

Advanced Oracle Troubleshooting – Live Session

Randolf Geist
Freelance Consultant
Mannheim, Germany

Schlüsselworte:

Advanced Oracle Troubleshooting, Live Session, Performance

Einleitung:

In this session we'll walk through some common troubleshooting scenarios, starting with the usual inefficient execution plan issues, showing that the Oracle Wait Interface and Session Statistics are usually well suited to spot anomalies and determine where excess work happens.

We'll then continue with some less common scenarios where the wait Interface or session statistics tend to be useless and demonstrate some advanced troubleshooting techniques that still allow a systematic approach under such circumstances.

This is a live demonstration session along with a couple of slides that describe the techniques applied.

The following will be covered during this session:

- SQL Monitoring
- Extended SQL trace resp. Oracle Wait Interface (I/O, contention, locks)
- Session Statistics
- Active Session History / S-ASH
- Latch/Mutex Activity
- Detailed Consistent gets analysis (excess consistent gets)
- Oracle heapdump analysis
- Operating System level process tracing

Basic Troubleshooting Techniques

Before we get too excited about Advanced Troubleshooting it is important to remember that most performance issues can be identified using basic troubleshooting techniques.

So what are your most important basic troubleshooting tools?

- Systematic approach
 - Always focus on the most important and affected business process; if possible avoid troubleshooting by looking at the "system" from an aggregated view
- Once you have identified these processes the most important things to check first are:

- Session Statistics and Wait Interface => Best tool: Tanel Poder's "snapper". Note that snapper covers more than just statistics and wait interface information
- If you have identified SQLs or sessions that are taking longer than expected, use proven technology: Extended SQL trace along with DBMS_XPLAN.DISPLAY_CURSOR / STATISTICS_LEVEL = ALL (10g and later)
 - These two tools are complementary: Both tell you where most of the time is spent. Extended SQL trace with waits enabled shows additionally the wait events whereas DBMS_XPLAN.DISPLAY_CURSOR shows the estimated cardinalities together with the actual ones. Furthermore it shows the estimated memory requirements along with the actual memory used and the amount of TEMP space used when spilling to disk
- Bad execution plans are often caused by incorrect cardinality estimates - check the estimated and actual cardinalities reported by DBMS_XPLAN.DISPLAY_CURSOR
- In 11g SQL Real-Time Monitoring has been added - however this requires a tuning pack & diagnostics license
 - V\$SQL_MONITOR
 - V\$SQL_PLAN_MONITOR
 - DBMS_SQLTUNE.REPORT_SQL_MONITOR
 - Combines information from V\$SQL_PLAN_MONITOR, V\$SESSION_LONGOPS and V\$ACTIVE_SESSION_HISTORY
 - Loads of parameters and options - worth to study the details
- Pimp up your basic tools:
 - Use Adrian Billington's XPLAN wrapper for DBMS_XPLAN.DISPLAY* functions to add execution order and parent operation info
 - Alberto Dell'Era's XPLAN / XTRACE tools (<http://www.adellera.it/>)
 - Use alternative trace file analyzers in addition to TKPROF:
 - Oracle's own Trace Analyzer (TRCA / TRCANLZR) See MOS Note 224270.1, and see also extended Explain Plan 215187.1 SQLT (SQLTXPLAIN) Both tools require an installation and are a bit cumbersome to use, however they provide a wealth of detailed information
 - Christian Antognini's TVD\$XTAT (<http://antognini.ch/top>)
 - OraSRP (<http://www.oradba.ru/orasrp>)
 - Support your developers by using Method R's MR Trace SQLDeveloper plug-in - allows to automatically fetching trace files from the server or write your own little helper tool that transports trace files to the client automatically

Live Demo

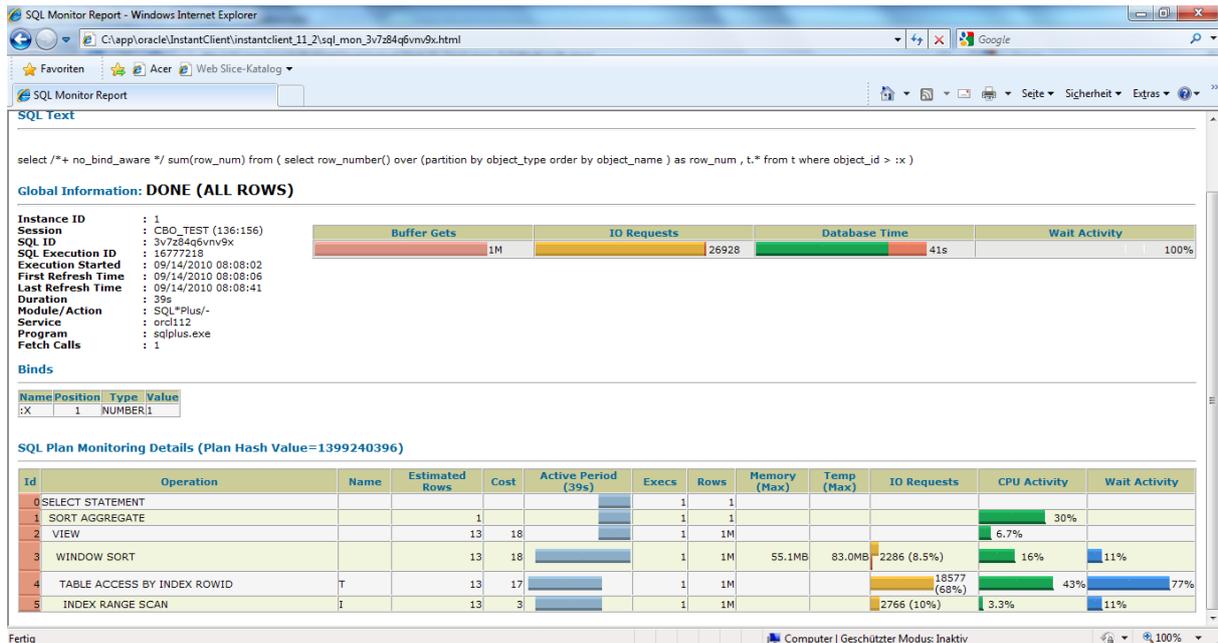


Image 1: Real-Time SQL Monitoring

Active Session History (ASH)

Another important tool for troubleshooting is the so called “Active Session History”. This feature has officially been added to Oracle in version 10g, but requires an additional license (diagnostic & tuning pack).

Sometimes you get a call "My process was slow 10 minutes ago, but it's completed now"

- Oracle 10g has added Active Session History (ASH)
 - Interfaces: V\$ACTIVE_SESSION_HISTORY and DBA_HIST_ACTIVE_SESS_HISTORY
 - Standard report available: RDBMS/ADMIN/ASHRPT.SQL
 - Requires additional license diagnostic & tuning pack
- The good news is that Active Session History is not rocket science, in principle it is about sampling V\$SESSION (or its underlying structures). Therefore you don't need necessarily an additional license, but can do it yourself or use instead Kyle Hailey's S-ASH scripts for an "Active Session History" without ASH license. See <http://www.ashmasters.com> or the OraSASH homepage on Sourceforge.net.
- When using connection pooling good instrumentation using client identifiers is important for effective troubleshooting

Live Demo

ASH Report For ORCL112/orcl112
(1 Report Target Specified)

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORCL112	3288365572	orcl112	1	11.2.0.1.0	NO	ACER-ASPRE

CPU#	SGA Size	Buffer Cache	Shared Pool	ASH Buffer Size
2	311M (100%)	144M (46.4%)	128M (41.2%)	4.0M (1.3%)

	Sample Time	Data Source
Analysis Begin Time:	14-Sep-10 08:04:25	V\$ACTIVE_SESSION_HISTORY
Analysis End Time:	14-Sep-10 08:09:25	V\$ACTIVE_SESSION_HISTORY
Elapsed Time:	5.0 (mins)	
Sample Count:	76	
Average Active Sessions:	0.25	
Avg Active Session per CPU:	0.13	
Report Target:	SQL_ID like '3v7z84q@vrv9x'	51% of total database activity

ASH Report

- Top Events
- Load Profile
- Top SQL
- Top PL/SQL
- Top Java
- Top Call Types
- Top Sessions
- Top Objects/Files/Latches
- Activity Over Time

Fertig

Image 2: ASH Report

Latch/Mutex Activity

- If multiple processes compete for certain resources you might see so called "latch contention"
- Latches and Mutexes protect and serialize access to in-memory structures of the SGA
- Most commonly "latch cache buffer chain" contention when multiple processes excessively attempt to access buffers protected by the same latch buffer chain
- Another common issue is library cache contention due to excessive hard parsing activity
- Note that latch contention usually is a symptom rather than a cause
- Since spinning on latches burns CPU, excessive buffer gets can significantly increase CPU load
- On the other hand if you overload your CPUs then latch contention can be exaggerated by this CPU overload since the latches are held for too long and therefore contention of latches will be observed
- So latch contention can cause CPU load but can also be a symptom of CPU overload
- In general there is not much you can do about this except for changing the application logic
- In case of excessive buffer gets check if a bad statement execution plan causes this (mostly plans including NESTED LOOP operations), or in case of "hot blocks" determine the cause, for instance a sequence-based insert into an indexed column might cause such issues - then a remedy might be to modify the index to reverse or global hash partitioned
- Details about latch/mutex activity can be obtained from dynamic performance views and internal X\$ fixed tables
- Tanel Poder's tool set comes handy: LATCHPROF / LATCHPROFX (only as SYS or when X\$ fixed views are accessible) and MUTEXPROF
- More information: Tanel Poder's presentation and seminars about Latch / Mutex troubleshooting

Live Demo

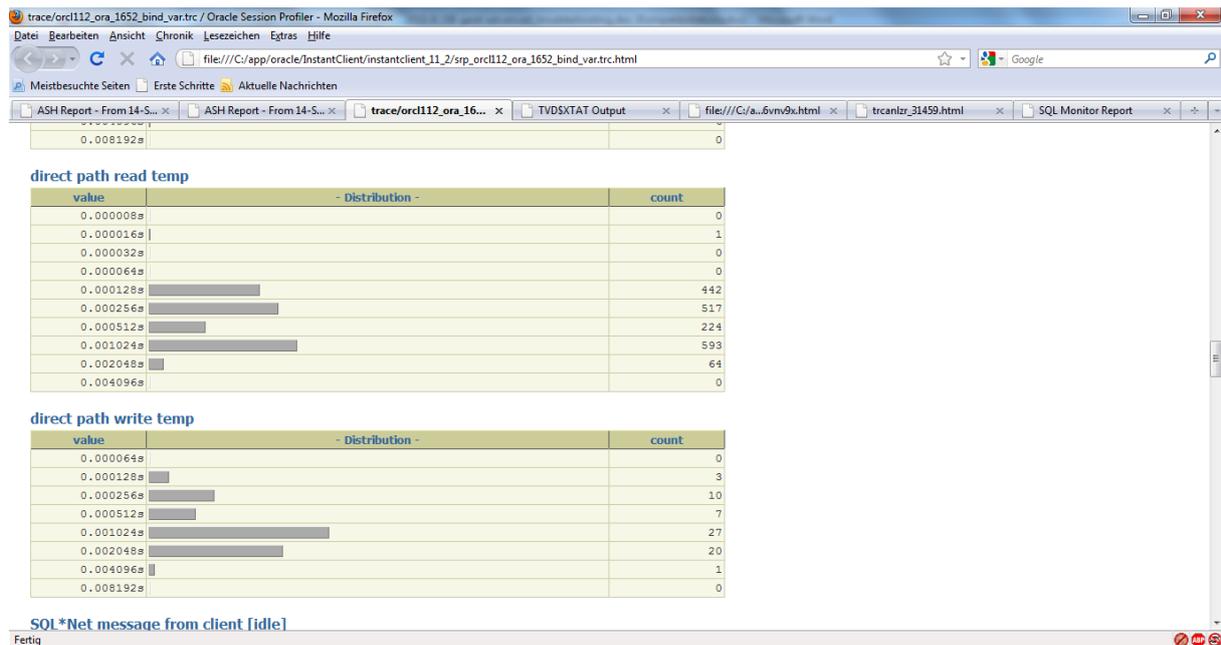


Image 3: Extended SQL Trace file analysis using OraSRP

Detailed Consistent gets analysis (excess consistent gets)

Tracing and DBMS_XPLAN.DISPLAY_CURSOR tell you the number of consistent gets performed, but don't tell you the reason why they have been performed. If you suspect that the number of consistent gets is higher than expected, it would be interesting to know the "reason" why the consistent get was performed.

- Excessive consistent gets can be caused by different reasons
 - The most obvious reason: Reconstructing older versions of a block by applying undo
 - Bugs: many ASSM related problems in past versions
- Up to version 10g internal X\$ tables can be used to obtain information about the reason, for 11g the internal structure has changed the contents are not updated any longer
- If you connect with a user capable of accessing X\$ tables, snapper will show you this as well
- Jonathan Lewis provides a nice script which takes a snapshot of this structure and outputs the delta afterwards, see http://www.jlcomp.demon.co.uk/buffer_usage.html
- Note that this is global information and not limited to particular sessions, so on-going activity may influence the measurement
- Alternatively use event 10200/10202 to trace consistent gets

Live Demo

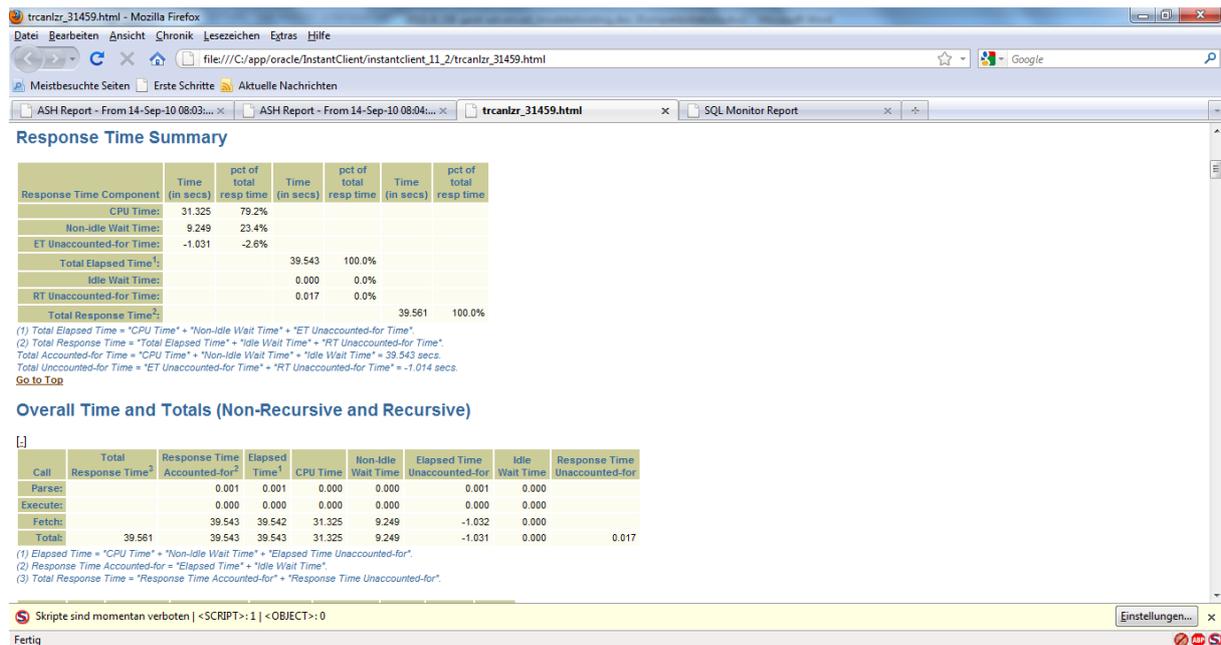


Image 4: Oracle Trace Analyzer output

When basic troubleshooting techniques are not sufficient

Now we'll come to the point where above tools will not allow a detailed diagnosis any longer.

You hit the limits for example if you have a case that:

- Consistently sits on CPU, but doesn't show anything in the session statistics, so you don't have a clue what the process is actually doing
- Performs some uninstrumented code section and therefore doesn't update the wait interface and/or the session statistics accordingly

Examples:

- PL/SQL processing
- Parsing bugs
- Instrumentation bugs: External table access up to 11g

In these cases a different approach is required if we need more details

Trace the process on OS level using:

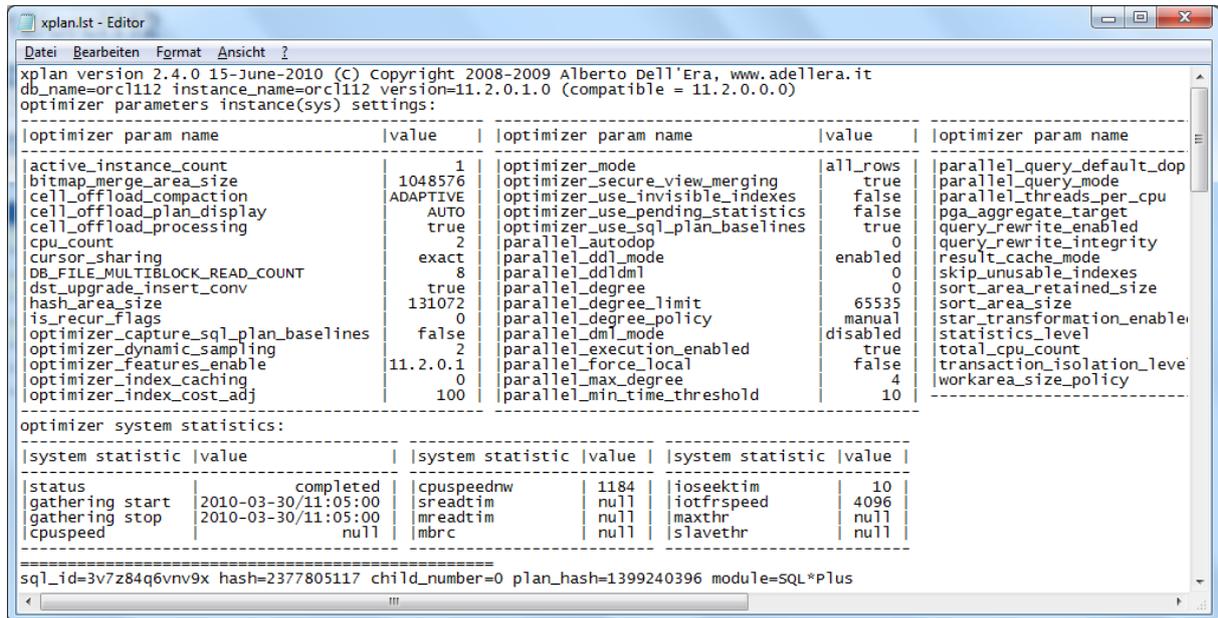
- Tanel Poder's OSStackProf
- DTrace (Solaris)
- procstack (AIX)
- pstack / gdb backtrace (Linux)
- In future may be: probevue (>= AIX 6.1), systemtap (Linux)

Sometimes it is also useful to find out the system calls performed by the process using:

- truss (Linux, AIX)
- strace
- ptrace

Use Metalink / MOS document 175982.1 to get an idea about the purpose of the functions traced

Live Demo



The screenshot shows the xplan tool interface with the following content:

```
xplan version 2.4.0 15-June-2010 (C) Copyright 2008-2009 Alberto Dell'Era, www.adellera.it
db_name=orcl112 instance_name=orcl112 version=11.2.0.1.0 (compatible = 11.2.0.0.0)
optimizer parameters instance(sys) settings:
```

optimizer param name	value	optimizer param name	value	optimizer param name
active_instance_count	1	optimizer_mode	all_rows	parallel_query_default_dop
bitmap_merge_area_size	1048576	optimizer_secure_view_merging	true	parallel_query_mode
cell_offload_compaction	ADAPTIVE	optimizer_use_invisible_indexes	false	parallel_threads_per_cpu
cell_offload_plan_display	AUTO	optimizer_use_pending_statistics	false	pga_aggregate_target
cell_offload_processing	true	optimizer_use_sql_plan_baselines	true	query_rewrite_enabled
cpu_count	2	parallel_autodop	0	query_rewrite_integrity
cursor_sharing	exact	parallel_ddl_mode	enabled	result_cache_mode
DB_FILE_MULTIBLOCK_READ_COUNT	8	parallel_ddlml	0	skip_unusable_indexes
dst_upgrade_insert_conv	true	parallel_degree	0	sort_area_retained_size
hash_area_size	131072	parallel_degree_limit	65535	sort_area_size
is_recur_flags	0	parallel_degree_policy	manual	star_transformation_enable
optimizer_capture_sql_plan_baselines	false	parallel_dml_mode	disabled	statistics_level
optimizer_dynamic_sampling	2	parallel_execution_enabled	true	total_cpu_count
optimizer_features_enable	11.2.0.1	parallel_force_local	false	transaction_isolation_level
optimizer_index_caching	0	parallel_max_degree	4	workarea_size_policy
optimizer_index_cost_adj	100	parallel_min_time_threshold	10	

```
optimizer system statistics:
```

system statistic	value	system statistic	value	system statistic	value
status	completed	cpuspeednw	1184	iousektim	10
gathering start	2010-03-30/11:05:00	sreadtim	null	iotfrspeed	4096
gathering stop	2010-03-30/11:05:00	mreadtim	null	maxthr	null
cpuspeed	null	mbrc	null	slavethr	null

```
sql_id=3v7z84q6vvnv9x hash=2377805117 child_number=0 plan_hash=1399240396 module=SQL*Plus
```

Image 5: Alberto Dell'Era's XPLAN tool

OS Explain Plan

The same technique can be used to obtain information about a currently executing execution plan, since the row sources used to execute a SQL statement are represented by corresponding C functions in the Oracle code.

Live Demo

```

C:\app\oracle\instantclient\instantclient_11_2\sqlplus.exe

MOATS: The Mother Of All Tuning Scripts v1.0 by Tanel Poder & Adrian Billington
http://www.e2sn.com & http://www.oracle-developer.net

-----
+ INSTANCE SUMMARY
-----
| Instance: orcl112 | Execs/s: 6.2 | sParse/s: 1.5 | LIOs/s: 944.4 | Read MB/s: 0.0 |
| Cur Time: 14-Sep 08:26:32 | Calls/s: 0.2 | hParse/s: 0.0 | PhyRD/s: 1.2 | Write MB/s: 0.0 |
| History: 0h 0m 37s | Commits/s: 0.0 | cHits/s: 6.0 | PhyWR/s: 2.1 | Redo MB/s: 0.0 |
-----

+ TOP SQL_ID (child#) -----+ TOP SESSIONS -----+ TOP WAITS -----+ WAIT CLASS --+
| 91% | 0199phd7g6p15 (0) | 12,136 | | 91% | ON CPU | | ON CPU |
| 9% | (0) | | | 5% | os thread startup | | Concurrency |
| | | | | 5% | control file sequential res | | System I/O |
-----

+ TOP SQL_ID -----+ PLAN_HASH_VALUE -----+ SQL TEXT -----+
| 0199phd7g6p15 | 2122880181 | select /*+ use_nl(t_i, t_o) index(t_o (id)) use_nl(iter) no_nl_batch |
| | | | ing(t_o) */ | max(t_o.vc_small) from | t_i | t_o |
-----

```

Image 6: Tanel Poder's MOATS tool

Other advanced techniques – heapdumps, errorstack dumps, systemstate dumps etc.

If you can't tell from the call stack what is going on, sometimes it might be helpful to analyze the memory using heapdumps, in particular if you see suspicious activity in the session statistics (for example, ever increasing PGA / SGA memory consumption)

Live Demo

```

C:\Windows\system32\cmd.exe
-1.          ENQG, JD - Job Queue Date                , 1, 5,
-1.          ENQG, JS - Job Scheduler                , 2, 1,
-1.          LATG, enqueue hash chains               , 18, 9,
-1.          LATG, cache buffers lru chain           , 26, 13,
-1.          LATG, cache buffers chains             , 3458, 1.73k,
-1.          LATG, tablespace key chain              , 3, 1.5,
-1.          LATG, mapped buffers lru chain          , 3, 1.5,
23. CBO_TEST STAT, session logical reads           , 763, 381.5,
23. CBO_TEST STAT, consistent gets                  , 763, 381.5,
23. CBO_TEST STAT, consistent gets from cache       , 763, 381.5,
23. CBO_TEST STAT, consistent gets from cache (fastpath) , 763, 381.5,
23. CBO_TEST STAT, calls to kmgcs                  , 4, 2,
23. CBO_TEST STAT, no work - consistent read gets   , 791, 395.5,
23. CBO_TEST STAT, table scans (short tables)       , 1, .5,
23. CBO_TEST STAT, table scan rows gotten           , 564324, 282.16k,
23. CBO_TEST STAT, table scan blocks gotten        , 772, 386,
23. CBO_TEST STAT, table fetch by rowid            , 564429, 282.21k,
23. CBO_TEST STAT, index scans kdixs1              , 568914, 284.46k,
23. CBO_TEST STAT, buffer is pinned count          , 1131102, 565.55k,
23. CBO_TEST TIME, DB CPU                           , 639504, 319.8ms, 32.0%, |0000 |
23. CBO_TEST TIME, sql execute elapsed time        , 1022688, 511.34ms, 51.1%, |000000 |
23. CBO_TEST TIME, DB time                           , 1022688, 511.34ms, 51.1%, |000000 |
134. CBO_TEST STAT, user calls                       , 1, .5,
134. CBO_TEST STAT, pinned cursors current           , -1, -.5,
134. CBO_TEST STAT, session logical reads           , 1058, 529,
134. CBO_TEST STAT, CPU used when call started      , 2156, 1.08k,
134. CBO_TEST STAT, CPU used by this session        , 2156, 1.08k,
134. CBO_TEST STAT, DB time                           , 2330, 1.17k,
134. CBO_TEST STAT, non-idle wait count             , 1, .5,
134. CBO_TEST STAT, consistent gets                  , 1058, 529,
134. CBO_TEST STAT, consistent gets from cache       , 1058, 529,
134. CBO_TEST STAT, consistent gets from cache (fastpath) , 1058, 529,
134. CBO_TEST STAT, calls to kmgcs                  , 1, .5,
134. CBO_TEST STAT, no work - consistent read gets   , 1053, 526.5,
134. CBO_TEST STAT, table scan rows gotten           , 744150, 372.08k,
134. CBO_TEST STAT, table scan blocks gotten        , 1018, 509,
134. CBO_TEST STAT, table fetch by rowid            , 744392, 372.2k,
134. CBO_TEST STAT, index scans kdixs1              , 741812, 370.91k,
134. CBO_TEST STAT, buffer is pinned count          , 1462416, 731.21k,
134. CBO_TEST STAT, workarea executions - optimal   , 1, .5,
134. CBO_TEST STAT, bytes sent via SQL*Net to client , 452, 226,
134. CBO_TEST STAT, bytes received via SQL*Net from client , 11, 5.5,
134. CBO_TEST TIME, SQL*Net roundtrips to/from client , 2, 1,
134. CBO_TEST TIME, DB CPU                           , 1185608, 592.8ms, 59.3%, |000000 |
134. CBO_TEST TIME, sql execute elapsed time        , 1229241, 614.62ms, 61.5%, |000000 |
134. CBO_TEST TIME, DB time                           , 1229303, 614.65ms, 61.5%, |000000 |
134. CBO_TEST WAIT, SQL*Net message to client       , 3, 1.5us, .0%, |
134. CBO_TEST WAIT, SQL*Net message from client     , 1000292, 500.15ms, 50.0%, |000000 |
-- End of Stats snap 10, end=2010-09-14 08:29:19, seconds=2

```

Image 7: Tanel Poder's snapper tool

Kontaktadresse:

Randolf Geist
 Freelance Consultant
 Ersteriner Straße 15
 D-68229 Mannheim

Telefon: +49 (0) 170-758 1171
 E-Mail info@sqltools-plusplus.org
 Internet: http://oracle-randolf.blogspot.com