

# ORACLE TRACE ANALYSIS ON STEROIDS

## AN IN-DEPTH LOOK AT TRACE ANALYZER

*Dave Moore, DBI*

### OVERVIEW

The process of analyzing Oracle trace files is an arduous task. Even though the raw trace files generated by Oracle Trace are readable, automation is useful to present the trace data in a more meaningful format. TKPROF is the most well known tool for this task and is distributed (as an executable) with each version of the database. However, even when using TKPROF, there are often issues that Oracle Professionals should know and research before concluding actual problems and solutions.

Trace Analyzer is a lesser known utility that analyzes Oracle trace (.trc) files. It is the most comprehensive tracing tool available and has many advantages over other methods including TKPROF. Trace Analyzer was originally created to assist debugging of Oracle Applications although the benefit of the tool is application independent. It is in the form of Oracle packages and is not yet distributed with the database. It is also open source. The Trace Analyzer packages are not wrapped, they are there for all to see, learn, and change. I have learned a ton about Oracle tracing by simply reviewing the PL/SQL code for Trace Analyzer.

The objective of this paper is to review the features of Trace Analyzer and determine how it can best be used in your environment.

### INSTALLATION

Trace Analyzer is available as a MetaLink ([metalink.oracle.com](http://metalink.oracle.com)) download (note #224270.1). The simple zip file download contains the following files:

**tacpkg.sql** – creates the trace analyzer package by calling other files below.

**tacpkgd.pks** – creates the trca\$d package specification.

**tacpkgd.pkb** – creates the trca\$d package body.

**tacpkgi.pks** – creates the tca\$i package specification.

**tacpkgi.pkb** – creates the trca\$i package body.

**tacreate.sql** – main creation SQL script. This script will invoke others to create the TRCANLZR user and objects needed.

**tactab.sql** – SQL script to create the Trace Analyzer tables.

**tacusr.sql** – creates the TRCANLZR user and required Oracle directory object.

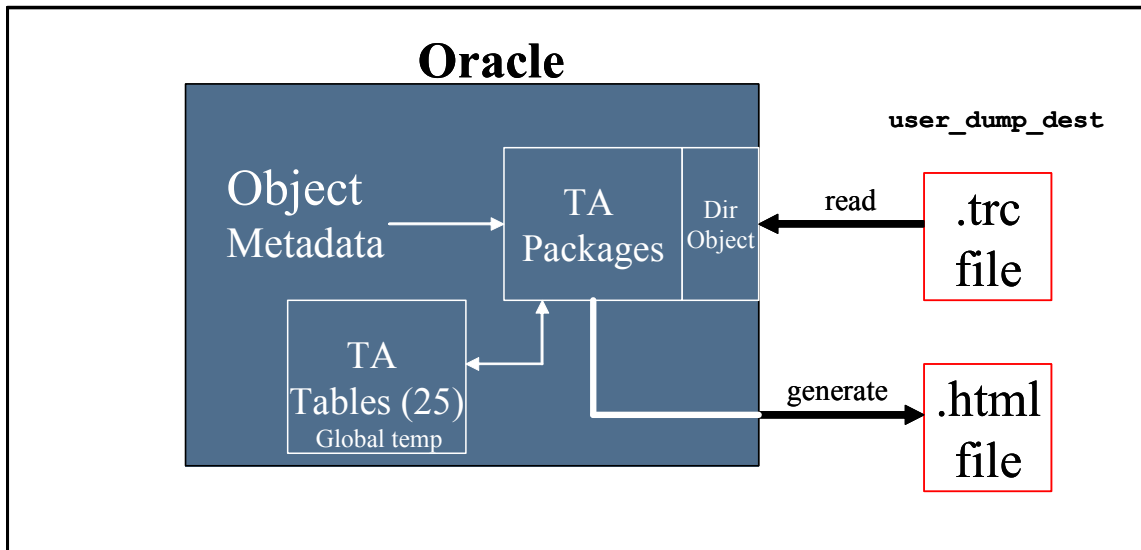
**tadrop.sql** – master drop script which calls the other drop scripts.

**tadtab.sql** – drops the tables.

**tadusr.sql** – drops the user.

**trcanlzs.sql** – main execution script.

## Database Server Machine



The architecture of TA (above) shows packages inside of Oracle that read from the user\_dump\_dest via an Oracle directory object. These packages read data from the trace file, gather statistics from the data dictionary and produce the resulting HTML report.

### PRIVILEGES

Trace Analyzer privileges fall into three different categories:

- Install
- Configuration
- Usage

The SYSDBA privilege is required to install Trace Analyzer. Part of this installation process involves creating the TRCANLZR user and granting the privileges below. These privileges are needed for the PL/SQL packages to access all of the objects needed for comprehensive trace file analysis. And since they are granted to the TRCANLZR user, the users executing the utility do not need any specific privileges. The TRCANLZR user password should remain confidential given that it is a privileged account. Other users permissions are expected to remain the same.

- create session
- alter session
- create table
- create view
- create sequence
- create public synonym
- drop public synonym
- read on directory
- write on directory
- select\_catalog\_role

- select on v\_\$session
- select on v\_\$parameter2
- select on v\_\$event\_name
- select on v\_\$latchname
- select on audit\_actions
- select on dba\_extents
- select on dba\_temp\_files

Once the TRCANLZR user is created and the privileges have been granted to this user, the install script will connect as TRCANLZR and create the necessary objects in the database. It will prompt for a tablespace for which to assign the objects. DBAs may wish to create one before installation and assign the TRCANLZR objects to the newly created tablespace. Given that almost all of the tables are created with 'GLOBAL TEMPORARY', there are no real storage concerns. All staging tables used by Trace Analyzer retain the data of one or more consecutive executions as long as the session that started the report is still alive. Once the session terminates, the data in these temporary tables is gone. There is currently only one permanent table (TRCA\$\_TRACE) which contains audit data – times, dates, users, trace files, etc. If a DBA desires for the tables to be permanent or would like to create a wrapper tool on top of Trace Analyzer, the install script tactab.sql can be modified. The keywords GLOBAL TEMPORARY and ON COMMIT PRESERVE ROWS should be deleted.

Once installed, the execution of the TRCA packages is granted to public. Anyone can use it and *should be* encouraged to do so.

## DATABASE OBJECTS

Most DBAs want to know exactly what is going to be created in the database with the installation of a new product or utility. Below is the list of database objects created by Trace Analyzer during installation. All schema objects created reside in the TRCANLZR schema.

### TABLES

TRCA\$_TRACE	TRCA\$_CURSOR_TABLES
TRCA\$_AUDIT_ACTIONS	TRCA\$_PLAN_TABLE
TRCA\$_IDLE_EVENT	TRCA\$_EXEC_SUMMARY
TRCA\$_CALL_SUMMARY	TRCA\$_WAIT_SUMMARY
TRCA\$_WAIT	TRCA\$_STAT_SUMMARY
TRCA\$_BIND	TRCA\$_HOT_BLOCK
TRCA\$_ERROR	TRCA\$_CURSOR_WAIT_LATCH
TRCA\$_GAP	TRCA\$_EXTENTS
TRCA\$_XCTEND	TRCA\$_IND_PARTITIONS
TRCA\$_STAT	TRCA\$_TAB_PARTITIONS
TRCA\$_CALL	TRCA\$_INDEXES
TRCA\$_CURSOR	TRCA\$_CURSOR_WAIT_SEGMENT
TRCA\$_TABLES	

### PACKAGES

TRCA\$D – This package is 5113 lines of PL/SQL and is the guts of Trace Analyzer.

TRCA\$I – This package is 604 lines of PL/SQL and is used for the data expansion process – explain plans, segment statistics, etc.

Also created by Trace Analyzer is a sequence (TRCA\$\_TRACE\_ID\_S) and a view (TRCA\$\_CURSOR\_V).

## EXECUTION

Below is the DESCRIBE results from the package, in particular the procedure call to TRACE\_ANALYZER.

PROCEDURE TRACE_ANALYZER	Argument Name	Type	In/Out	Default?
	P_TRACE_FILENAME	VARCHAR2	IN	
	P_OUTPUT_FILENAME	VARCHAR2	IN	DEFAULT
	P_TOP_SQL_THRESHOLD	NUMBER (38)	IN	DEFAULT
	P_TOP_EXEC_THRESHOLD	NUMBER (38)	IN	DEFAULT
	P_IO_TIME_THRESHOLD	NUMBER (38)	IN	DEFAULT
	P_LATCH_TIME_THRESHOLD	NUMBER (38)	IN	DEFAULT
	P_HOT_BLOCK_THRESHOLD	NUMBER (38)	IN	DEFAULT
	P_DEC_DIGITS_SECONDS	NUMBER (38)	IN	DEFAULT
	P_FORCE_DATA_EXPANSION	VARCHAR2	IN	DEFAULT
	P_SKIP_WAITS	VARCHAR2	IN	DEFAULT
	P_SKIP_BINDS	VARCHAR2	IN	DEFAULT
	P_SKIP_EXPLAIN_PLANS	VARCHAR2	IN	DEFAULT
	P_SKIP_COUNT_STAR	VARCHAR2	IN	DEFAULT
	P_DESTINATION	VARCHAR2	IN	DEFAULT

The only required parameter is the P\_TRACE\_FILENAME. The other parameters are described below:

P_TRACE_FILENAME	This is the name of the trace file to process.
P_OUTPUT_FILENAME	The output (report) name can optionally be specified otherwise one will be created automatically. The default name begins with "trcanl\$r" and ends with ".html" but also contains the PID and the trace (sequence number).
P_TOP_SQL_THRESHOLD	This is the number of SQL statements that will be expanded upon in the analysis. It defaults to the value of trca\$d.top_sql_threshold (currently 20).
P_TOP_EXEC_THRESHOLD	This is the number of executions per cursor to report. It defaults to the value of trca\$d.top_exec_threshold (currently 10). This restricts the output to the 10 most significant executions of a cursor based on execution time. When bind variables exist, they will also be reported.
P_IO_TIME_THRESHOLD	Will report IO time only if > this value – default is 5. Units are seconds.
P_LATCH_TIME_THRESHOLD	Will display latch time only if > # seconds. The default is 1 second.
P_HOT_BLOCK_THRESHOLD	Number of hot blocks to report. The default is 5.
P_DEC_DIGITS_SECONDS	Number of decimal digits to report seconds. The default is 3.
P_FORCE_DATA_EXPANSION	Will display explain plan and detailed segment information. The default is 'N' – data expansion will not be forced.
P_SKIP_WAITS	Will skip displaying wait event information. The default is 'N', as in 'No', they will not be skipped.
P_SKIP_BINDS	Will skip displaying bind variable information. The default is 'N'.
P_SKIP_EXPLAIN_PLANS	Will skip displaying Explain Plan information. The default is 'N'.
P_SKIP_COUNT_STAR	Will skip displaying the row counts. The default is 'N'. Set this option to 'Y' if you are confident that your statistics are up to date and your actual row counts closely match those generated by statistics.
P_DESTINATION	Default value is 'UDUMP' and this value should not be changed. However, the DBA may wish to change where UDUMP points to via CREATE OR REPLACE DIRECTORY if they have trace files to process in another directory. Another option for this parameter is 'SCREEN' which would allow the execution of Trace Analyzer from a URL provided that the database has that feature enabled.

The actual execution of Trace Analyzer is easy:

```
SQL> set serveroutput on
SQL> exec trca$i.trace_analyzer('orcl_ora_118a.trc');
```

If serveroutput is on, Trace Analyzer will display feedback to the screen – either the error that occurred or the name of the report from a successful execution. Remember that the installation of Trace Analyzer created a directory object pointing to the *user\_dump\_dest*. The trace file referenced in the command must be in this directory on the database server. Otherwise you'll get the following error:

```
Trace file orcl_ora_118a.trc not found on C:\ORACLE\ADMIN\ORCL\UDUMP
```

## COMPARISONS & OUTPUT ANALYSIS

Before analyzing the output and comparing Trace Analyzer to tools like TKPROF, let's start with the raw trace file contents for a given SQL statement. We can reference this as we review what TKPROF and Trace Analyzer do with the raw trace data.

```

PARSING IN CURSOR #7 len=338 dep=0 uid=69 oct=3 lid=69 tim=83538801262 hv=3748129421 ad='696495a0'
select a.first_name, a.last_name,
       a.title_id, a.region_id, b.pay_grade, c.description, d.amount
from   demosql.employee a,
       demosql.title b,
       demosql.region c,
       demosql.bonus d
where  a.title_id = b.title_id
       and a.region_id = c.region_id
       and a.employee_id = d.employee_id
       and a.employee_id = d.employee_id
       and d.amount = 6000
END OF STMT
PARSE #7:c=190273,e=1125067,p=0,cr=67,cu=0,mis=1,r=0,dep=0,og=1,tim=83538801251
BINDS #7:
EXEC #7:c=0,e=4258,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=83538864314
WAIT #7: nam='SQL*Net message to client' ela= 9 p1=1111838976 p2=1 p3=0
WAIT #7: nam='db file sequential read' ela= 16700 p1=6 p2=585 p3=1
WAIT #7: nam='db file sequential read' ela= 248 p1=6 p2=586 p3=1

```

Note – The raw trace output above was shortened for readability purposes for this paper.

## TKPROF

Probably one of the best ways to describe Trace Analyzer and what it can do is to compare it to the well known TKPROF utility. TKPROF is tried and true and used by many Oracle professionals around the world. As an operating system executable, it resides in \$ORACLE\_HOME/bin and is simple to execute. TKPROF has many options that aid in analyzing a trace file. The familiar TKPROF output is as follows:

```

select a.first_name, a.last_name,
       a.title_id, a.region_id, b.pay_grade, c.description, d.amount
from   demosql.employee a,
       demosql.title b,
       demosql.region c,
       demosql.bonus d
where  a.title_id = b.title_id
       and a.region_id = c.region_id
       and a.employee_id = d.employee_id
       and a.employee_id = d.employee_id
       and d.amount = 6000

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.05	0.17	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	57	0.03	0.12	2	300	0	832
total	59	0.08	0.29	2	300	0	832

```

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 69

```

```

Rows      Row Source Operation
-----
832      HASH JOIN      (cr=300 pr=2 pw=0 time=146596 us)
1000     TABLE ACCESS FULL TITLE (cr=5 pr=0 pw=0 time=4187 us)
832      HASH JOIN      (cr=295 pr=2 pw=0 time=134086 us)
125      TABLE ACCESS FULL REGION (cr=3 pr=2 pw=0 time=95768 us)
832      HASH JOIN      (cr=292 pr=0 pw=0 time=26634 us)
832      TABLE ACCESS FULL BONUS (cr=47 pr=0 pw=0 time=2700 us)
15000    TABLE ACCESS FULL EMPLOYEE (cr=245 pr=0 pw=0 time=60128 us)

```

## TRACE ANALYZER OUTPUT

Trace Analyzer reads in a raw trace file, parses it and loads data into temporary tables. These tables are accessed in order to generate the HTML report. The data in these tables disappear when the Trace Analyzer session exits.

The Trace Analyzer report begins with a header section followed by the SQL statement. Notice the rank in the header statement, its formula is explained later.

hash:3748129421 length:338 depth:0(non-recursive) user:69(user) line:5300 rank:9 [next prior top](#)

```
select a.first_name, a.last_name,
       a.title_id, a.region_id, b.pay_grade, c.description, d.amount
from demosql.employee a,
     demosql.title b,
     demosql.region c,
     demosql.bonus d
```

The next section is the call summary – nothing really new here compared to TKPROF.

### Call Summary

Call Type	Call Count	CPU Time (secs)	Elapsed Time (secs)	Disk (blocks)	Query (blocks)	Current (blocks)	Rows Returned	Cache Misses
Parse:	1	0.190	1.125	0	67	0	0	1
Execute:	1	0.000	0.004	0	0	0	0	0
Fetch:	57	0.080	0.285	2	300	0	832	0
Total:	59	0.270	1.414	2	367	0	832	1

Next is the Waits section where both non-idle and idle waits are listed. The P\_SKIP\_WAITS option set to [N] will skip this section of the report. However it skips all waits – idle and non-idle. I would prefer that the utility gave me the option of skipping only the idle waits.

### Non-idle Wait Events

Non-idle Wait Event	Times Waited	Max Wait (secs)	Total Time Waited (secs)
db file sequential read:	2	0.017	0.017
Total non-idle:	2	0.017	0.017

### Idle Wait Events

Idle Wait Event	Times Waited	Max Wait (secs)	Total Time Waited (secs)
SQL*Net message from client:	57	0.042	0.180
SQL*Net message to client:	57	0.000	0.001
Total idle:	114	0.042	0.181

The core I/O Waits section expands on the P1, P2, P3 values used commonly in wait-based performance tuning. In the prior section we noticed db file sequential read waits. This section shows us which segment(s) are involved.

### Core I/O Waits Summary

Owner	Segment	Partition	Type	Tablespace	I/O Wait Event	Times Waited	Max Wait (secs)	Total Time Waited (secs)
DEMO SQL	REGION		TABLE	DEMO SQL 1	db file sequential read	2	0.017	0.017

The next section of the report shows the real execution plan of the statement. No real additional information over TKPROF but more nicely documented and formatted.

#### Row Source Plan (Real Execution Path)

Rows Returned	Exec Order	Row Source Operation	Object Id	Logical Reads (blocks)	Physical Reads (blocks)	Physical Writes (blocks)	Elapsed Time (secs)
832	7	HASH JOIN	0	300	2	0	0.000
1000	1	TABLE ACCESS FULL TITLE	62061	5	0	0	0.000
832	6	HASH JOIN	0	295	2	0	0.000
125	2	TABLE ACCESS FULL REGION	62055	3	2	0	0.000
832	5	HASH JOIN	0	292	0	0	0.000
832	3	TABLE ACCESS FULL BONUS	62039	47	0	0	0.000
15000	4	TABLE ACCESS FULL EMPLOYEE	62043	245	0	0	0.000

The next section of the report shows the Explain Plan output. By default, Trace Analyzer does not produce an Explain Plan (and all supporting data) if it detects that the trace file being analyzed was generated from a different instance than where it is executing. Otherwise, Explain Plans would be generated from environments other than where the statement was executed and that would not present an accurate picture.

I don't find this section very useful when the actual execution plan exists, as we examined in the previous section. However, this can be helpful when the statement has not completed and therefore the actual execution plan does not exist yet. The DBA has the option of forcing plans to be generated no matter what – by setting P\_FORCE\_DATA\_EXPANSION to [Y]. Also, keep in mind that the data expansion process provides extended data details – it's all or nothing. My experience has been that you're better off expanding the data and just skipping over the Explain Plan output most of the time. The Explain Plan output also includes execution order which is handy for non-expert users.

#### Explain Plan (Generated by Trace Analyzer)

Id	Exec Order	Explain Plan Operation
0:	8	SELECT STATEMENT
1:	7	HASH JOIN
2:	1	TABLE ACCESS FULL DEMO SQL.TITLE
3:	6	HASH JOIN
4:	2	TABLE ACCESS FULL DEMO SQL.REGION
5:	5	HASH JOIN
6:	3	TABLE ACCESS FULL DEMO SQL.BONUS
7:	4	TABLE ACCESS FULL DEMO SQL.EMPLOYEE

Id	Object Type	Parent Id	Pos	Obj Inst	Search Cols	Estim Card	Estim Bytes	Cost	CPU Cost	IO Cost	Time	Temp Space
0:			60			1667	83350	60	23609594	58		
1:		0	1			1667	83350	60	23609594	58		
2:	TABLE	1	1	2		1000	5000	2	151364	2		
3:		1	2			1667	75015	58	17712328	56		
4:	TABLE	3	1	3		125	2250	2	23371	2		
5:		3	2			1667	45009	55	12074304	54		
6:	TABLE	5	1	4		1667	10002	11	1613343	11		
7:	TABLE	5	2	1		15000	315000	43	3281709	43		

Id	Column Name	Column Value
1:	Access Predicates	"A"."TITLE ID"="B"."TITLE ID"
3:	Access Predicates	"A"."REGION ID"="C"."REGION ID"
5:	Access Predicates	"A"."EMPLOYEE ID"="D"."EMPLOYEE ID"
6:	Filter Predicates	"D"."AMOUNT"=6000

The next section contains the table information. Any table referenced in the execution plan will be expanded in this section. The current count(\*) column can be skipped by using the P\_SKIP\_COUNT\_STAR parameter. There is an obvious performance penalty for using this option and it is unneeded if the table statistics are fresh. However, this is one way to verify that the row counts are fairly accurate. The rest of the data is from Oracle optimizer statistics and is very useful when analyzing execution plans.

#### Tables Referenced by Explain Plan

Owner.Table Name	Current COUNT(*)	Num* Rows	Sample* Size	Last* Analyzed	Avg* Row Len	Chain* Cnt	Blocks*	Empty* Blocks	Avg* Space	Global* Stats
DEMOSQL.BONUS	5000	5000	5000	26-JAN-06 09:19:37	60	0	44	3	1080	NO
DEMOSQL.EMPLOYEE	15000	1500 0	15000	26-JAN-06 09:19:38	93	0	187	68	454	NO
DEMOSQL.REGION	125	125	125	26-JAN-06 09:19:39	22	0	1	6	5136	NO
DEMOSQL.TITLE	1000	1000	1000	26-JAN-06 09:19:40	21	0	3	12	514	NO

(\*) Columns marked with an asterisk are refreshed only when collecting CBO statistics

The next section gives more table information – the storage attributes for each.

Owner.Table Name	Temporary	Partitioned	Logging	Degree	Cache	IOT Type	Ini Trans	Max Trans	Freelist Groups	Freelists
DEMOSQL.BONUS	N	NO	YES	1	N		1	255	1	1
DEMOSQL.EMPLOYEE	N	NO	YES	1	N		1	255	1	1
DEMOSQL.REGION	N	NO	YES	1	N		1	255	1	1
DEMOSQL.TITLE	N	NO	YES	1	N		1	255	1	1

The next section displays the index information. All indexes are described by the columns below:

#### Indexes on Tables Referenced by Explain Plan

Table Owner.Table Name	Index Owner.Index Name	Index Type	Uniqueness	Indexed Columns
DEMOSQL.BONUS	DEMOSQL.PK_BONUS_DEPT	NORMAL	NONUNIQUE	DEPT ID
DEMOSQL.EMPLOYEE	DEMOSQL.PK_DEPARTMENT_ID	NORMAL	NONUNIQUE	DEPARTMENT_ID
DEMOSQL.EMPLOYEE	DEMOSQL.PK_EMPLOYEE_ID	NORMAL	UNIQUE	EMPLOYEE_ID
DEMOSQL.EMPLOYEE	DEMOSQL.UK_TAXPAYER_ID	NORMAL	UNIQUE	TAXPAYER_ID
DEMOSQL.REGION	DEMOSQL.PK_REGION_ID	NORMAL	UNIQUE	REGION_ID
DEMOSQL.TITLE	DEMOSQL.PK_TITLE_ID	NORMAL	UNIQUE	TITLE_ID

The next section is the index statistics information. Similar to the table section above, this section shows the statistics for all indexes.

Table Owner.Table Name	Index Owner.Index Name	Num* Rows	Sample Size	Last Analyzed	Distinct Keys	Level*	Leaf Blcks	Avg Leaf Blks per Key	Avg Data Blks per Key	Clustering Factor	Global Stats
DEMOSQL.BONUS	DEMOSQL.PK_BONUS_DEPT	5000	5000	26-JAN-06 09:19:37	4	1	14	3	41	164	YES
DEMOSQL.EMPLOYEE	DEMOSQL.PK_DEPARTMENT_ID	15000	15000	26-JAN-06 09:19:38	200	1	29	1	75	15000	NO
DEMOSQL.EMPLOYEE	DEMOSQL.PK_EMPLOYEE_ID	15000	15000	26-JAN-06 09:19:38	15000	1	29	1	1	187	NO
DEMOSQL.EMPLOYEE	DEMOSQL.UK_TAXPAYER_ID	15000	15000	26-JAN-06 09:19:38	15000	1	40	1	1	187	NO
DEMOSQL.REGION	DEMOSQL.PK_REGION_ID	125	125	26-JAN-06 09:19:39	125	0	1	1	1	1	NO
DEMOSQL.TITLE	DEMOSQL.PK_TITLE_ID	1000	1000	26-JAN-06 09:19:40	1000	1	2	1	1	3	NO

(\* ) Columns marked with an asterisk are refreshed only when collecting CBO statistics

The next section gives more index information – the storage attributes for each.

Table Owner.Table Name	Index Owner.Index Name	Temp	Partitioned	Log	Degree	Ini Trans	Max Trans	Freelist Groups	Freelists	Status	Domid x Status	Fncid x Status
DEMOSQL.BONUS	DEMOSQL.PK_BONUS_DEPT	N	NO	YES	1	2	255	1	1	VALID		
DEMOSQL.EMPLOYEE	DEMOSQL.PK_DEPARTMENT_ID	N	NO	YES	1	2	255	1	1	VALID		
DEMOSQL.EMPLOYEE	DEMOSQL.PK_EMPLOYEE_ID	N	NO	YES	1	2	255	1	1	VALID		
DEMOSQL.EMPLOYEE	DEMOSQL.UK_TAXPAYER_ID	N	NO	YES	1	2	255	1	1	VALID		
DEMOSQL.REGION	DEMOSQL.PK_REGION_ID	N	NO	YES	1	2	255	1	1	VALID		
DEMOSQL.TITLE	DEMOSQL.PK_TITLE_ID	N	NO	YES	1	2	255	1	1	VALID		

## TRACE ANALYZER BEST PRACTICES

### SCHEMA

It is important that Trace Analyzer is executed as the same user (which is the schema owner of the SQL) and the same instance that generated the trace file. If so, additional information regarding segments will be provided. It is only in this scenario that a comprehensive picture is provided by Trace Analyzer. This is when the data expansion process will occur and gather data on tables, indexes, partitions, etc.

## OPEN ACCESS

Developers should be using this tool particularly in development environments. As a DBA I would encourage the use of Trace Analyzer by developers and Development DBAs. It is a great learning tool from which the developers can develop better SQL and be more efficient in debugging their own code before it goes into production. Just reading the source code provides training for Oracle Tracing.

## TIMELY ANALYSIS

One of the best things about Trace Analyzer is that it gathers additional information about the trace from the database itself. However, these database statistics can and will change over time. Therefore, Trace Analyzer should be executed against the trace file immediately after the trace file was generated.

## STATEMENT RANKING

Statement Ranking is used by Trace Analyzer to rank SQL statements by certain criteria. This ranking appears in the output of Trace Analyzer but more importantly is used to determine the top SQL statements to analyze in detail. It is obvious from the code below how this ranking is determined.

```

UPDATE trca$_cursor
  SET ranking = ROUND((
    (i.cpu_secs * CPU_FACTOR) +
    (i.elapsed_secs * ELAPSED_FACTOR) +
    (i.non_idle_wait_secs * NON_IDLE_WAIT_FACTOR) +
    (i.idle_wait_secs * IDLE_WAIT_FACTOR) +
    (i.logical_reads * LOGICAL_READS_FACTOR) +
    (i.physical_reads * PHYSICAL_READS_FACTOR)
  ) * BIG_FACTOR / total_time)
WHERE trace_id = p_trace_id
  AND cursor_id = i.cursor_id;

```

```

CPU_FACTOR           CONSTANT INTEGER := 32;
ELAPSED_FACTOR       CONSTANT INTEGER := 16;
NON_IDLE_WAIT_FACTOR CONSTANT INTEGER := 8;
IDLE_WAIT_FACTOR     CONSTANT INTEGER := 4;
LOGICAL_READS_FACTOR CONSTANT INTEGER := 2;
PHYSICAL_READS_FACTOR CONSTANT INTEGER := 1;
BIG_FACTOR           CONSTANT INTEGER := 1000000;

```

These factors are currently hard-coded in the PL/SQL package. Since I would not use the same weights all the time, I would prefer to make these optional input parameters when launching Trace Analyzer. Then I could assign factors as I wish or fall back on the defaults which I could change them as well.

## INCONSISTENT DATA

There are certain times in which the report data generated from the same trace file may differ between Trace Analyzer and TKPROF. Usually these discrepancies can be ignored. However, if the gaps are significant and you are relying on these numbers for tuning advice, you may need to consult the raw trace file directly for verification.

## SAVE TO CLIENT

One of the challenges with the Trace Analyzer architecture is that it relies on server-side operating system directories. Even though the package has public execute access, someone would need access to the *user\_dump\_dest* directory on the database server host machine in order to view the Trace Analyzer report. To get around this issue, the command below will generate

the report and copy it to the local directory on the client machine. If launching SQL\*Plus from Windows, the directory will default to %ORACLE\_HOME%\BIN.

```
SQL> start c:\davedata\trca\trcanlzt.sql orcl_ora_3320.trc
...generating report

Trace Analyzer Report "trcanlzt_3320_01.html" has been created in
C:\ORACLE\ADMIN\ORCL\UDUMP

...copying report
...report "trcanlzt_3320_26.html" has been copied into local directory
...exiting now (hit enter key)
```

## Using TRCSSESS

If your *user\_dump\_dest* is like most, it is littered with tons of trc files. Tracing activity can generate a lot of trace file data and Oracle will store this data in multiple files. In the past, we've needed to run TKPROF or Trace Analyzer against the individual files but this approach does not scale when there is a large number of them. Have no fear – trcsess to the rescue!

Trcsess is a utility that combines multiple trace files into one file based on the criteria specified. The utility resides in \$ORACLE\_HOME/bin and is available in version 10G of Oracle. It's a script that calls a Java program (oracle.ss.tools.trcsess.TrcSess) to do the work of consolidating the trace data.

To get the full picture of trcsess, we must start with new entries that Oracle puts in the trace file (below). These values are set by the application through the *dbms\_session.set\_identifier('client\_id')* and *dbms\_application\_info.set\_module('module', 'action')* procedures.

```
*** ACTION NAME: (DAVE_ACTION) 2006-01-10 20:06:22.396
*** MODULE NAME: (DAVE_MODULE) 2006-01-10 20:06:22.396
*** SERVICE NAME: (SYS$USERS) 2006-01-10 20:06:22.396
*** SESSION ID: (148.49932) 2006-01-10 20:06:22.386
*** CLIENT ID: (HERE IS THE CLIENTID) 2006-01-11 07:57:45.135
```

The command below will take all trace files with a module of SQL\*Plus and consolidate that into one trace file called all.trc. Nice!

```
$ trcsess output=all.trc module=SQL*Plus *.trc
```

Functionality provided by trcsess is extremely useful when dealing with many files that need to be analyzed. While trcsess is capable of combining hundreds of files into one, approach this utility with caution since both TKPROF and Trace Analyzer will experience memory issues with very large files. Also, not that on Linux and you must escape \$ characters with a backslash and put the string within double quotes. For instance:

```
SERVICE="SYS\$USERS"
```

## SUMMARY

At the end of the day, whatever tool helps solve the problem the quickest and the best is the one that should be utilized. TKPROF is ubiquitous just like *vi* – it's always there for you. And TKPROF executes more quickly than Trace Analyzer. However the difference in execution times can be easily explained - Trace Analyzer does more analysis than TKPROF. It uses all of the data in the trace file and then gathers additional information from the database to provide a comprehensive tuning picture. There are many times I've executed TKPROF and then needed to run additional queries against the database to further pinpoint the issue. Trace Analyzer does this automatically.

The database object requirements of Trace Analyzer can present a challenge when tuning many databases since the Trace Analyzer objects must be installed on each database. However, given the ease of setup and configuration, this is a small price to pay for the benefit received. Oracle could make this easier on us all by supplying it with the database and having the package already installed as part of the standard installation. Since it utilizes the database to gather data, Trace Analyzer requires a database connection to do its thing. If all you prefer is quick analysis with no database connection required, then continue to use TKPROF. If you desire more comprehensive analysis, then Trace Analyzer is the way to go.

The thing that puts Trace Analyzer in the winners circle is that it is PL/SQL and open source. Knowing that I can read the code and learn what it is doing and change it if I want to suit my needs (and I have) is golden. The downside is that once you change it, it's yours. The author of the tool welcomes constructive feedback and has been very responsive to my requests. His contact information is distributed within the instructions.txt file included in the zip file.