



Select Magazine - November 2000

Volume 7, No. 2

Taming Ad-hoc Queries in a Very Large Data Warehouse

By Sundaresh Valiveti

In a large and geographically dispersed user population, understanding what users run against our databases is difficult, time consuming and an inexact science. To battle these problems, a series of SQL code was developed to identify potential end user-query issues.

Our SQL tool provides a simple way to view what programs our users run and the amount of resources they use.

We all know the beauty and value of our data warehouse applications. They provide an almost unlimited ability for an end user in an ad-hoc query environment to gather data and reports for a competitive advantage. However, with every good comes some bad. In a large data warehouse application, end users can severely impact performance of our database through these ad-hoc tools. When database performance is compromised, the DBA team is put on the spot.

Although a myriad of items can affect database and system performance, some items we want to understand better are end-user queries. In a large and geographically dispersed user population, understanding what users run against our databases is difficult, time consuming and an inexact science. To battle these problems, a series of SQL code was developed to identify potential end user-query issues.

The goals of this project were to do the following:

- Provide SQL code that could be maintained by anyone with SQL and DBA experience (no need for other language skills).
- Generate a formatted and complete SQL code being run by an end user.
- Generate SQL code that could be run to get an Oracle Explain Plan.
- Determine when and for how long to monitor the database.
- Provide as little or as much detail as needed.

Our SQL tool provides a simple way to view what programs our users run and the amount of resources they use. We do this by employing Oracle's dynamic performance tables. Specifically, we draw from information in the shared pool using V\$SESSION. After we have obtained

the SQL addresses of queries that meet our criteria, we join to V\$SQLAREA and DBA_USERS table to obtain detailed user and resource information. This information can be viewed on-line or spooled to a file for later analysis.

In order to make this tool as generic as possible, we utilized V\$SQLTEXT instead of V\$SQLTEXT_WITH_NEWLINES. This was done to capture, parse, join and re-format any SQL generated by third party ad-hoc tools. Many of these tools send SQL to the database that have no new line characters within the text. By using V\$SQLTEXT, the tool becomes generic enough to format almost any SQL sent to the database. This gives us an easy-to-read and executable SQL statement for analysis.

Other material has been written regarding the use of Oracle's V\$SESSION and V\$SQLAREA to identify problem queries. All of the examples I have seen set up a parameter to measure against items such as disk reads or buffer gets. Although these criteria can be useful, unless you have a strong historical baseline, it is difficult to select a threshold to measure against. Are ten disk reads too much or are 10,000?

The one thing all DBAs know is that if a query is taking a long time to complete, their phone is going to quickly ring. Time is something that is measurable and understandable and can be the baseline to gather other statistics for future thresholds (such as disk reads).

Using the Tool in Real Life

This tool was first developed in order to identify which users were submitting ad-hoc requests against our databases that were impacting performance.

In order to accomplish this goal, a set of SQL code was wrapped around UNIX shell programs. The process was set up to query the database at set intervals and gather information on long running user queries. What was considered "long running" was just a parameter that we passed on the command line to the shell. For example:

```
Unix> monitor_db.ksh          &dbms &oracleid &oracle_password &minutes &output          &runfor
Dbms = database              instance
Oracle_id = oracle           id
Oracle_password              = password
Minutes = get                 queries running > minutes
Output = directory/          file name
Runfor = how                  long to monitor database
```

The result of running this shell was the ability to monitor and report on any query that was running longer than the minute threshold we passed to the shell.

An example of the output:

```
Rem
Rem SID 25 DB Instance TESTDB Date 21-feb-00 12:04:41 How long ?
0Hrs:11Mi:48Ss
Rem Username RGANTA OSuser RGANTA Machine TESTMACHINE Terminal Windows 95
Rem Query Start Time Assumed 21-feb-00 11:52:53 Logon Time 21-feb-00 11:52:52
Rem SqlAddr 8970B160 Status ACTIVE
Rem Program OraPgm
Rem
Rem About The SQL DB Info
Rem Executions 1 Sorts 2 Disk Reads 0 Buffer Gets 20 Rows 0
Rem First Load time 21-feb-00 11:52:53 Parsing User/Id RGANTA/0905
Rem Optimizer CHOOSE
set echo on
set timing on
set autotrace traceonly explain
set pause off
set linesize 132
set trimspool on
SELECT EMP_NAME, EMP_TITLE, EMP_SSN, EMP_SALARY, DEPT_DEPTNUM, DEPT_DEPTDESC
From EMP_TABLE, DEPT_TABLE
```

```

WHERE

EMP_DEPTNO=DEPT_DEPTNUM

ORDER BY EMP_NAME

spool off

quit

```

This output allows the DBA team to analyze the SQL a user was executing that was taking more than XX minutes to complete. One nice thing about the tool is that when it is run every day, we have a history of activity to look back on. So when you go home early Friday afternoon (ha ha), and on Monday a user complains that their query took forever to finish Friday night, you have someplace to start looking. With this tool, further analysis of real database usage can be used to fine tune the database or even assist end users to obtain data through more efficient SQL queries.

How it Works

The SQL code is broken into different parts that can easily be modified or extended to meet various needs.

Once the monitor_db.ksh is started with all of your parameters, it executes monitor_db.sql. This SQL monitors the database activity, continually looking for queries running over the threshold we set up. This SQL obtains the SQL addresses of any active SQL running. It does this by joining to V\$SESSION and V\$SQLAREA on address. The result of this query is written to an intermediate file that is used to obtain the detailed information on the queries. The output from this SQL is really a SQL command file to be executed later.

```

Example output of monitor_db.sql

REM Sample output of monitor_db.sql

REM scott/tiger = userid/password

REM @testdb = database instance

REM @get_sql = execute get_sql with sql address

REM and database instance

REM =====

sqlplus

HYPERLINK mailto:scott/tiger@testdb

```

```

scott/tiger@testdb

@get_sql 8A2D0AD8 testdb .

sqlplus

HYPERLINK mailto:scott/tiger@testdb

scott/tiger@testdb

@get_sql 8B5EAC8D testdb .

```

This intermediate file is then executed, which runs our `get_sql.sql` script. This script takes as an argument the SQL address that we obtained from `monitor_db.sql`. The `get_sql.sql` script is really at the heart of our mini application. Using the SQL address and Oracle's `V$SESSION` and `V$SQLAREA` views, along with `DBA_USERS` table, we can obtain a great deal of useful information.

The first set of SQL in `get_sql.sql` obtains the user information for the given SQL address. Joining the SQL address from Oracle's `V$SESSION` and `V$SQLAREA` allows us to obtain the user's name, machine name, database instance, etc. The SQL code can be easily modified to include any other information that may be of value to the DBA staff that is available in `V$SESSION` or `V$SQLAREA`.

A closer look at this SQL will reveal that we perform a "union all" to determine whether the SQL address is still active. This is done in the unlikely case that a query that we identified earlier has already completed before we were able to obtain the detailed information.

The second part of the SQL obtains more information using the `DBA_USERS` table. By joining on parsing `user_id` from `V$SQLAREA` and `user_id` from `DBA_USERS`, we can arrive at a user name that we can recognize. This part of our code also obtains detailed resource information for this SQL. In our case, we have elected to retrieve information such as disk reads, buffer gets and rows processed. Again, the code could be modified for other items of interest to you.

The third `SELECT` simply creates the commands that are useful for creating an Oracle Explain Plan. You could easily "cut and paste" this SQL to obtain an explain plan for the SQL statement you are analyzing.

The fourth set of SQL within `get_sql` nicely formats the actual SQL from `V$SQLAREA` based on the SQL address. The `SQL_TEXT` column in `V$SQLTEXT` is a `varchar2(64)`. The way that this data is stored does not allow us to get a clean piece of SQL code that we can execute. Each SQL text is stored up to 64 bytes, so we use a series of in-line views and `SQL*Plus` commands to parse, join and format the SQL. Since our goal is to provide an executable SQL script, we parse and put together the various `SQL_TEXT`. By looking for keywords such as "FROM", "WHERE", etc., we are able to break an SQL statement from Oracle into two parts for readability. Each part is then joined with the next piece of `SQL_TEXT` through the use of the `PIECE_ID` field in `V$SQLTEXT`.

The fifth SQL just does some cleanup like taking the spool off and exiting.

Benefits

A DBA can easily modify the shell to meet almost any criteria he or she may wish to monitor. The example illustrated within this article is looking for long running queries, but it could just as easily look for a large number of disk reads, sorts or whatever you wish to monitor that is available in Oracle's V\$SESSION and V\$SQLAREA.

We have been using this tool for some time to capture long running queries in our data warehouse application. Using the output from the tool has allowed us to analyze end-user query trends. With this information, we are able to suggest alternative tables or views to a user that would improve performance. As a DBA in a data warehouse, we are better able to recommend summary tables that would benefit end users. By having at our disposal an on-going database monitoring process, we are able to know and understand exactly what SQL a user is running and also its impact on the system.

These tools showed us that the same set of users executed many similar, complex queries. This allowed us to consolidate on the reporting side and drastically reduce the amount of SQL requests to the database. The direct effect of this is that users obtained their reports more quickly (with less database contention), thereby freeing up our hardware resources. From a data architecture perspective, we determined SQL is being run. Information like this can be used by a data architect to analyze table and column usage in order to make recommendations about how best to utilize the database.

In any data warehouse environment, tuning is critical to the success of the project. The tool mentioned here can serve any DBA well. It allows you to see and capture what is being run against the database based upon what you want to monitor. Using this tool offers you a wealth of information about which users are running what queries, how those queries are impacting performance, reporting trends and a lot more. From my experience, although Oracle Enterprise Manager is a good tool, I have yet to find something in it that puts all of the pieces together for me in as simple a manner as this tool. The added benefit is when I am on the beach over the weekend and a user has one of those long queries, I'll know exactly what they were doing when I get in on Monday.

Caveats

Since this SQL runs against internal Oracle tables, the user should have DBA or "select any table" privileges.

Shell and SQL programs were developed and tested on UNIX System V and Oracle 7.3.4. These scripts and SQL should run on other flavors of UNIX and Oracle 8.

Also it should be noted that depending upon how many users or what thresholds you set, your output files could get large.

About the Author

Sundaresh Valiveti is a consultant with DMR Consulting Group, Inc., and has 10 years experience as an Oracle DBA. For the past two years he has been supporting AT&T's Financial Community Information Warehouse. Sundaresh has an MS in Mathematics from Osmania University in Hyderabad, India. He extends thanks to his fellow DBA staff at AT&T for their support. You can reach him at svaliveti@dmr.com

monitor_db.ksh

```
#!/bin/ksh

# =====

# monitor_db.ksh is outer shell to monitor the database

# Use '?' to get details on how to run the shell script

#=====

#=====

# Start of Script

#=====

#=====

# check # of arguements

#=====

if (( $# == 0 )) || (( $# < 3 )) || [[ $1 = '?' ]]

then

echo "Database Monitoring Script to Locate Long Running Queries ..."

echo "Following are the Parameters/Arguments to the Shell script ..."

echo "1. Database / Connect String (REQUIRED) ....."

echo "2. Oracle User Id (REQUIRED) ....."

echo "3. Oracle Password (REQUIRED) ....."

echo "4. Search for Queries running Longer then XX minutes....."

echo " Default is 30 Minutes (Optional, If specified should be 4th Argument)"

echo "5. Directory name where Ouput is created ....."
```

```
echo " Default is current directory (Optional, If specified should be 5th Argument)"

echo "6. How long to run this monitoring script. Enter time in Minutes ....."

echo " Default is 60 Minutes (Optional, If specified should be 6th Argument) "

exit 1

fi

#=====

# Query time criteria to check... default is 30 minutes

#=====

echo "Default monitor interval is 10 Minutes ....."

Sleep_Time=5

Query_Time=$4

if [[ -z $Query_Time ]]

then

Query_Time=30

echo "Default setting for finding Long Running queries is $Query_Time Minutes ..."

fi

#=====

# Check directory for output to be written to .... default is current dir

#=====

Dir_Name=$5

if [[ -z $Dir_Name ]]
```

```
then

Dir_Name=`pwd`

echo "Default setting for Directory is $Dir_Name ..."

fi

#=====

# Check how long this shell should monitor database.... default is 60 Mts

#=====

How_Long_To_Run=$6

if [[ -z $How_Long_To_Run ]]

then

How_Long_To_Run=60

echo "Default setting for How long Monitoring should be made is $How_Long_To_Run
Minutes ..."

fi

#=====

# Determine starting time of this script

#=====

START_HH=`date +%H`; export START_HH

START_MM=`date +%M`; export START_MM

(( START_IN_MINUTES = ( $START_HH ) * 60 + $START_MM ))

#=====

# Loop through searching for Queries that meet criteria
```

```
# until $How_Long_To_Run is reached (parm passed to shell or default value

#=====

while :

do

echo "Database Instance " $1 `date`

t=`sqlplus -s $2/$3@$1 @monitor_db $1 $2 $3 $Query_Time $Dir_Name `

if [[ -z $t ]]

then

echo "No Queries running Longer than $Query_Time Minutes"

fi

chmod +x $1_sqls.ksh

$1_sqls.ksh

#=====

# Check to see if script has run for allotted time

#=====

END_HH=`date +%H`; export END_HH

END_MM=`date +%M`; export END_MM

((END_IN_MINUTES = ( $END_HH ) * 60 + $END_MM ))

((RUN_TIME = $END_IN_MINUTES - $START_IN_MINUTES))

if [ $RUN_TIME -ge $How_Long_To_Run ]

then

echo "Monitoring Completd."
```

```
exit

fi

# Sleep time

sleep $Sleep_Time

done

#=====

# End of script

#=====
```

monitor_db.sql

```
Rem =====
Rem monitor_db.sql script runs every 10 minutes to obtain
Rem the sql address for any active sql that meets our
Rem selection criteria (Query_Time).
Rem =====
Rem Input Params:
Rem Database / Connect String = &1 .....
Rem Oracle User Id = &2 .....
Rem Oracle Password = &3 .....
Rem Search for Queries running Longer then XX minutes = &4 .....
```

```
Rem Directory name where Ouput is created = &5 .....

Rem =====

set trimspool on

set heading on

set pages 0

set feedback off

set pause off

set verify off

Rem =====

Rem Create ksh output file with sqlplus commands

Rem =====

spool &1._sqls.ksh

select 'sqlplus '|| '&2/&3&&1' ||' @get_sql '||b.sql_address||' &1.
&5'

from v$session b, v$sqlarea c

where b.type='USER' and

b.status = 'ACTIVE' and

b.username not in ('SYS', 'SYSTEM') and

b.sql_address=c.address and

(sysdate -

greatest(logon_time,to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss')))

* 86400 / 60 >= &4 ;
```

```
quit

Rem =====

Rem End of Script

Rem =====
```

get_sql.sql

```
Rem =====

Rem get_sql.sql      script is to get the SQL from Oracle Database Shared Pool
Rem This SQL        gets the complete SQL statements for a given SQL address
Rem from Oracle     Shared Pool. The output of the SQL is nicely formatted
Rem with additional user and resource information to Know more about query
Rem Running for     Long Time

Rem =====

Rem Input Params:

Rem SQL Address      = &1 ....."
Rem Database         / Connect String = &2 ....."
Rem Output Directory Name = &3 ....."

Rem

Rem This Sql        Gets SQL for a Given SQL Address From Oracle Databse SharedPool
Rem This has        5 SELECTS....
Rem SELECT 1        Gets User Identification Details, How long Query is running etc
```

```
Rem SELECT 2          Gets SQL DB Info like Executions,DiskReads,BufferGets info
Rem SELECT 3          Gets Few "set " commands helpful to get Explain Plan
Rem SELECT 4          Gets ACTUAL SQL(DbInstance_SqlAddres.sql) from Shared Pool
Rem SELECT 5          Gets "spool off and quit" to be spooled to the SQL generated
Rem Last Step        is to Move SQL to a Directory Specified(Long running sql Dir)
Rem
set linesize         132
set verify off
set trimspool        on
set feedback         off
set pause off
set pages 0
column piece         noprint
column howlong       heading "Howlong ?" format a15
spool &2._&1..sql
column sqltext       format a132
Rem
Rem SELECT 1          Gets User Identification Details, How long Query is running etc
Rem
select 'Rem'         ||chr(10)||
'Rem SID ' ||       sid || ' DB Instance &2' ||
```

```

' Date ' || to_char(sysdate,'dd-mon-yy          hh24:mi:ss') ||

' How long ?          ' ||

trunc((sysdate          - greatest(logon_time,

to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss')))          * 86400/3600) || 'Hrs:' ||

trunc(mod((sysdate          - greatest(logon_time,

to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss')))          *

86400,3600)/60) || 'Mi:' ||

trunc(mod(mod((sysdate          - greatest(logon_time,

to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss')))          * 86400,3600),60)) || 'Ss' ||

chr(10) ||

'Rem Username          ' || username || ' OSuser ' || osuser ||

' Machine ' ||          machine || ' Terminal ' || terminal || chr(10) ||

'Rem ' || 'Query          Start Time Assumed ' ||

to_char(greatest(logon_time,

to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss')),

'dd-mon-yy hh24:mi:ss') ||

' Logon Time          ' || to_char(logon_time,'dd-mon-yy hh24:mi:ss') || chr(10) ||

'Rem' || ' SqlAddr          ' || '&1' || ' Status ' || status || chr(10) ||

'Rem Program          ' || program || chr(10) || 'Rem'

from v$session          a, v$sqlarea b

where a.sql_address='&1'

and b.address='&1'

```

```

union all

select 'Rem ' ||chr(10)||

'Rem User/Session          Information is Unavailable at ' ||

To_char(sysdate,'dd-Mon-yyyy          hh24:mi:ss') ||chr(10)||

'Rem May be                the Session Completed.' ||chr(10)||

'Rem '

from dual

where not exists          (select 1

from v$session          a

where a.sql_address='&1');

Rem

Rem SELECT 2              Gets SQL DB Info like Executions,DiskReads,BufferGets info

Rem

select 'Rem About          The SQL DB Info ' ||chr(10)||

'Rem Executions          ' || executions ||' Sorts ' || sorts||

'Rem Disk Reads          ' || disk_reads||

'Rem Buffer Gets          ' || buffer_gets ||

'Rem Rows ' ||          rows_processed||chr(10)||

'Rem First Load          time ' ||

to_char(to_date(first_load_time,'yyyy-mm-dd/hh24:mi:ss'),

'dd-mon-yy hh24:mi:ss') ||

'Rem Parsing User/Id          ' ||nvl(b.username,'NOUSER') || '/' || parsing_user_id ||

```

```

chr(10)|| 'Rem          Optimizer '||optimizer_mode
from v$sqlarea          a, dba_users b
where address='&1'
and b.user_id(+)=a.parsing_user_id;

Rem

Rem SELECT 3          Gets Few "set " commands helpful to get Explain Plan

Rem

select 'set echo          on' ||chr(10)||
'set timing          on' ||chr(10)||
'set autotrace          traceonly explain' ||chr(10)||
'set pause off' ||chr(10)||
'set linesize          132' ||chr(10)||
'set trimspool          on' ||chr(10)||
'spool &2._&1..lst'

from dual;

Rem

Rem SELECT 4          Gets ACTUAL SQL From Database Shared Pool

Rem

select d.piece          piece,
decode(substr(upper(ltrim(sqltext,'          ')),1,4),
'SELE', ltrim(sqltext,'          '),

```

```

'FROM', ltrim(sqltext, ' '),
'WHER', ltrim(sqltext, ' '),
'ORDE', ltrim(sqltext, ' '),
'GROU', ltrim(sqltext, ' '),
'UNIO', ltrim(sqltext, ' '),
'AND ', ltrim(sqltext, ' '),
'EXPL', ltrim(sqltext, ' '),
'(SEL', ltrim(sqltext, ' '),
'UPDA', ltrim(sqltext, ' '),
'DELE', ltrim(sqltext, ' '),

' '||sqltext)          sqltext
from (select          a.piece, decode(b.sql1,' ','',b.sql1)||a.sql sqltext
from ( select          piece,
decode(piece,          last_piece, sql_text||';',
decode(instr(upper(sql_text),'          FROM ',-1),0,
decode(instr(upper(sql_text),'          WHERE ',-1),0,
decode(instr(upper(sql_text),'          UNION ALL', -1),0,
decode(instr(upper(sql_text),'          UNION ', -1),0,
decode(instr(upper(sql_text),'          ORDER ',-1),0,
decode(instr(upper(sql_text),'          GROUP ',-1),0,
decode(instr(sql_text,' ','',-1),0,
decode(instr(sql_text,'          ',-1),0,

```

```

decode(instr(sql_text,')',-1),0,          sql_text,
substr(sql_text,1,instr(upper(sql_text),')',          -1))),
substr(sql_text,1,instr(upper(sql_text),'          ',-1))),
substr(sql_text,1,instr(upper(sql_text),',',-1))),
substr(sql_text,1,instr(upper(sql_text),'          GROUP ',-1))),
substr(sql_text,1,instr(upper(sql_text),'          ORDER ',-1))),
substr(sql_text,1,instr(upper(sql_text),'          UNION ALL',-1))),
substr(sql_text,1,instr(upper(sql_text),'          UNION ',-1))),
substr(sql_text,1,instr(upper(sql_text),'          WHERE ',-1))),
substr(sql_text,1,instr(upper(sql_text),'          FROM ',-1))) sql

from v$sqltext,          (select max(piece) last_piece
from v$sqltext
where address='&&1')          c
where address='&&1'
and piece=c.last_piece(+)          a,
( select piece          + 1 tail_piece,
decode(instr(upper(sql_text),'          FROM ',-1),0,
decode(instr(upper(sql_text),'          WHERE ',-1),0,
decode(instr(upper(sql_text),'          UNION ALL', -1),0,
decode(instr(upper(sql_text),'          UNION ', -1),0,
decode(instr(upper(sql_text),'          ORDER ',-1),0,

```

```

decode(instr(upper(sql_text),'          GROUP ',-1),0,
decode(instr(upper(sql_text),' ',-1),0,
decode(instr(upper(sql_text),'          ',-1),0,
decode(instr(upper(sql_text),' )',-1),0,          sql_text,
substr(sql_text,instr(upper(sql_text),' )',          -1)+1)),
substr(sql_text,instr(upper(sql_text),'          ', -1)+1)),
substr(sql_text,instr(upper(sql_text),' ',          -1)+1)),
substr(sql_text,instr(upper(sql_text),'          GROUP ',-1)+1)),
substr(sql_text,instr(upper(sql_text),'          ORDER ',-1)+1)),
substr(sql_text,instr(upper(sql_text),'          UNION ',-1)+1)),
substr(sql_text,instr(upper(sql_text),'          UNION ALL',-1)+1)),
substr(sql_text,instr(upper(sql_text),'          WHERE ',-1)+1)),
substr(sql_text,instr(upper(sql_text),'          FROM ',-1)+1)) as sql1

from v$sqltext          ,(select max(piece) last_piece
from v$sqltext
where address='&&1')          c
where address='&&1'
and piece!=c.last_piece)          b
where a.piece=b.tail_piece(+)          ) d

order by piece

/

Rem

```

```
Rem SELECT 5           Gets "spool off and quit" to be spooled to the SQL generated
Rem
select 'spool         off '||chr(10)||
'quit '
from dual;
spool off
Rem
Rem Last Step         is to Move SQL to a Directory Sepecified(Long running sql Dir)
Rem
!mv -f &2._&1..sql    &3/
quit
Rem =====
Rem End Of Script
Rem =====
```

MBED Visio.Drawing.4

[Download Acrobat Reader](#)

Copyright 2003 by the International Oracle Users Group