to include the scripts we've just gone over. Add these custom scripts by using the SQL Extensions page. (Navigate to Site Map > Others > SQL Extensions). The procedure for adding custom scripts as SQL extensions is well documented; see the Oracle Applications System Administrator's Guide – Maintenance (B13924-02), pp. 4-22 ff.

Conclusion

We've seen how to extend OAM's SQL monitoring capability greatly by running a few simple scripts against the database itself. The SQL activity report has gone from a cryptic list of hash values to a springboard into in-depth information on SQL execution in the database hosting your Oracle Application instance. And OAM can be extended to include these custom scripts as SQL Extensions, making this additional information easily accessible and convenient.

Future articles will drill down on other aspects of OAM's database monitoring, such as its list of database sessions.

Natalka Roshak – Natalka is a Senior Oracle Database Administrator and an Oracle Certified Professional in Database Administration. She provides expert database consulting solutions across North America from her base in Southern Ontario. Natalka may be contacted at Natalka.Roshak@ERPtips.com

ORAtips Journal