

Select Magazine - September 2001

Volume 7, No. 5

TKPROF - Your "Get out of Jail" Free Card
Kenneth M. Guion, QED Solutions, Incorporated**Introduction**

All too often the database or hardware shoulders the blame for poor application performance when the culprit is almost decidedly the application program itself. It would be nice if tweaking the database would provide substantial performance gains, but the fact remains that tuning the individual application SQL statements achieves the largest gain.

This paper is based on Oracle8i Release 2 (8.1.6). Many of the commands and options explained in this paper are also available in Oracle 8.0 and earlier, and TKPROF itself has been around since Oracle 6. This paper will not only focus on mainstream features of TKPROF and SQL Trace, but will also cover some advanced tracing techniques not documented in the Oracle manuals.

Using SQL Trace and TKPROF ("Transient Kernel Profile")

Typically only a few statements in an application or program are the cause of most of the performance problems. A combination of SQL Trace and TKPROF allows the developer to work smarter by easily identifying these statements. During program execution, the SQL Trace facility dumps execution and resource usage to an effectively unusable output file. The TKPROF utility then uses this output file to produce an easy to comprehend report identifying the following:

- each statement executed
- the resources the statement has consumed
- the number of times the statement was called, and the number of rows the statement processed

These tools also report CPU usage, elapsed time, physical disk reads, logical reads for the parse, execute, and fetch phases for each SQL statement.

With this information developers can easily determine the offending statements that are consuming the most resources. These tools can also be used to help discover what individual statements are being executed by packaged applications. Developers can even use tracing to discover what values are actually being passed to a SQL statement by a particular program or user.

There are four steps to effectively use these tools.

Step 1: Preparing to Trace

Step 2: Creating the Trace File

Step 3: Creating the TKPROF Report

Step 4: Understanding the TKPROF Report

Step 1: Preparing to Trace

In order to effectively use SQL Trace and TKPROF, the following three parameters must be set correctly. With each release, Oracle has made it easier to set these parameters. With release 8.1.6, all three of these parameters are now dynamic and can be set at the system or session level. Prior to 8.1.6 some of these parameters could only be set at the system level and could not be set dynamically and therefore required changes to the init.ora file and for the database to be restarted.

1. `TIMED_STATISTICS` must be set to `TRUE`. This causes the database to perform extra low-level calls and will have a slight impact on the performance and therefore should be set to `FALSE` when these statistics are not needed.
2. `MAX_DUMP_FILE_SIZE` should be set to the largest trace dump file (in operating system blocks) you want created. If a trace file tries to grow pass this limit, the database silently stops writing further trace information for that process.
3. `USER_DUMP_DEST` is the path where the trace files are to be created. This directory should be cleaned routinely. (Although the Oracle Doc in 8.1.6 says that this is a session level parameter, it isn't on my NT version.)

Set these parameters using the `ALTER SYSTEM` or `ALTER SESSION` command or alternatively setting their values in the `init.ora` file.

Here is an example of setting these parameters dynamically.

```
SQL> -- Test setting instance parameters dynamically

SQL>

SQL> ALTER SESSION SET TIMED_STATISTICS = TRUE;

Session altered.

SQL> ALTER SESSION SET MAX_DUMP_FILE_SIZE = 100000;

Session altered.

SQL> ALTER SESSION SET USER_DUMP_DEST = 'c:\oracle\admin\ora816\udump';

ALTER SESSION SET USER_DUMP_DEST = 'c:\oracle\admin\ora816\udump'

      *

ERROR at line 1:

ORA-02096: specified initialization parameter is not modifiable with this
option

SQL> ALTER SYSTEM SET USER_DUMP_DEST = 'c:\oracle\admin\ora816\udump';

System altered.
```

You can determine the current values of these parameters by using the `V$PARAMETER` view. For example:

```
SQL> SELECT NAME, VALUE, ISSES_MODIFIABLE, ISSYS_MODIFIABLE
```

```

2 FROM V$PARAMETER

3 WHERE NAME IN ('timed_statistics','user_dump_dest','max_dump_file_size')

4 /

```

NAME	VALUE	ISSES	ISSYS_MOD
-----timed_statistics	TRUE		TRUE
IMMEDIATE user_dump_dest	c:\oracle\admin\ora816\udump	FALSE IMMEDIATE	
max_dump_file_size	100000	TRUE IMMEDIATE	

Step 2: Creating the Trace File

The next step is to turn on tracing for a particular session. Tracing is turned on simply by executing a statement within the session that should be traced or by using some Oracle-supplied packages that typically are used by the DBA to turn on tracing in another session. Turning on tracing causes trace files to be generated in the user dump directory with a file name in a format that typically resembles ORA99999.trc. The "99999" part of the file name is the operating system process identifier (V\$PROCESS.SPID) of the process. These dump files are typically owned by another operating system user. This operating system user will have to grant operating system privileges to the developers who need to read them and process them with TKPROF.

Beware, operating system process identifiers are recycled and used again. If a trace file for that id has already been created, your session will be appended its trace information to the bottom of the existing file which already contains another session's trace statements and statistics. To avoid this problem, the dump directory should be cleaned out regularly.

Tracing from within the Session

To turn on SQL trace use one of these commands in the program or session that needs to be traced.

```

SQL> -- Set SQL Trace on in the current session (2 methods)

SQL>

SQL> ALTER SESSION SET SQL_TRACE = TRUE; -- use FALSE to turn it off

Session altered.

SQL> EXECUTE DBMS_SESSION.SET_SQL_TRACE(SQL_TRACE=>TRUE); -- again use FALSE

PL/SQL procedure successfully completed.

```

Alternatively tracing can be enabled by invoking Oracle Forms with the `-s` option. In Oracle Applications, tracing can be turned on using the `HELP -> TOOLS -> TRACE` menu option to trace database activity executed by the form only. To trace Oracle Reports, a `BEFORE REPORT` trigger can be used to execute one of the above methods. For example, `SRW.DO_SQL('ALTER SESSION SET SQL_TRACE = TRUE')`.

Tracing Another Session

In order to trace a program where the source code is unavailable, the DBA will have to turn on tracing using the `DBMS_SYSTEM` supplied package using the session's serial number and session identifier as soon as the program or session is initiated. This procedure turns out to be a trace; any statements executed before the trace is started will not be included in the trace file.

The following script can be used to help identify and then turn tracing on in another session.

```
SQL> -- Find another sessions information
```

```
SQL>
```

```
SQL> SELECT
```

```

2  S.USERNAME  ORACLE_USER_NAME,
3  S.OSUSER    OPERATING_SYSTEM_USER,
4  S.SID      SESSION_ID,
5  S.SERIAL#   SERIAL#,
6  P.SPID     OS_PROCESS_ID -- TRACE FILE IDENTIFIER
7 FROM
8  V$SESSION S,
9  V$PROCESS P
10 WHERE
11  S.PADDR = P.ADDR
12  AND  S.STATUS = 'ACTIVE'
13  AND  S.USERNAME = '&user'
14 ORDER BY
15  LOGON_TIME
16 /
```

```
Enter value for user: SCOTT
```

```
ORACLE_USER_NAME  OPERATING_SYSTEM_USER      SESSION_ID  SERIAL#  OS_PROCES
```

```
-----
SCOTT             QEDLAB\Administrator          11   3826  104
```

```
SQL>
```

```
SQL> -- Setting another session
```

```
SQL>
```

```
SQL> EXECUTE SYS.DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(SID=>&sid, SERIAL#=>&serialno,
SQL_TRACE=>TRUE)
```

```
Enter value for sid: 11
```

Enter value for serialno: 3826

PL/SQL procedure successfully completed.

If tracing is turned on for many programs at the same time, a particular program's trace file can be identified by including a statement such as `SELECT 'ProgramName' FROM DUAL` in the program itself. This statement along with all of the others will be dumped into the trace file. The trace files can then be searched for the word 'ProgramName'. To determine where the trace file was created, `V$PARAMETER` can be queried for the parameter `user_dump_dest`. As a final step, tracing should be turned off for the session by either issuing the appropriate command or exiting the Oracle session.

Step 3: Creating the TKPROF Report

Now that the trace file has been created it is time to process the trace file into something usable by running TKPROF. The individual trace files are legible and can be edited, but realistically they need to be processed by the TKPROF facility to provide clear results. To get a full list of parameters that can be specified when running TKPROF, issue the command TKPROF without any parameters.

A typical and useful execution of TKPROF would be as follows (where `ORA00104.trc` is the input file and `test2.prf` is the resulting report file):

```
C:\oracle\ADMIN\ora816\udump>tkprof ORA00104.trc test4.prf sys=no sort=exeqry print=10 explain=scott/
tiger
```

```
TKPROF: Release 8.1.6.0.0 - Production on Wed Jul 19 13:37:52 2000
```

```
(c) Copyright 1999 Oracle Corporation. All rights reserved.
```

```
C:\oracle\ADMIN\ora816\udump>dir
```

```
...
```

```
07/19/00 01:34p      1,000,270 ORA00104.TRC
```

```
07/19/00 01:38p      17,564 test4.prf
```

A description of these common parameters is shown below:

TKPROF Parameter	Notes
<code>SYS = NO YES</code>	Suppresses recursive SQL statements from the output. YES is the default, but NO is probably what you want.
<code>SORT = <sort option></code>	Causes the statements to be sorted in descending order by the sort option specified. See table below for sort options allowed.
<code>PRINT = <n></code>	Causes only n statements to be reported (with sort provides the highest n statements).
<code>EXPLAIN = <username/ password></code>	Provides the execution plan for each statement captured. Each step shows the number of rows investigated/resulting. If the user does not have a <code>PLAN_TABLE</code> , TKPROF creates one, uses it, and drops it.

SQL statements are executed by the instance in three phases: parse, execution, and fetch.

- During the Parse phase, the statement syntax is checked, object permissions checks are performed, the statement may be transformed into an equivalent more efficient statement, and the execution plan is determined.
- During the Execution phase, the statement is actually executed. For DML statements the data is actually manipulated during this phase. For queries, the actual rows selected are identified.

- During the Fetch phase, the actual rows are retrieved for the query. For DML, nothing happens during this phase.

Below is a chart detailing twenty-two statistics that can be used to sort the SQL statements in the TKPROF report. The report can also be sorted by the sum of two or more statistics by simply listing the sort options separated by commas. For example: 'SORT=exeqry,fchqry'.

Description	Sort Options / Phase			Report Column Name
	Parse	Execute	Fetch	
Number of times ...	1. PRSCNT	8. EXECNT	16. FCHCNT	Count
CPU time spent in...	2. PRSCPU	9. EXECPU	17. FCHCPU	CPU
Elapsed time spent in...	3. PRSELA	10. EXEELA	18. FCHELA	Elapsed
Number of physical reads from disk during ...	4. PRSDSK	11. EXEDSK	19. FCHDSK	Disk
Number of consistent mode block reads during ...	5. PRSQRY	12. EXEQRY	20. FCHQRY	Query
Number of current mode block reads during ...	6. PRSCU	13. EXECU	21. FCHCU	Current
Number of rows processed during ...		14. EXEROW	22. FCHROW	Rows
Number of library cache misses during ...	7. PRSMIS	15. EXEMIS		Misses

Consistent mode block reads, or the 'Query' statistic, are block access made for queries and sub queries and are subject to read consistency. Current mode block reads, or the 'Current' statistics, are block access made for DML statements and are not subject to read consistency.

Step 4: Understanding the TKPROF Report

TKPROF creates a text file that includes some identification information at the top of the file, detailed statement results in the middle, and a summary of statistics for all statements traced at the bottom of the file.

For each SQL statement, TKPROF lists the statement in the report, provides a chart for each statistic (see sort option chart above), identifies the last user ID who issued the statement, and produces an explain plan type output which shows the number of rows that resulted from each step.

It is possible to trace a program without setting TIMED_STATISTICS to TRUE; however, the CPU time and Elapsed time statistics will be zero. With timing turned on, these statistics have a resolution of one hundredth of a second; therefore, any operation that takes less than one one-hundredth of a second may not be timed accurately. For simple queries or SQL statements that execute quickly, the time statistics should be used with caution.

Below is a portion of a TKPROF file for a single SQL statement.

```

select ename, dname
from emp e, dept d
where e.deptno = d.deptno
call  count  cpu  elapsed  disk  query  current  rows
-----
Parse  1    0.08  0.20    0    0    0    0

```

Execute	1	0.00	0.01	0	0	0	0
Fetch	2	0.01	0.03	2	3	8	14

total	4	0.09	0.24	2	3	8	14

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 32 (SCOTT)

Rows Execution Plan

```

0  SELECT STATEMENT  GOAL: CHOOSE
14 HASH JOIN
4  TABLE ACCESS    GOAL: ANALYZED (FULL) OF 'DEPT'
14 TABLE ACCESS    GOAL: ANALYZED (FULL) OF 'EMP'

```

The execution plan is read inside out and from top to bottom. The first step executed is the topmost leaf in the tree structure. Refer to the Oracle documentation for more information about how to read an execution plan (see explain plan). Also be aware that beginning with release 8i of ORACLE, the definition of 'Rows' in TKPROF's explain plan changed from being the number of rows processed (input to the step) to the number of rows returned (output from the step).

An average hit ratio can also be determined for the statement by simply dividing the physical read count and logical read count for the execution and fetch phase.

$$\text{Hit Ratio} = 1 - (\text{EXEDSK} + \text{FCHDSK}) / (\text{EXEQRY} + \text{FCHQRY} + \text{EXECU} + \text{FCHCU})$$

$$\text{Hit Ratio} = 1 - (0 + 2) / (0 + 3 + 0 + 8) = .82 \text{ -- From the above example.}$$

As always, the higher the hit ratio the faster a statement will usually execute.

Using TKPROF to Locate Problems

The predominate use of SQL Trace and TKPROF is to evaluate the most expensive worst five to ten statements using the PRINT and SORT parameters. Below are some sort options/statistics that can be used to locate or identify common problems.

DISK versus BLOCK read statistics

When comparing two different alternative SQL statements, don't use the disk statistics (EXEDSK, FCHDSK). Running back-to-back statements that go after the same data will usually result in the second statement running faster and with less disk accesses since the desired data blocks may still be resident in buffer pool from the first statements execution. Running the same statements in the opposite order may provide a totally different answer using the disk statistics. Therefore, when comparing alternative statements, the actual buffers accessed (EXEQRY + FCHQRY + EXECU + FCHCU) should normally be used rather than disk accesses or elapsed time etc.

COUNT and Library MISS statistics

The parse and execution count statistics (PRSCNT, EXECNT) can be used to identify unnecessary re-parsing of a statement. Ideally, the statement can be executed numerous times and only parsed once. However, if the same statement is being parsed many times the SHARED_POOL_SIZE may need to be increased.

In addition, if the program is issuing dynamic SQL without using bind variables, the kernel treats these as different statements that have to each be parsed. The application should be rewritten to use bind variables to reduce this unnecessary parsing. Alternatively with Oracle 8i the instance parameter CURSOR_SHARING can be set to FORCE, which will dynamically rewrite the statements to use a system generated bind variable. Beware, however, that setting this parameter can hurt decision support system performance that may rely on the optimizer actually knowing the value so that column histograms can be used.

CPU and ELAPSED statistics

Typically CPU and ELAPSED time statistics (%%%CPU, %%%ELA) are not a common approach to finding performance problems; however, they can be used to identify potential locking problems for DML statements, which are indicated by a high elapsed time versus the CPU time.

ROWS statistics

A high row count at a particular explain plan step versus other steps may indicate that an index with poor cardinality is being used or no index is being used at all. Also it is common for simple statements to be executed thousands of times within a loop that perform a test that returns no rows or the same row over and over again. While these statements execute quickly they can often be removed by using a simple last value variable and a simple conditional test instead of executing the query again. An example would be a loop that a currency conversion is done inside the loop. Probably the currency doesn't change for each repetition of the loop and the actual conversion rate only needs to be checked when it changes.

Other Useful Techniques

Another useful technique is to concatenate together multiple trace files and then process them with TKPROF. This technique can be used to identify the worst statements across numerous sessions, programs or even applications.

In addition, many applications allow for the user to turn on tracing through menu options or batch programs to be traced by setting switches or flags. These techniques avoid the problem of having the programmer edit the program in order to trace and avoid the potential rat race of having to identify the user's session and turning tracing on, potentially missing some of the initial statements.

A Plug for AUTOTRACE

Starting with Oracle 7.3, Oracle introduced a new feature for SQL*PLUS called AUTOTRACE. AUTOTRACE is a simplified tracing mechanism that provides statistical information for the current session similar to using tracing and TKPROF. AUTOTRACE provides a superior execution plan format that clearly shows the parent-child relationship between each step as well as the steps cost and cardinality, and the statements statistics. When using the cost based optimizer, it is useful to explain a statement using AUTOTRACE without actually executing the statement in order to identify the specific object level statistics being used.

Below is an example of using AUTOTRACE.

```
SET AUTOTRACE OFF|ON|TRACEONLY [EXPLAIN] [STATISTICS]
```

```
SQL> set autotrace traceonly
```

```
SQL> select ename, dname
```

```
2 from emp e, dept d
```

```
3 where e.deptno = d.deptno;
```

14 rows selected.

Execution Plan

```
-----
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=3 Card=14 Bytes=252)
1  0  HASH JOIN (Cost=3 Card=14 Bytes=252)
2  1  TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=44)
3  1  TABLE ACCESS (FULL) OF 'EMP' (Cost=1 Card=14 Bytes=98)
```

Statistics

```
-----
277 recursive calls

8 db block gets      (CU – Current)
43 consistent gets   (QRY – Query)
2 physical reads     (DSK – Disk)

0 redo size

1112 bytes sent via SQL*Net to client
424 bytes received via SQL*Net from client

2 SQL*Net roundtrips to/from client

6 sorts (memory)
0 sorts (disk)

14 rows processed
```

Adding Events to a Trace File

Session level events can be set to cause the database to put more information into the trace file. This information is ignored by TKPROF, so the actual trace file will have to be examined. Two particular events are useful in tuning applications; however they should only be used one at a time.

Event 10046 - Bind Variables and Waits

Sometimes the developer might need to know the actual bind variable values used in a statement in order to perform some test of alternate equivalent statements, use of hints or the creation of indexes. In order to see these values, some undocumented features of Oracle will have to be used. Specifically event 10046 will have to be set. This event will put the actual bind values in the trace file.

Most events have many levels, only some being meaningful to the world outside of Oracle Corp. For event 10046, level 1 is the default, which provides the same information as SQL Trace that has been covered in this

paper until this point. Level 4 contains bind variable values; level 8 contains wait events; and level 12 contains both.

```
SQL> -- Setting Event 10046, in Current Session
```

```
SQL>
```

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 12';
```

Session altered.

```
SQL> declare
```

```
2 x VARCHAR2(100);
```

```
3 z NUMBER;
```

```
4 begin
```

```
5 z := 7369;
```

```
6 select job
```

```
7 into x from emp where empno = z;
```

```
8 end;
```

```
9 .
```

```
SQL> /
```

PL/SQL procedure successfully completed.

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context off';
```

Session altered.

The following portion of a trace dump file shows the actual value of the bind variable being used. The bind value can be identified by looking for "value=" at the end of a bind line for the cursor (identified by #<cursor number>). The PARSING IN CURSOR keywords can be used to locate the cursor number of the statement of interest. The actual SQL statement follows these keywords.

```
PARSING IN CURSOR #2 len=40 dep=1 uid=32 oct=3 lid=32 tim=1609504 hv=517058596 ad='3cf7298'
```

```
SELECT JOB FROM EMP WHERE EMPNO = :b1
```

```
END OF STMT
```

```
PARSE #2:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1609506
```

```
BINDS #2:
```

```
bind 0: dty=2 mxl=22(21) mal=00 scl=00 pre=00 oacflg=03 oacfl2=1 size=24 offset=0
```

```
 bfp=095c4ebc bln=22 avl=03 flg=05
```

```
value=7369
```

The following portion of a trace dump file shows a latch free wait on cursor 52. To analyze wait information contained in the trace file (identified by WAIT#<cursor_number>) search the Oracle Doc for "Wait Events" which will provide more information for the Wait Event occurring and how to decipher the values of p1, p2, and p3. In this case, P1 is the address of the latch, P2 is the latch number in V\$LATCHNAME, and P3 is the number of tries. In this case latch number 60 turned out to be a library cache wait.

```
...
WAIT #52: nam='latch free' ela= 1 p1=-1553004852 p2=60 p3=0
...
```

It is also possible to set an event in another session using the SET_EV procedure.

```
SQL> -- Setting Event 10046, in Another Session
```

```
SQL>
```

```
SQL> EXECUTE SYS.DBMS_SYSTEM.SET_EV(SI=>&sid, SE=>&serialno, EV=>&event, LE=>&level, NM=>")
```

```
Enter value for sid: 11
```

```
Enter value for serialno: 3826
```

```
Enter value for event: 10046
```

```
Enter value for level: 12
```

```
PL/SQL procedure successfully completed.
```

```
Event 10053 - Cost Optimizer Detail
```

Event 10053 dumps information about what the Cost Based Optimizer did to create an execution plan for the query including the cost of each step of each plan. Not only will the trace file contain different execution paths that the Optimizer evaluated, it will also contain all of the initialization parameters used by the Optimizer as well as all of the table, column, and index statistics used as well. This event must be set before the statement is parsed, so if it is already in the shared pool, modify the statement slightly or as a last resort have the DBA flush the shared pool. Use level 1 which is the default.

```
SQL> -- Set Event 10053 for Cost information.
```

```
SQL>
```

```
SQL> ALTER SESSION SET EVENTS '10053 trace name context forever';
```

```
Session altered.
```

```
SQL>
```

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL; -- Clearing Parsed Statements!
```

```
System altered.
```

```
SQL>
```

```
SQL> select ename, dname from emp, dept
      2 where emp.deptno = dept.deptno and empno = 7369;
```

```
ENAME    DNAME
-----
```

```
SMITH    RESEARCH
```

Below is only a portion of the information that event 10053 dumps to the trace file.

```
--- EXAMPLE OF PARAMETER INFORMATION DUMPED

OPTIMIZER_FEATURES_ENABLE = 8.1.6

OPTIMIZER_MODE/GOAL = Choose

...

--- EXAMPLE OF OBJECT LEVEL STATISTICS DUMPED

Table stats  Table: EMP  Alias: EMP

TOTAL :: CDN: 14 NBLKS: 1 TABLE_SCAN_CST: 1 AVG_ROW_LEN: 40

-- Index stats

INDEX#: 21956 COL#: 1

TOTAL :: LVLS: 0 #LB: 1 #DK: 14 LB/K: 1 DB/K: 1 CLUF: 1

...

--- EXAMPLE OF ACCESS PATHS CONSIDERED BY COST BASED OPTIMIZER

Join order[1]: EMP [EMP] DEPT [DEPT]

Now joining: DEPT [DEPT] *****

NL Join

Outer table: cost: 1 cdn: 1 rcz: 10 resp: 1

Inner table: DEPT

Access path: tsc Resc: 1

Join resc: 2 Resp: 2

Access path: index (unique)

INDEX#: 21954 TABLE: DEPT

CST: 1 IXSEL: 2.5000e-001 TBSEL: 2.5000e-001
```

```

Join resc: 2  resp:2

Column:  DEPTNO Col#: 8   Table: EMP  Alias: EMP

NDV: 3     NULLS: 0     DENS: 3.3333e-001 LO: 10 HI: 30

Column:  DEPTNO Col#: 1   Table: DEPT  Alias: DEPT

NDV: 4     NULLS: 0     DENS: 2.5000e-001 LO: 10 HI: 40

Access path: index (eq-unique)

INDEX#: 21954 TABLE: DEPT

CST: 1 IXSEL: 0.0000e+000 TBSEL: 0.0000e+000

Join resc: 2  resp:2

Join cardinality: 1 = outer (1) * inner (4) * sel (2.5000e-001) [flag=0]

Best NL cost: 2  resp: 2

...

```

Error Events

An event can be set that will create a trace file whenever a specific Oracle error is encountered. For packaged applications especially, it can help the implementation team determine what table is causing a 942 error (table or view does not exist) or what SQL statement is giving a 1403 error (no data found). Use level three.

```

SQL> -- Set Error Level Trace for Event 942 for current session

SQL>

SQL> ALTER SESSION SET EVENTS '942 trace name errorstack level 3';

Session altered.

SQL> -- Set Error Level Trace for Event 942 OFF for current session

SQL>

SQL> ALTER SESSION SET EVENTS '942 trace name context off';

Session altered.

SQL> -- Set Error Level Trace for Event 942 for another session

SQL>

SQL> EXECUTE SYS.DBMS_SYSTEM.SET_EV(SI=>&sid, SE=>&serialno, EV=>&event, LE=>3,
NM=>'ERRORSTACK')

Enter value for sid: 11

```

Enter value for serialno: 3826

Enter value for event: 942

PL/SQL procedure successfully completed.

It is also possible to turn tracing on for the entire database. This decision should be made with great caution and should be used only for a very short period of time. To turn on SQL Trace for all sessions of the database, set the initialization parameter `SQL_TRACE=TRUE` in the `init.ora` file and bouncing the database. It is also possible to set database wide events by issuing the `ALTER SYSTEM SET EVENTS` command. Be sure to turn these back off.

Conclusion

Unfortunately, tracing and TKPROF remain a globally overlooked opportunity by new and experienced developers alike. A reviewed trace file should be part of each developer's test plan. By using these free facilities, you can catch common mistakes early in development. In a well-organized environment, this step would require minimal additional effort by the application developer or tester.