

All about SQL Tracing

One of the most important responsibilities of an Oracle Database Administrator/Performance tuning Analyst, when it comes to performance diagnostics is to determine in detail how applications/users access the database. This article is my attempt to show the different ways one can activate tracing of an Oracle session for performance diagnostics. While the Oracle documentation mentions some of the methods, it doesn't cover all and you'll see this article consolidates many methods of tracing an Oracle session some of which are well documented and some others that are undocumented and reserved.

The primary audience targeted by this article is the Oracle DBA and Performance tuning analyst (Developer or DBA) and assumes prior SQL familiarity. However the article does not get into the trace file analysis itself or troubleshooting guidelines (that'll be a separate article/book by itself). By 'tracing' we mean sql tracing for all purposes, unless otherwise mentioned.

Session tracing can be useful in any of the following circumstances:

- One or more sessions seem to be using a disproportionate amount of some resource (from V\$SESSTAT, V\$SESSION_EVENT or V\$SQL)
- To determine how time is divided for typical users of an online system.
- To isolate more information about particular waits that are occurring.

Now, lets get down to the subject, shall we?

Prerequisites and Setup

Prior to activating tracing of a session, the following parameters need to be set either at the system or session level

TIMED_STATISTICS = true

MAX_DUMP_FILE_SIZE = unlimited (or a good enough high value, measured in OS blocks)

USER_DUMP_DEST =<location of the Server process trace file generated>

Timed_Statistics is a dynamic session or system level parameter that controls whether timing related information is collected. Without activating timed_statistics, the trace files are far from useful as the cpu & elapsed time default to zero. My recommendation is to set timed_statistics to true **always** as this puts negligible overhead on the overall performance of the system.

To turn this parameter on instance wide, issue the following in SqlPlus:

```
Alter system set timed_statistics = true;
```

If you cannot even turn on timed_statistics for the fear of affecting performance negatively, then the system needs performance tuning anyway, for which you will need to turn on timed_statistics anyway! . Besides performance tuning is an ongoing process and it may be better to just leave this turned on.

Note that this parameter defaults to TRUE in Oracle 9i since STATISTICS_LEVEL(another parameter that we'll cover shortly) defaults to TYPICAL.

Max_Dump_File_Size specifies the maximum size of the generated trace file, in OS blocks and is 'unlimited' by default.

Output files contain raw trace output and can be found in the USER_DUMP_DEST directory for user sessions and BACKGROUND_DUMP_DEST for background processes, on the Database Server. The raw trace file can be analyzed as such (beyond the scope of this article) or formatted into a more readable format using Tkprof (but note that Tkprof doesn't show all the details the raw trace file contains).

Before we get into the different ways of tracing Oracle sessions, there's just couple of other parameters, I want to mention

TRACEFILE_IDENTIFIER specifies a custom identifier that becomes part of the Oracle Trace file name. It doesn't have a default value. On Unix platforms, the name of the trace file generated for Server processes is of the following format

```
[ORACLE_SID]_ora_[Server_Process_Id]_[traceid].trc
```

where traceid is the value of TRACEFILE_IDENTIFIER ,if defined on the session ,otherwise null. So you may want to define a custom tracefile_identifier , based on say username, hostname, osuser etc to easily identify the session tracefile in user_dump_dest directory. You could query TRACEID from V\$PROCESS to get the TRACEFILE_IDENTIFIER of a given session.

STATISTICS_LEVEL: This is a parameter that controls the level of statistics collected in the database and available in Oracle 9i. Depending on the setting of STATISTICS_LEVEL, certain advisories and statistics are collected, as follows:

BASIC: No advisories or statistics are collected.

TYPICAL: The following advisories or statistics are collected:

- Buffer cache advisory
- MTTR advisory
- Shared Pool sizing advisory
- Segment level statistics
- PGA target advisory
- **Timed statistics**

ALL: All of the preceding advisories or statistics are collected, plus the following:

- Timed operating system statistics
- Row source execution statistics

The default level is TYPICAL. STATISTICS_LEVEL is a dynamic parameter and can be altered at the system or the session level. Timed statistics are automatically collected for the database if the initialization parameter STATISTICS_LEVEL is set to TYPICAL or ALL. If

STATISTICS_LEVEL is set to BASIC, then you must set TIMED_STATISTICS to TRUE to enable collection of timed statistics.

You could query v\$statistics_level to determine what statistics are being enabled/collected.

```
select * from v$statistics_level where STATISTICS_NAME like 'Timed%';
```

Various ways of tracing a session:

Instance wide tracing:

For Instance wide tracing (which is rarely required and incurs heavy performance penalty) set

```
ALTER SYSTEM SET SQL_TRACE=TRUE scope=spfile;
```

and bounce the instance. To disable instance wide sql tracing

```
ALTER SYSTEM SET SQL_TRACE=false scope=spfile;
```

In 8i and prior, set

```
SQL_TRACE=TRUE|FALSE
```

in the init.ora parameter file to control instance wide sql tracing.

Tracing your session:

The simplest way to activate sql tracing for your own session in sqlplus is

```
Alter session set sql_trace=TRUE;
```

and to disable tracing

```
Alter session set sql_trace=FALSE;
```

To control tracing within PL/SQL, we could make use of **DBMS_SESSION** package.

To enable tracing within Pl/sql

```
dbms_session.set_sql_trace(TRUE);
```

and to disable tracing

```
dbms_session.set_sql_trace(FALSE);
```

Oracle provides an almost dedicated package for sql tracing viz DBMS_SUPPORT

Let's see how **DBMS_SUPPORT** package can be used to enable/disable tracing your session.

To enable tracing

```
dbms_support.start_trace;
```

and to disable

```
dbms_support.stop_trace;
```

Note that Dbms_support package may not be part of your default installation and hence you may need to request Oracle support for the package creation scripts. It needs to be created by running \$ORACLE_HOME/rdbms/admin/dbmssupp.sql. This package cannot be used against 7.1 versions or earlier.

The default is to trace everything SQL_TRACE would capture plus WAIT information.

```
DBMS_SUPPORT.START_TRACE (waits=>false, binds=>false);
```

is equivalent to standard SQL_TRACE

```
DBMS_SUPPORT.START_TRACE (waits=>true, binds=>false);
```

is the same as the default and gives information on WAITS as well as standard SQL_TRACE

```
DBMS_SUPPORT.START_TRACE (waits=>true, binds=>true);
```

shows both bind variables values and wait information.

Note: WAITS are the waits experienced on events that happen within database calls like

'db file scattered read' or 'log file parallel write' or events that occur between database calls like 'SQL*Net message from client'.

BINDS are the values of the bind variables used in the sql statements.

Before proceeding to tracing another session, let's touch on the wonderful SqlPlus Autotrace facility.

SqlPlus Autotrace:

Oracle provides autotrace facility to provide the execution plan and some useful statistics for sql statements executed in your session in sqlplus. This is a nice little tool that can be used quickly without having to worry about accessing trace files and can be made available to all the developers.

Setup:

- create the PLAN_TABLE (to store explain plans) by executing @\$ORACLE_HOME/rdbms/admin/utlxplan.sql.
- Make Plan_table publicly available by creating a public synonym and the necessary Insert,Update,Delete privileges to either PUBLIC or selected users.
- Setup the PLUSTRACE role as follows:
 - Invoke sqlplus and connect as sys user
 - Run \$ORACLE_HOME/sqlplus/admin/plustrc.sql
 - Grant Plustrace to dba with admin option;
 - Grant plustrace to 'your user';

Once this is setup, it is as easy as running

```
SQL> set autotrace on
```

```
SQL> select count(*) from dual;
```

```
COUNT(*)
```

```
-----  
1
```

```
Execution Plan
```

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE
```

```
1  0  SORT (AGGREGATE)
```

```
2  1  TABLE ACCESS (FULL) OF 'DUAL'
```

```
Statistics
```

```
-----  
0 recursive calls
```

```
0 db block gets
```

```
3 consistent gets
```

```
0 physical reads
```

```
0 redo size
```

```
383 bytes sent via SQL*Net to client
```

```
503 bytes received via SQL*Net from client
```

```
2 SQL*Net roundtrips to/from client
```

```
0 sorts (memory)
```

```
0 sorts (disk)
```

```
1 rows processed
```

and to disable autotracing ,

```
set autotrace off
```

I'll just briefly explain about some of the statistics autotrace shows us.

Recursive calls:

those additional calls(sql) executed by Oracle implicitly to process your (user) sql statement.

Can be many things, hard parses,trigger executions , sort extent allocations , data dictionary lookups/updates etc

db block gets :

number of data blocks read in CURRENT mode ie) not in a read consistent fashion, but the current version of the data blocks. DML like Update,Delete will need to access the blocks in the current mode for modification.

consistent gets :

number of data blocks accessed in READ CONSISTENT mode. Inorder to maintain statement level read consistency, Oracle has to read the blocks in a consistent fashion (as of the snapshot SCN) and hence may fetch from rollback segments, which is also added to this statistic.

physical reads :

Physical(disk and/or filesystem page cache) reads. Basically those that cannot be satisfied by the cache and those that are direct reads.

redo size :

Redo (redo entries are necessary for instance/media recovery : consists of table/index/rollback segment/data dictionary & other changes) generated in bytes. The redo entries are written out to the online redolog files from the log buffer cache by LGWR.

There are a few options that you could exercise when using autotrace such as

set autotrace on explain : only the explain plan and the query result

set autotrace on statistics : only the result set and statistics. No explain plan

set autotrace traceonly : only the explain plan and statistics . No query result

set autotrace traceonly statistics : Only the statistics. No query result or explain plan

set autotrace traceonly explain : only the explain plan. No query result or statistics

Note that autotrace doesn't create trace files by itself.

Tracing a different session:

Setup:

The first thing you need inorder to trace another session is a unique identifier for that session. This is provided by SID, SERIAL# combination from V\$SESSION.

For eg

```
select sid,serial# from v$session where username='SCOTT' and machine='APOLLO';
```

Once you get the SID,SERIAL# of the session you want to trace, you may want to enable TIMED_STATISTICS and set MAX_DUMP_FILE_SIZE for that session ,if you do not have the luxury of setting them on the whole instance. We could do this by making use of a couple of procedures contained in the DBMS_SYSTEM package.

To turn on timed_statistics for the session identified by SID,SERIAL# (12,13) ,

```
exec sys.dbms_system.set_bool_param_in_session(
```

```
SID=>12,SERIAL#=>13,parnam=>'timed_statistics',bval=>true);
```

SET_BOOL_PARAM_IN_SESSION procedure of dbms_system is used to set BOOLEAN type init.ora parameters in a specific session.

To set max_dump_file_size parameter for the session identified by (12,13),

```
exec sys.dbms_system.set_INT_param_in_session(
```

```
SID=>12,SERIAL#=>13,parnam=>'max_dump_file_size', INTVAL=>10000);
```

SET_INT_PARAM_IN_SESSION procedure of dbms_system is used to set INTEGER type init.ora parameter in a specific session.

There are several ways to trace a different session. Let's start with DBMS_SYSTEM.

.To enable tracing of a session whose SID=12 and SERIAL#=13),

```
execute dbms_system.set_sql_trace_in_session(12,13,TRUE);
```

and to disable tracing

```
execute dbms_system.set_sql_trace_in_session(12,13,FALSE);
```

Although not commonly used for tracing,

DBMS_SYSTEM.SET_BOOL_PARAM_IN_SESSION procedure maybe used to turn on sql tracing, like this

```
exec dbms_system.set_bool_param_in_session(10,35,'sql_trace',true);
```

although I have rarely used this procedure for this purpose(ie sql tracing).

Lets see how we can use **DBMS_SUPPORT** package to trace another session. The general syntax is

```
DBMS_SUPPORT.START_TRACE_IN_SESSION(SID,SERIAL#,WAIT  
BOOLEAN,BINDS BOOLEAN);
```

So enable tracing with wait and bind information of a session with SID=12 and SERIAL#=13,

```
DBMS_SUPPORT.START_TRACE_IN_SESSION (12, 13,waits=>true,binds=>true);
```

```
DBMS_SUPPORT.START_TRACE_IN_SESSION( 12,13 );
```

will enable tracing for the session(12,13) including the wait information but no bind values.

and to disable tracing

```
DBMS_SUPPORT . STOP_TRACE_IN_SESSION( sid , serial# )
```

The trace output is similar to SQL_TRACE output but may include additional WAIT or BIND lines depending on the tracing options chosen.

Event based tracing:

Now, let's look at ways to enable tracing through events. Oracle provides event 10046 that can also aid us in collecting extended sql trace data.

There are four levels available when setting up trace with Event 10046:

- Level 1: this cause tracing of sql activities and is similar to 'alter session set sql_trace=true'
- Level 4: provides level 1 tracing + displays the values for all bind variables. It is equivalent to dbms_support.start_trace(waits=>false,binds=>true);
- Level 8: provides level 1 tracing and displays a list of all database wait events. It is equivalent to dbms_support.start_trace(waits=>true,binds=>false);
- Level 12 provides level 1 tracing in addition to both bind variable substitution and database wait events. It is equivalent to dbms_support.start_trace(waits=>true,binds=>true);

Note that Level 0 disables tracing.

To enable sql trace collection for your session at level 12,

```
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 12';
```

and to disable the session level tracing

```
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';
```

To enable extended sql trace for the whole instance to collect wait related information,

```
ALTER SYSTEM SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 8';
```

and to disable the instance wide tracing

```
ALTER SYSTEM SET EVENTS '10046 TRACE NAME CONTEXT off';
```

To set an event in a different session, we can make use of **DBMS_SYSTEM.SET_EV** procedure as follows:

To start tracing a session with SID=12, SERIAL#=13 with only bind values

```
dbms_system.set_ev(12,13,EV=>10046,LE=>4,NM=>");
```

and to disable tracing for that session ,

```
dbms_system.set_ev(12,13,EV=>10046,LE=>0,NM=>");
```

Also there's this undocumented utility **oradebug** that can used to trace another session by setting event 10046.

We can do this by attaching to the process either via Oracle process id(PID) or OS process id(SPID) from v\$process.

```
SQL>select p.pid,p.spid from v$process p,v$session s
```

```
2 where p.addr=s.paddr and s.username='SCOTT';
```

```
PID SPID
```

```
-----
```

```
17 2044
```

```
-- Attach to the process using oracle pid
```

```
SQL>oradebug setorapid 17
```

```
Windows thread id: 2044, image: ORACLE.EXE
```

```
-- set event 10046 for this process
```

```
SQL>oradebug event 10046 trace name context forever, level 12 ;
```

```
Statement processed.
```

```
-- let the session perform database operations that you want to analyze and then
```

-- disable tracing

```
SQL>oradebug event 10046 trace name context off
```

Statement processed.

Tracing with triggers:

In some cases, you may want to enable tracing right when a user connects to the database and only stop tracing when he/she disconnects. On such occasions, we can make use of database event level triggers to enable/disable tracing automatically.

For example, when you want to automatically enable tracing when user Scott logs into the database with a custom tracefile_ identifier, you can do something like this

```
SQL> create or replace trigger trace_trigger_scott
 2 AFTER LOGON ON DATABASE
 3 WHEN (USER='SCOTT')
 4 declare
  stmt varchar2(100);
  hname varchar2(20);
  uname varchar2(20);
  begin
 5  select sys_context('USERENV','HOST'),sys_context('USERENV','SESSION_USER')
into hname,uname from dual;
 6  stmt := 'alter session set tracefile_ identifier='||hname||'_'||uname;
 7  EXECUTE IMMEDIATE stmt;
  EXECUTE IMMEDIATE 'alter session set sql_trace=true';
 8  end;
 9 10 11 12 13 14
15 /
```

Trigger created.

and disable tracing when the user disconnects .

```
create or replace trigger trace_trigger_off
 BEFORE LOGOFF ON DATABASE
 when(user='SCOTT')
begin
 execute immediate 'alter session set sql_trace=false';
end;
/
```

Once you have diagnosed the issue,the trigger can be disabled or dropped.

All this time I talked about how to enable/disable sql tracing for Oracle sessions, let's briefly look at the trace file itself and the very useful Tkprof o/p before concluding this article.

Trace file contents/Tkprof:

Identification of trace file:

You'll need to identify the trace file before actually starting to analyze. As mentioned earlier, the trace files are generated in the directory specified by user_dump_dest in the specified format. You'll need to know this directory location and V\$process.SPID to locate the trace file of interest.

```
select value from v$parameter where name='user_dump_dest';
```

For example, to get the SPID of your local session (assuming you are tracing your session),

```
Select p.spid from v$process p, v$session s where p.addr=s.paddr and  
s.audsid=sys_context('USERENV','SESSIONID');
```

or you can get the complete trace file name using the following query :

Let's set tracefile_identifier on our session to help us identify the trace file easily.

```
SQL>alter session set tracefile_identifier=TEST;
```

Session altered.

```
SQL>select  
i.instance_name||'_ora_'||trim(to_char(p.spid))||decode(p.traceid,null,null,'_'||p.traceid)||'.trc'  
TRACEFILE  
2 from v$instance i,v$process p,v$session s where  
3 p.addr=s.paddr and s.audsid=sys_context('USERENV','SESSIONID');  
TRACEFILE
```

thiru_ora_2772_TEST.trc

You can modify the query accordingly for a different session (use v\$session.sid , v\$session.serial#).

Sql tracing with Shared servers: A word of caution when trying to locate/analyze trace files generated by sessions connected through shared servers (MTS): since the sql statements issued in the session connected through dispatcher/shared server can be processed by more than one Shared server, the trace information for that session is going to be scattered around in multiple trace files making it almost impossible to identify and analyze them. Also the same trace file will have sql statements from different sessions, because the processes are shared amongst sessions.

You may want to consider switching those sessions to 'dedicated' servers for the purpose of performance diagnostics temporarily, but again you need to take into account the performance difference between Shared and Dedicated connections (ie Shared servers have longer code path than dedicated Servers).

Now, lets look at a typical trace file generated on Windows platform.

```
Dump file c:\oracle\admin\thiru\udump\thiru_ora_1600.trc
Sat Oct 18 10:35:59 2003
ORACLE V9.2.0.1.0 - Production vsnsta=0
vsnsql=12 vsnxt=3
Windows 2000 Version 5.1 Service Pack 1, CPU type 586
Oracle9i Release 9.2.0.1.0 - Production
JServer Release 9.2.0.1.0 - Production
Windows 2000 Version 5.1 Service Pack 1, CPU type 586
Instance name: thiru
```

Redo thread mounted by this instance: 1

Oracle process number: 14

Windows thread id: 1600, image: ORACLE.EXE

```
*** 2003-10-18 10:35:59.000
*** SESSION ID:(11.3) 2003-10-18 10:35:59.000
*** 2003-10-18 10:43:24.000
SEARCH in kdisti: tsn = 8, objd = 31098, rdba = 33554531APPNAME mod='SQL*Plus' mh=3669949024 act=' ' ah=4029777240
```

```
=====
PARSING IN CURSOR #1 len=32 dep=0 uid=73 oct=42 lid=73 tim=25796157458 hv=3999951541 ad='65ca3d28'
alter session set sql_trace=true
END OF STMT
EXEC #1:c=0,e=136,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=25796156541
```

```
=====
PARSING IN CURSOR #1 len=94 dep=0 uid=73 oct=3 lid=73 tim=25802821809 hv=1358874256 ad='65d16d7c'
Select Sal from Emp E Where 3 >= (Select Count(Distinct sal) from Emp E1 Where E1.Sal >=E.Sal)
END OF STMT
PARSE #1:c=0,e=17269,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=25802821799
EXEC #1:c=0,e=61,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=25802823817
FETCH #1:c=10014,e=336,p=0,cr=5,cu=0,mis=0,r=1,dep=0,og=4,tim=25802824624
FETCH #1:c=0,e=417,p=0,cr=2,cu=0,mis=0,r=3,dep=0,og=4,tim=25802825918
STAT #1 id=1 cnt=4 pid=0 pos=1 obj=0 op='FILTER '
STAT #1 id=2 cnt=15 pid=1 pos=1 obj=31097 op='TABLE ACCESS FULL EMP '
STAT #1 id=3 cnt=13 pid=1 pos=2 obj=0 op='SORT GROUP BY '
STAT #1 id=4 cnt=107 pid=3 pos=1 obj=31098 op='INDEX RANGE SCAN EMP_SAL_IDX '
```

```
=====
PARSING IN CURSOR #1 len=33 dep=0 uid=73 oct=42 lid=73 tim=25811846923 hv=2993913867 ad='65c9cd74'
alter session set sql_trace=false
END OF STMT
PARSE #1:c=0,e=325,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=25811846914
EXEC #1:c=0,e=131,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=25811849066
```

As I mentioned earlier, analyzing the tracefile is beyond the scope of this article. Formatting the raw tracefile using Tkprof yields us the following output:

TKPROF: Release 9.2.0.1.0 - Production on Sat Oct 18 10:44:47 2003

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Trace file: thiru_ora_1600.trc
Sort options: default

count = number of times OCI procedure was executed
cpu = cpu time in seconds executing

elapsed = elapsed time in seconds executing
 disk = number of physical reads of buffers from disk
 query = number of buffers gotten for consistent read
 current = number of buffers gotten in current mode (usually for update)
 rows = number of rows processed by the fetch or execute call

alter session set sql_Trace=true

call	count	cpu	elapsed	disk	query	current	rows
Parse	0	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	0	0.00	0.00	0	0	0	0
total	1	0.00	0.00	0	0	0	0

Misses in library cache during parse: 0
 Misses in library cache during execute: 1
 Optimizer goal: CHOOSE
 Parsing user id: 73

Select Sal
 from
 Emp E Where 3 >= (Select Count(Distinct sal) from Emp E1 Where E1.Sal >= E.Sal)

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.01	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	2	0.01	0.00	0	7	0	4
total	4	0.01	0.01	0	7	0	4

Misses in library cache during parse: 1
 Optimizer goal: CHOOSE
 Parsing user id: 73

Rows Row Source Operation

 4 FILTER
 15 TABLE ACCESS FULL EMP
 13 SORT GROUP BY
 107 INDEX RANGE SCAN EMP_SAL_IDX (object id 31098)

alter session set sql_Trace=false

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	0	0.00	0.00	0	0	0	0
total	2	0.00	0.00	0	0	0	0

Misses in library cache during parse: 1
 Optimizer goal: CHOOSE
 Parsing user id: 73

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call	count	cpu	elapsed	disk	query	current	rows
------	-------	-----	---------	------	-------	---------	------

Parse	2	0.00	0.01	0	0	0	0
Execute	3	0.00	0.00	0	0	0	0
Fetch	2	0.01	0.00	0	7	0	4
total	7	0.01	0.01	0	7	0	4

Misses in library cache during parse: 2
Misses in library cache during execute: 1

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call	count	cpu	elapsed	disk	query	current	rows
Parse	0	0.00	0.00	0	0	0	0
Execute	0	0.00	0.00	0	0	0	0
Fetch	0	0.00	0.00	0	0	0	0
total	0	0.00	0.00	0	0	0	0

Misses in library cache during parse: 0

3 user SQL statements in session.
0 internal SQL statements in session.
3 SQL statements in session.

Trace file: thiru_ora_1600.trc
Trace file compatibility: 9.00.01
Sort options: default

1 session in tracefile.
3 user SQL statements in trace file.
0 internal SQL statements in trace file.
3 SQL statements in trace file.
3 unique SQL statements in trace file.
45 lines in trace file.

Note that since the trace files are located on the Database server, they are typically available only to the DBAs who have local Server access. On Development/Test databases where developers do need access to the trace files for performance analysis, they can be made available by setting the undocumented init.ora parameter `_TRACE_FILES_PUBLIC=true`. This makes the trace files readable by everybody who has OS level access on the database server.

and that concludes this article. Happy tracing everyone!

About the Author: Thiru Vadivelu is a Senior Oracle DBA at Advance Magazine Group, Delaware, USA. He has been working as Sr. Database Admin/Performance tuning Analyst at various clients/companies in USA. He can be reached at oracle-dba@comcast.net