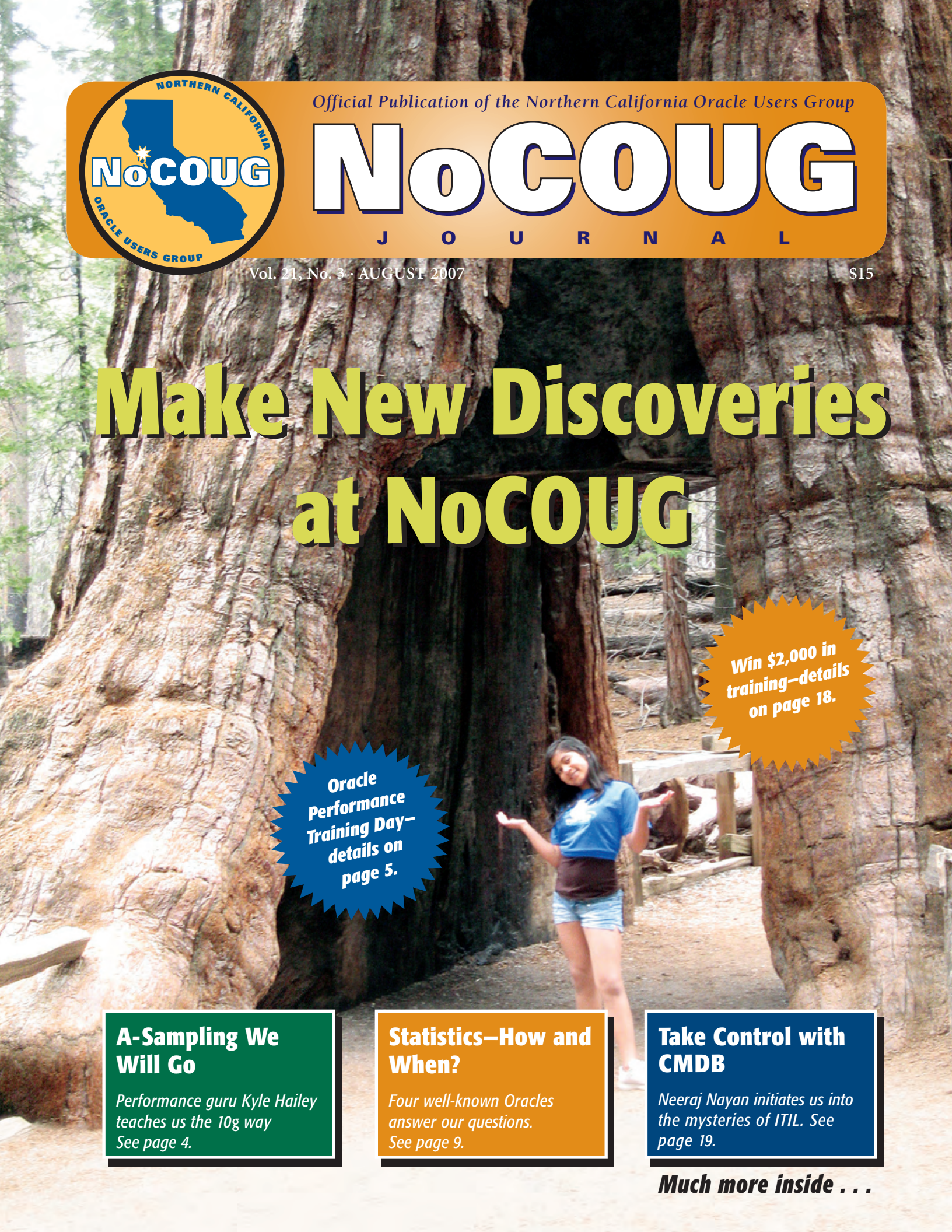
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 21, No. 3 · AUGUST 2007 $15

Make New Discoveries at NoCOUG



Win $2,000 in training—details on page 18.

Oracle Performance Training Day—details on page 5.

A-Sampling We Will Go

Performance guru Kyle Hailey teaches us the 10g way
See page 4.

Statistics—How and When?

Four well-known Oracles answer our questions.
See page 9.

Take Control with CMDB

Neeraj Nayan initiates us into the mysteries of ITIL.
See page 19.

Much more inside . . .

Get Involved!

Three years ago, on August 19, 2004, I went to my very first NoCOUG conference—the summer conference at Chevron in San Ramon—and introduced myself to then president, Roger Schrag, and vice president, Darrin Swan. The very next day, I received the following message from then editor, Lisa Loper.

"Hello Ignatius, I am the editor for the NoCOUG Journal*, and Roger Schrag and Darrin Swan gave me your card. They said you were interested in writing about a deployment of 10*g*. We really enjoy real-life stories in the* Journal*. What did you have in mind? Lisa"*

I wrote my very first article for the next issue of the Journal and never stopped writing! NoCOUG may have benefited from my decision to get involved but I benefited even more. My writing skills slowly improved and I have just been invited by Apress to write a book on Oracle 11*g*. *Beginning Oracle 11*g *Database Administration* should be on bookshelves by this time next year.

Thanks to Ted Syrett, Danny Chow, and Sumit Aneja for volunteering to be our technical reviewers, and thanks to Manoj Thomas for volunteering to be our book reviewer. Send a message to our president, Lisa Loper (**lloper@dbspecialists.com**) if you too would like to get more involved with NoCOUG in any way.

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

NoCOUG BOARD

Publication Notices and Submission Format

The *NoCOUG Journal* is published four times a year by the Northern California Oracle Users Group (NoCOUG) approximately two weeks prior to the quarterly educational conferences.

Please send your questions, feedback, and submissions to the *NoCOUG Journal* editor at **journal@nocoug.org**.

The submission deadline for the upcoming November 2007 issue is August 31, 2007. Article submissions should be made in Microsoft Word format via email.

ADVERTISING RATES

The *NoCOUG Journal* is published quarterly.

Size	Per Issue	Per Year
Quarter Page	$125	$400
Half Page	$250	$800
Full Page	$500	$1,600
Inside Cover	$750	$2,400

Personnel recruitment ads are not accepted.

journal@nocoug.org

Rejuvenate!

by Lisa Loper

"Eight years." "Three years." "*Fifteen* years." "Five years." "Seven years." At our last NoCOUG Board meeting, we took turns introducing ourselves to a guest at our meeting. As each board member gave their name, volunteer role, and how long they've volunteered with NoCOUG, it struck me that we have more longevity within our volunteer organization than many companies have with their employees. We must be doing something right!

We are a diverse group of people who work together to lead NoCOUG in its goals of being an independent volunteer organization dedicated to the education and representation of the users of Oracle Corporation's database and tools software. Along the way, we laugh, network, get into impassioned debates, bring new ideas to the table, and deal with a variety of challenges as a team. With over 450 members, quarterly conferences, training days, and an award-winning technical journal, we have been very successful.

But, this is no time to rest on our laurels. As the great German polymath von Goethe said, "*We must always change, renew, rejuvenate ourselves; otherwise we harden.*" For any organization to remain successful, it needs an ongoing influx of new growth, new ideas, and new enthusiasm. And, like any other organization, NoCOUG needs this, too. We have an amazing group of dedicated NoCOUG board members who volunteer their time and accomplish a lot throughout the year. At the same time, we would love to hear your new ideas, have your volunteer help, and listen to your thoughts about how we can better serve our members.

Lisa Loper

Maybe you've been saying it's time for you to do something new, take on new challenges, meet new people, and rejuvenate yourself? Then, contact me about getting more involved with NoCOUG. I can talk with you about various volunteer opportunities (big or small), invite you to a future board meeting, and help you get more involved. I can be reached at (415) 344-0500, ext. 42, or **lloper@dbspecialists.com**. I look forward to hearing from you.

P.S. Don't miss a couple of other ways to get rejuvenated in August. August 16: *NoCOUG Summer Conference hosted by Chevron in San Ramon* (**www.nocoug.org/next.html**). August 17: *Oracle Performance Training Day with Kyle Hailey in Pleasanton* (**www.nocoug.org/TrainingDay/Flyer.pdf**). ▲

(DIANE LEE)

*Back row, left to right: Jen Hong—Secretary and Treasurer, Lisa Loper—President, Roger Schrag—Director of Conference Programming, Eric Hutchinson—Webmaster, Randy Samberg—Track Leader, Hamid Minoui—Training Day Coordinator, Naren Nagtode—Director of Marketing, Joel Rosingana—Director of Membership, Darrin Swan—Vice President; Front row, left to right: Iggy Fernandez—*Journal *Editor, Nora Rosingana—NoCOUG Staff Member. Not pictured: Hanan Hit—Track Leader, Diane Lee—Vendor Coordinator and IOUG Representative.*

A-Sampling We Will Go

by Kyle Hailey

Oracle wait events took years to become widely known for their power in performance tuning. With this article you'll be able to beat the curve by harnessing the power the new Oracle 10*g* concept of Active Session History (ASH). ASH represents a revolution in performance tuning as radical as the introduction of wait events.

Wait events revolutionized tuning way back in Oracle 7 by identifying performance bottlenecks in the database and explicitly quantifying their impact. Still lacking in the wait events interface was the ability to identify the causes of the problems. The solution has come with ASH, which clearly identifies the SQL and sessions at the root of wait bottlenecks.

Ironically, ASH has been technically feasible for a savvy DBA since Oracle 7, but it took Oracle 10*g* for anyone to put this cutting-edge technology to work, and in no small way, by actually implanting it in the Oracle kernel and basing Oracle 10*g* performance tools on ASH.

ASH is founded upon a data-gathering technique called "Sampling." Sampling is a method of collecting data by taking snapshots every second. The snapshots record the state of every active session in the database. For every active session, Oracle records the SQL being executed as well as the state of the session. A session state can be running on the CPU or waiting on any of several hundred wait events. With this simple amount of information, the session + state + sql, we can quickly identify SQL that is CPU intensive, IO intensive, or involved in any one of a number of the possible resource contention issues that can arise in an Oracle database.

Looking at the situation more concretely, when there is a slowdown on the database we look at our trusted STATSPACK report for the period of the slowdown. The first step in analyzing the STATSPACK report is to look at the Top 5 Timed Events section.

The top five timed wait events will tell us if any wait event has crept up to cause a bottleneck. If we do find a wait event bottleneck, we will need to know who or what is causing the problem in order to solve it. For example, if there is a CPU bottleneck, we need to know what SQL statement is hogging the CPU. If there is an I/O bottleneck, we need to know what SQL statement is stuck on I/O and needs tuning. If there is a complex situation like a buffer busy wait or latch contention, we need to know which sessions were involved, what the wait event arguments were, and what SQL they were executing. STATSPACK fails to give us the necessary detailed information, but ASH provides it.

Real-Life Example

Let's look at a real-life example on a 9*i* database. This example is probably the hardest wait bottleneck to solve, so if you can follow this one, it's downhill from there. This example is one of the more complex tuning problems; thus the analysis requires a deep understanding of Oracle wait events.

Imagine a situation in which users call into the help desk complaining that the application has slowed down. The help desk determines that it's not the application and calls me, the DBA, to fix it. To solve the situation I run the STATSPACK reports over the last hour and look at "Top 5 Timed Events"

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                      % Total
Event                          Waits    Time (s)    Call Time
------------------------------ --------- ---------- ----------
buffer busy waits              2,748          250       78.72
CPU time                                       32       10.16
free buffer waits              1,588           15        4.63
write complete waits              10            8        2.51
log buffer space                 306            5        1.51
```

Sure enough there is a bottleneck and the bottleneck is on "buffer busy waits," but what do I do to solve it? There are some guesses I can make, and I can even use some data in STATSPACK to help, but there is no way to conclusively analyze this wait bottleneck or most wait events with STATSPACK. To solve most wait event bottlenecks we need more detailed information than is available in STATSPACK. What information do we need? We mainly need to know what SQL, sessions, objects, and wait details are involved. How do we get that information? We get it by sampling v$session_wait, i.e., via ASH. We also need to understand how to solve each wait event, and specifically in this case how to solve "buffer busy waits." A complete explanation of buffer busy waits is beyond this presentation but a little information on them will suffice. A buffer busy wait has two central causes. The first cause is contention on disk reads and is equivalent to an I/O wait. The second cause is sessions bottlenecking on writing to a particular block. In order to solve the problem, I need to know the kind of buffer busy wait bottleneck—read or write—and I need to know the objects and queries involved. With this information I can take steps to alleviate the problem.

How do we find the objects and SQL involved in the buffer busy waits? The information we need can be found in a table v$session_wait when the problem exists, but v$session_wait only represents the current state of the system. If our problem is intermittent or passed, the information we need from v$session_wait will be lost. The lack of history in V$session_wait is where the new Oracle 10*g* concept, Active Session History (ASH), comes in. ASH at its simplest is just reading v$session_wait every second and storing the data. Oracle does this automatically in 10*g*, but it has been possible in every version since Oracle 7 by running the following query:

```
column sid format 999
column username format a10
column serial format 99999
column hash_value format 9999999999
column event format a15
column p1 format 9999999999
column p2 format 99999999
column p3 format 9999

select
                nvl(s.username,s.program) username,
                s.sid sid,
                s.serial# serial,
                s.sql_hash_value hash_value,
                substr(decode(w.wait_time,
                        0, w.event,
                        'ON CPU'),1,15) event ,
                w.p1  p1,
                w.p2  p2,
                w.p3  p3
from             v$session       s,
                 v$session_wait  w
where        w.sid=s.sid
          and s.status='ACTIVE'
          and s.type='USER'
```

I joined v$session_wait with v$session to get the SQL hash value as well. ASH can be extended with similar joins to build up a rich source of sampled data.

From the above query we get output like this:

```
USERNAME SID  SERIAL HASH_VALUE  EVENT             P1       P2       P3
-------- ---  ------ ----------- ----------------- -------  -------  ---
SYS      64   8717   4116021597  PL/SQL lock timer 300      0        0
SYS      58   19467  961168820   ON CPU            16508152 1        0
STARGUS  71   6251   1311875676  direct path write 201      2155902  127
(CJQ0)   9    1      0           rdbms ipc message 500      0        0
```

In the case of the buffer busy wait, I've been running the above query and storing it in a table that I call v$ash. I can now read v$ash and look for the detail information I need to solve the buffer busy wait problem. First I want to know what kind of buffer busy waits were happening, read or write. Read or write can be determined from the P3 column, which tells us whether the buffer busy wait was a read or write bottleneck. (The P1, P2, and P3 columns have a different meaning for each wait event.) If P3 is 200 or greater, the problem is write contention; if P2 is less than 200, it is a simple I/O wait.

```
SQL>  Select count(*), p3
      from v$ash
      where event = 'buffer busy waits'
      group by p3;

  COUNT(*)  P3
----------  ----
      3423  220
```

We see that all the P3 values are 220, thus this is a write problem (below 200 is IO, above 200 is write contention). Write problems can happen for all sorts of reasons, depending on the block type the writing is occurring on; in order to solve this issue we need to know what kind of block the problem is occurring on. We can find this out by looking at P1 and P2 for the buffer busy waits that represent the file and the block:

```
select count(*),   p1 filen,   p2 blockn ,   hash_value
from v$ash
where event='buffer busy waits'
group by p1, p2, hash_value;
```

```
  COUNT(*)   FILEN      BLOCKN
----------   --------   --------
         1    11         90644
         2    11         90651
         3    11         98233
         1    11         104767
         3    11         113291
         1    11         119842
         1    11         119856
         3    11         121632
         1    11         126334
```

I'll just pick the last block and find out what object it's coming from:

```
column segment_name format a30
select owner,
segment_name,
segment_type,
block_id, blocks+block_id
from dba_extents
where file_id = 1
and 126334 between block_id AND block_id + blocks-1;

OWNER  SEGMENT_NAME  SEGMENT_TY   BLOCK_ID  BLOCKS+BLOCK_ID
------ ------------  ----------   --------  ---------------
SYSTEM TOTO1          TABLE         125201           127249
```

From this result I know that it's a data block of a table and not a header block (because the header block would be the first block in the object). Now the question is what are the sessions doing? I can look at the SQL and see what they are executing:

```
SQL> select count(*),  p1 filen, p2 blockn , hash_value
  2  from v$ash
  3  where event='buffer busy waits'
  4  group by p1, p2, hash_value;

     COUNT(*)   FILEN      BLOCKN      HASH_VALUE
   ----------   --------   --------    ----------
            3   1          94609        558666863
            2   11         81163        558666863
            2   11         87123        558666863
            1   11         90644        558666863
            2   11         90651        558666863
            3   11         98233        558666863
            1   11         104767       558666863
            3   11         113291       558666863
            1   11         119842       558666863
            1   11         119856       558666863
            3   11         121632       558666863
            1   11         126334       558666863
```

I can see that the same SQL is causing all the buffer busy waits because the hash value is the same. Now I can find out what that SQL statement is:

```
select sql_text
from v$sqltext
where hash_value=558666863;

SQL_TEXT
------------------------------------------------
INSERT into toto1 values (:b1,lpad('a',1000,'a'))
```

With this final piece of information, it looks like the segment is probably missing free lists or not in an Automatic Segment Space Managed (ASSM) tablespace. Let's check these possibilities.

```
select FREELISTS,  TABLESPACE_NAME
from dba_tables
where table_name='TOTO1' and owner='SCOTT';

    FREELISTS  TABLESPACE_NAME
   ----------  ------------------------------
            1  USERS

select tablespace_name, SEGMENT_SPACE_MANAGEMENT
from dba_tablespaces;

TABLESPACE_NAME              SEGMEN
---------------------------  ------
SYSTEM                       MANUAL
UNDOTBS1                     MANUAL
TEMP                         MANUAL
USERS                        MANUAL
TOOLS                        AUTO
```

We see that that the table is in a tablespace, USERS, that does not have ASSM active.

In summary, the table TOTO1 has wait contention on writes to blocks. Multiple sessions are concurrently inserting into it, and it has no free lists nor is it in an ASSM tablespace. This causes all sessions inserting to insert into the same block, the last free one on the one free list and thus fight over exclusive write access to that last block. The solution is either to put the segment into an ASSM tablespace and let Oracle handle the contention or put free lists on the segment, which is more precise. If I choose free lists, how many do I need? I need as many as there are concurrent users inserting into the table, which I can get with:

```
select count(*), sid, serial  from v$ash
where event='buffer busy waits'
group by sid, serial;

    COUNT(*)   SID     SERIAL
    --------   -----   --------
           7   12      79
           4   14      99
           4   19      770
           8   20      176
```

There were four sessions inserting during the period of ASH data I looked at, thus in my case I need four free lists.

Extended Power of ASH

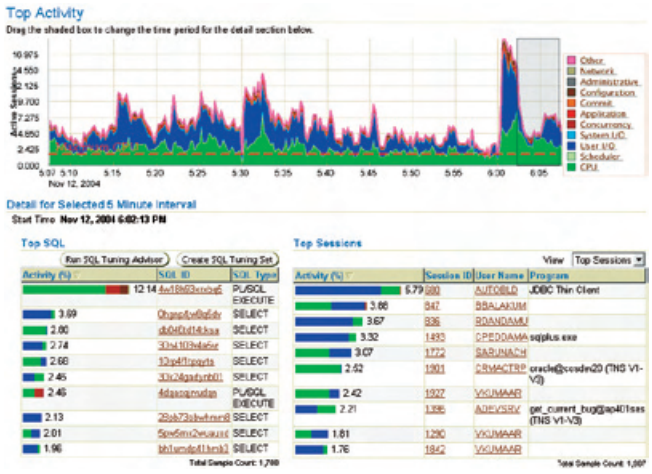
The above example is simplified. The two other important pieces of data that also need to be tracked are sql_text and sample_time. Sample_time is needed to group data by performance bottleneck time periods, and sql_text should be collected on an ongoing basis in case the SQL found in v$session_wait gets kicked out of the shared pool. At the end of the day, collecting ASH becomes quite complicated with the need to partition and purge data and send it to a different repository to reduce overhead on the target, combined with the need to collect other bits of information such as maps of user# to username and object# to object name. To see a package that handles most of these tasks, refer to **www.perfvision.com/ash.php**.

Even with a good collection running, it takes some non-trivial SQL to mine the richness of the data. And even with a good set of scripts to analyze the data, it takes a number of well-written SQL statements to drill down sufficiently into the problem.

Luckily, 10*g* supplies all of this functionality. Oracle 10*g* automatically collects all of the data, analyzes all of the wait

events, reports on bottlenecks, and supplies solutions.

On top of this, the data is displayed graphically in OEM 10*g* in a way that condenses dozens of SQL statements that would take time and effort to run and comprehend into a visual that can be immediately understood:



Even in this reduced graphic, we can quickly and easily see that there is a performance problem on the machine. In this case there is clearly a CPU bottleneck, because the Top Activity page reports that CPU (in the top chart, the green) is above the maximum CPU on the machine (the dotted red line). This means that the machine is at 100% CPU and there are processes in the run queue. The chart at the bottom right also shows that one SQL statement is using much more CPU than any other SQL statement, so this is the statement to tune. It only takes a second to scan the graphic and see where the problem is, and it's all based on ASH.

Summary

ASH is a leap in performance-tuning technology. It took a clear vision of the future and a bit of a leap of faith to let go of the compulsive need to have exact measurements and instead embrace sampling a technology that statistically approximates the same value with little loss of accuracy but at the same time brings a great wealth of information necessary for analyzing the performance of a database.

ASH can be simulated on any version of Oracle since v7 with the simple act of repeatedly sampling and storing the values from v$session_wait into a historic table. Although the idea is easy, the implementation turns out to be fairly complex. Luckily Oracle doesn't only implement ASH in version 10*g*; more importantly, Oracle implements a diagnostic package that reads the data, analyzes the data, and reports the findings and solutions automatically, so you don't even need to know all the esoteric wait analysis. The best part is that OEM 10*g* exposes all of the data and analysis graphically in order to provide an interface that communicates the state of the system quickly and clearly. ▲

Kyle Hailey is a regular speaker at NoCOUG conferences and needs very little introduction to NoCOUG members. He is a founding member of the Oak Table and co-author of Oracle Insights: Tales of the Oak Table, together with other NoCOUG regulars like Jonathan Lewis, Cary Millsap, and Gaja Krishna Vaidyanatha. He had a long and distinguished career at Oracle and designed the performance-tuning module in Oracle Enterprise Manager 10g.

Don't Monitor Your *Database* Don't Diagnose Your *System*

by Gary Goodman

Bring the business back into business system performance optimization! What I want you to think about is what's keeping the CEO of your company awake at night? Is it optimizing the business or is it optimizing the database? (And if you're honest with yourself, you're going to find that the answer is probably *both*.) By both I mean that the database has to be doing all the things it needs to do to be able to give the business the performance characteristics that are required to meet all of the company's goals.

Now there are some obvious disconnects here. Businesses focus on things like order lines, products shipped, and invoices paid—customer-facing transactions that help create profitability, return on investment, and cash flow for the company. Conversely, DBAs deal with a different world. They deal with things like tablespace management, logical I/Os, statistics, and SQL execution plans—very technical things tied specifically to an Oracle database. The problem that I've seen throughout my 20-plus-year career in an Oracle space is that there's a major disconnect between the people managing an Oracle environment and the business that they're trying to support. They don't always understand what they're doing and what impact it's going to have on the business.

> ***"When users use the word system, they don't mean what a DBA thinks it means. They're not talking about memory and disk and latches and locks and SQL."***

When users use the word system, they don't mean what a DBA thinks it means. They're not talking about memory and disk and latches and locks and SQL. They're talking about the three reports and the four online forms that they use. If you're not looking at those three reports and four online forms, you have virtually no chance of ensuring that those things are performing the way they need to be.

So first, quit monitoring your *database*. Don't think of your database as one single entity. Think of your database as a collection of prioritized business tasks that all have a different impact on the profitability of your company. Once you've done that, focus your monitoring on those most critical business activities.

Second, don't diagnose your *system*. You're going to get lost—it's going to lead you down paths that may have no impact on your most critical tasks. Diagnose the specific performance of targeted critical transactions—it's the only way to get the exact information you need to understand how to make it faster.

Don't let systemwide data lead you down the wrong path. In virtually every firefight that we face, the first thing we see are reams and reams of output from some systemwide monitoring tool, and while they do certain things well, what they don't do is diagnose specific critical business tasks. If that's your goal, you need to be able to segregate that data so that you can understand what's consuming the time for the most critical business tasks that are important to your CEO.

And finally, don't just functionally QA your system; bring performance accountability in, whether it's while you're writing code as a developer, being able to test for performance anomalies during unit test, or as late as changing things later in a production environment. Understand the performance impact of everything you do as well as the functional impact.

This is the world we set out to fix with Hotsos. The solutions we've created cover three main areas. First, proactively monitoring your *business* tasks. We're not monitoring databases, we're monitoring the performance of your business. Once issues arise out of that monitoring, you need the ability to rapidly diagnose the most critical business tasks in terms of what's consuming the response time. And then once you've got that diagnostic done and you've come up with a remedy, ensure that when you promote that remedy in, you don't create as many problems as you just fixed. You don't want to introduce new performance issues as you fix your problems, so you need the ability to bring performance quality assurance into the QA process. This is a cycle that you're going to live in, in terms of proactively monitoring, diagnosing, and then QA.

Hotsos performance suite of solutions is the only suite dedicated to supporting your business, not your database. With Hotsos you can proactively monitor all your critical tasks, rapidly diagnose the root cause of any performance issue in those tasks, and then QA for unrelated or unintended performance impacts as you promote these changes. We've given you complete coverage toward creating and maintaining an optimal Oracle environment. ▲

Gary Goodman and his colleagues at Hotsos are world-class Oracle system performance talent. Whether you're a database administrator, performance analyst, line of business manager, or application developer, Hotsos can help you. Go to **www.hotsos.com** *to find out more.*

Statistics—How and When

Ask the Oracles!

Mogens Nørgaard: Lies, damned lies, and statistics. Some things never change, including old jokes . . .

Anyway, I've always been mystified by statistics gathering in databases until Dave Ensor, the utterly bitter and twisted Scotsman with the razor-sharp mind, ended a presentation at the UKOUG conference in Birmingham some years ago with these words . . . and of course I remember them 100% correctly (smiling).

"Oh, and by the way, could you please stop gathering statistics constantly? I don't know much about databases, but I do think I know the following: small tables tend to stay small, large tables tend to stay large, unique indexes have a tendency to stay unique, and non-unique indexes often stay non-unique."

> ***"Small tables tend to stay small, large tables tend to stay large, unique indexes have a tendency to stay unique, and nonunique indexes often stay nonunique."***

With those words, he finished the presentation. On the way out of the room I asked Anjo Kolk what he thought about this, and he told me it was very true. People gather statistics way too often.

A couple of weeks later I got a call from a large customer, who had discovered that they hadn't gathered stats on their ERP system with more than 1,000 users for two years.

Normally, I would have laughed at them and told them to get it going. However, since I had heard the above from Dave Ensor, I asked instead: "So how has the system been performing in that period?" and got the surprising answer that it had worked very, very well and that the users had been really happy.

I bet. Because if you can keep the stats constant there's a very high chance that your SQL statements will execute in the same way every day, meaning that the vast majority of users will have the same experience w.r.t. response times day in and day out, which is what they want: predictability is king.

The goal of the CBO development team is to make it 100% dynamic, which interestingly enough is the way of the SQL Server optimizer too, whereas DB2 has this notion of having the execution plan stored physically in the data dictionary to ensure stability and predictability.

If you think about what Dave Ensor was really saying, it makes a lot of sense not to surrender to the view that everything should be dynamic and Oracle knows best.

> ***"I have advised many customers to stop analyzing, thereby creating a more stable environment overnight."***

Since the above experience, I have advised many customers to stop analyzing, thereby creating a more stable environment overnight.

However, there are, of course, good cases for re-gathering stats. They include situations where the amount or distribution of the data changes dramatically.

But instead of suggesting various "good numbers" for the percentage of rows changed in a table before re-gathering stats or percentage of tables/indexes to be analyzed, I'd like to suggest a different approach:

Monitor the changes in execution plans and/or performance for the individual SQL statements. If the execution plan for a given statement changes, you should examine whether it's for the better or for the worse. On the other hand, if the execution plan stays the same but performance becomes worse, you should examine the real reason(s) for this, and perhaps as a consequence re-gather stats.

That way, you'd leave stuff alone that works very well, thank you, and you'd put your efforts into exactly the things that have become worse. In the process, you have also saved a lot of hardware resources and minimized the load on the system because of stats gathering.

> ***"Leave stuff alone that works very well and put your efforts into exactly the things that have become worse."***

Torben Holm of Miracle A/S has a free tool called MirPlan, which will do some of this for you. You can contact me (**mno@MiracleAS.dk**) *or him* (**thh@MiracleAS.dk**) *if you're interested.* ▲

---

*Mogens Nørgaard is the CEO of Miracle A/S (***www.miracleas.dk***), a database knowledge center and consulting/training company based in Denmark, and is the co-founder and "father figure" of the Oak Table network. He is a renowned speaker at Oracle conferences all over the world and organizes some highly respected events through Miracle A/S, including the annual Master Class and the Miracle Database Forum. He is also the co-founder of the Danish Oracle User Group (OUGKD), and was voted "Educator of the Year" in Oracle Magazine's Editor's Choice Awards, 2003. Mogens can be reached at* **mno@miracleas.dk**.

Jonathan Lewis: Imagine you own a supermarket: as time passes, the number of distinct suppliers you have doesn't change much, the number of distinct products changes slowly and steadily, but the volume of sales changes constantly. What does this tell you about the statistics that Oracle needs to describe your data?

There are some statistics about your data that can be left unchanged for a long time, possibly forever; there are some statistics that need to be changed periodically; and there are some statistics that need to be changed constantly.

You may even need to massage some statistics so that they describe part of the data rather than all of the data—and there is no way that any supplied Oracle package can create such statistics. For example, many businesses hold data for seven years, but the end-users are often only interested in what's been happening in the last few weeks. The "average statistics" for seven years may give the optimizer a misleading image of the data the user wants to query. (Think of the odd sales patterns that a supermarket has around Christmas and Thanksgiving.)

For most people, the critical problem is to work out how to do the minimum amount of work to generate the best possible statistics, and it can take a lot of up-front effort to work out which statistics fall into which of the classes identified above. Fortunately it is possible to start simply and enhance your collection strategy over time.

My guidelines are simple: you may as well compute statistics for small tables, but large tables usually need just a small sample size (can you check this on a backup); some indexes will need their clustering_factor adjusted; a few columns will need histograms; a few columns will need specially constructed programs to manufacture "business aware" statistics; and partitioned tables will, in general, need programs to construct their statistics (possibly only at the table level after a "normal" call to the dbms_stats package has created some new partition-level statistics).

The biggest problem is that you need to understand the data. You may be able to take advantage of table monitoring to determine which tables are subject to significant change and v$col_usage to see how columns are used in predicates, but neither of these helps you understand which columns have data distributions that require special treatment and which indexes need correction.

Ultimately I believe you need a table-driven mechanism—and initially you could probably set up a simple system to emulate your current stats collection. For each non-partitioned table, you need to record the frequency of collection, sample size, collection method, and whether or not to include indexes—in other words, most of the parameters to the gather_table_stats() procedure. For tables where you don't automatically gather index statistics, you need to include records for the relevant indexes with a similar list of parameter values. Finally, for any special columns, you need entries showing how to handle them—which may simply mean calls to create histograms. In all three cases, you may choose to reference a (homegrown) procedure that specifies a method for generating a completely artificial (though appropriate) set of figures. For partitioned tables, my approach is to design a custom stats collection package for each table as soon as I define the table.

Your stats collection routine can start as a loop to scan the table and obey the instructions it finds there—and you can start simply with a driving table that emulates your current dbms_stats calls, enhancing the system as your knowledge grows.

There's a lot more to say—but with only 600 words, no room to say it—but this is the core of the optimum strategy. ▲

Jonathan Lewis is well known to the Oracle community as a consultant, author, and speaker, with more than 18 years' experience in designing, optimizing, and troubleshooting on Oracle database systems. His latest book is Cost-Based Oracle Fundamentals, which is the first of three volumes on understanding and using the cost-based optimizer.

"There are some statistics about your data that can be left unchanged for a long time, possibly forever; there are some statistics that need to be changed periodically; and there are some statistics that need to be changed constantly . . . You may even need to massage some statistics so that they describe part of the data rather than all of the data . . . The biggest problem is that you need to understand the data."

Christian Antognini: The package dbms_stats provides many features to manage object statistics. The question is how and when should we use them to achieve a successful configuration? It's difficult to answer this question. Probably no definitive answer exists. In other words, there is no single method that can be implemented in all situations. Nevertheless, let me share my thoughts with you in this area.

The general rule, and probably the most important one, is that the query optimizer needs object statistics that describe the data stored in the database. Therefore, when data changes, object statistics should change as well. As you may understand, I am an advocate of regularly gathering object statistics. Those who are opposed to this practice argue that if a database is running well, there is no need to re-gather object statistics. The problem with that approach is that more often than not, some of the object statistics are dependent on the actual data. For example one statistic that commonly changes is the low/high value of columns. True, there are not many of them that change in typical tables, but usually those that change are critical because they are used over and over again in the application. So in practice, I run into many more problems caused by object statistics that are not up-to-date than the opposite.

Obviously it makes no sense to gather object statistics on data that never changes. Only stale object statistics should be re-gathered. Therefore it is essential to take advantage of the feature that logs the number of modifications occurring to each table. In this way we re-gather object statistics only for those tables experiencing substantial modifications. By default a table is considered stale when more than 10% of the rows change. This is a good default value. As of Oracle 11*g*, this can be changed if necessary.

> ***"It's difficult to answer this question. Probably no definitive answer exists. In other words, there is no single method that can be implemented in all situations."***

The frequency of the gathering is also a matter of opinion. I have seen everything from hourly to monthly or even less-frequent gatherings as being successful. It really depends on your data. Nevertheless, when the staleness of the tables is used as a basis for re-gathering object statistics, intervals between two gatherings that are too long lead to too many stale objects and, therefore, the time required for the gathering could be too long. For this reason I like to frequently schedule the gatherings to spread out the load and to keep a single gathering as short as possible. If your system has daily or weekly low-utilization periods, scheduling gatherings during those periods is usually a good thing. If your system is a true 24/7 system, it is usually better to use very frequent schedules

(many times per day) to spread out the load as much as possible.

If for some good reasons object statistics should not be gathered on some tables, as of Oracle 10*g* you should lock them. In this way the job that regularly gathers object statistics will simply skip them. This is much better than completely deactivating the gathering job for the whole database. Unfortunately, up to Oracle 9*i* there is no feature that can be used simply to achieve the same goal.

If you have jobs that load or modify lots of data, you shouldn't wait for a scheduled gathering of object statistics. Simply make the gathering for the modified objects part of the job itself. In other words, if you know that something has substantially changed, trigger the gathering immediately.

As of Oracle 10*g*, you should take advantage as much as possible of the default gathering job. In order for this to meet your requirements, you should check—and if necessary change—the default configuration. Since a configuration at object level is only possible as of Oracle 11*g*, if you have particular requirements for some tables, you should schedule a job that processes them before the default job. In this way the default job will simply skip them. Locks might also be helpful to ensure that only a specific job is re-gathering object statistics on those critical tables.

> ***"As of Oracle 10g, you should take advantage as much as possible of the default gathering job. In order for this to meet your requirements, you should check—and if necessary change—the default configuration . . . Locks might also be helpful [if you have particular requirements for some tables]."***

If a gathering leads to inefficient execution plans, there are two things to do. The first one is to fix the problem by restoring the object statistics that were successfully in use before the gathering. The second one is to understand why inefficient execution plans are generated by the query optimizer with the new object statistics. For that purpose, at first you should check whether the newly gathered statistics correctly describe the data or not. For example, it is possible that sampling along with a new data distribution will lead to different histograms. If object statistics are not good, you know that the gathering itself—or possibly a parameter used for the gathering—is the problem. If object statistics are in fact good, there are two more possible causes. Either the query optimizer is not correctly configured or the query optimizer is making a mistake. Over the latter we have little control. On the other hand, we should be able to find a solution for the former. In any case, you should avoid thinking too hastily that gathering object statistics is inherently problematic and, as a result, stop gathering them regularly. ▲

---

*Since 1995, Christian Antognini has been focusing on understanding how the Oracle database engine works. He is currently working as a principal consultant and trainer at Trivadis AG (***www.trivadis.com***) in Zürich, Switzerland. If he is not helping one of his customers get the most out of Oracle, he is somewhere lecturing on optimization.*

Wolfgang Breitling: When thinking about or deciding on a statistics-gathering strategy, we need to keep the eventual goal in mind. Gathering statistics is not an end in itself. The sole objective is to give the optimizer all the information it needs to generate good access paths for the queries it is confronted with. We also should keep in mind that the statistics are not the only input into the access path generation process. There are at least three major parts that play a role:

a. The optimizer itself, including its built-in assumptions and heuristics
b. The statistics
c. The SQL to be optimized

Frequently I hear developers bemoan the demise of the rule-based optimizer and the (plan) stability or predictability it afforded them. The claimed chaos created by the cost-based optimizer can be traced back to lack of understanding as well as a lack of stability of parts a and b above. The pace of development of the CBO is frantic. Patchsets and even patches often bring with them a change in "behavior" of the CBO. Then what about the statistics? Oracle keeps hammering out the message that the optimizer needs fresh statistics in order to create optimal plans. "Stale" statistics are evil. So DBAs have

> ***"Gather statistics as rarely as possible and as frequently as necessary . . . Statistics are not an end in themselves but are there so that the optimizer has information to create good plans. If it does not, you need to change the SQL, the statistics, or the optimizer."***

implemented scheduled statistics-gathering jobs to run when there are resources to spare, on weekends, or even nightly. But do stale statistics really deserve the bad reputation they get? One could argue that stable statistics will foster some semblance of plan stability. If the optimizer doesn't change—between upgrades or patches—the SQL does not change; and if the statistics do not change, there is no reason for the optimizer to generate a different plan. I allege that this is unequivocally true, provided all inputs remain the same. After all, the optimizer is not a game governed by chance but a deterministic algorithm.

"It is my firm belief that most scheduled statistics-gathering jobs do not cause much harm only because (most) changes in the statistics were insignificant as far as the optimizer is concerned—meaning that it was an exercise in futility."

The reason we may not be able to collect statistics once and leave them be, at least until the next upgrade to Oracle and the optimizer, is that even the third part (c) above, the SQL itself, is often not as static as it seems. Think of bind variables, or time-related queries (e.g., "give me the sales figures for the past month"). The translation of "past month" into dates changes as time progresses. Therefore, we will need to update some statistics occasionally. It is imperative to know which ones.

Beyond that, we ought to keep the other goal in mind when devising a statistics gathering strategy: "predictable performance"—closely tied to "plan stability," which in turn is aided by stable statistics—unless the underlying data changes so dramatically that what used to be a good plan no longer is. But that is rare; business does not change that dramatically, so why would the data that describes it? It is my firm belief that most scheduled statistics-gathering jobs do not cause much harm only because (most) changes in the statistics were insignificant as far as the optimizer is concerned—meaning that it was an exercise in futility.

Gather statistics as rarely as possible and as frequently as necessary. *With its automatic statistics gathering, Oracle ties the frequency to the (perceived) rate of change. But some tables may be completely replaced without materially outdating the statistics that describe them, while for others the st**a**tistics can become outdated—as far as some crucial SQL is concerned—by the addition, deletion, or update of just a single row.*

Gather statistics consistently. *Do not gather statistics in the scheduled job with a full compute and for a one-off refresh with an estimate—or vice versa.*

Gather only as many statistics as necessary. *E.g., do not gather a histogram on every column in the database just in case.*

Do not be afraid to modify the gathered statistics, or even create some yourself, if the gathered statistics lead the optimizer to generate a bad plan. *Remember, the statistics are not an end in themselves but are there so that the optimizer has information to create good plans. If it does not, you need to change the SQL, the statistics, or the optimizer.*

Last but not least: if what you are doing now for statistics gathering does not cause you any grief, do not change it because of this article! ▲

Wolfgang Breitling was born in Stuttgart, Germany, and studied mathematics, physics, and computer sciences at the University of Stuttgart. Following several years as a systems programmer for IMS and later DB2 databases on IBM mainframes, he got on the project to implement Peoplesoft on Oracle. In 1996 he became an independent consultant specializing in administering and tuning Peoplesoft on Oracle. The particular challenges in tuning Peoplesoft, often with no access to the SQL, motivated him to explore Oracle's cost-based optimizer in an effort to gain a better understanding of how it works and use that knowledge in tuning. He has shared the findings from this research in papers and presentations at IOUG, UKOUG, local Oracle user groups, and other conferences and newsgroups dedicated to Oracle performance topics.

Toad for Oracle 9.1

by Cecilia Weiss and Nicole Tamms

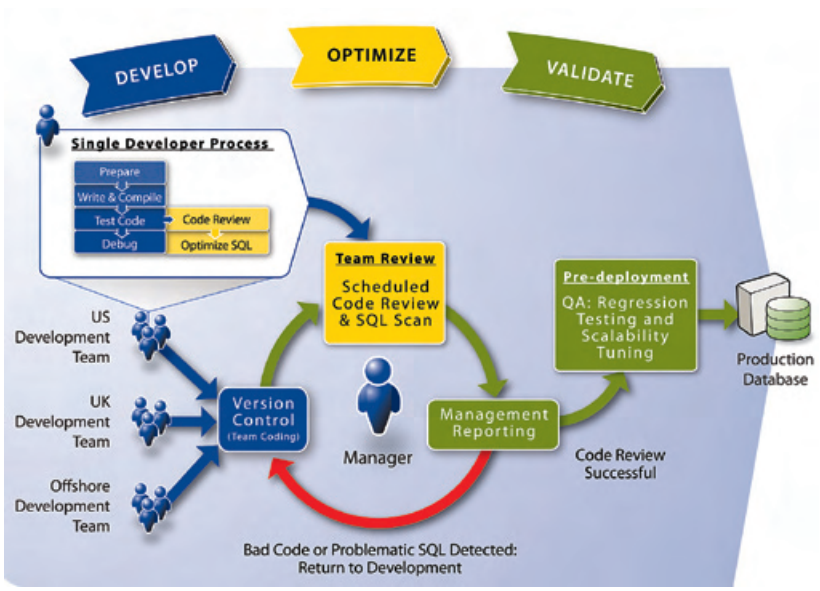
Introduction by Iggy Fernandez

Here's how to answer the stock question that every interviewer asks you at the end of the job interview: "Do you have any questions for us?" Resist the temptation to ask for the cell phone number of the database developer or database administrator who quit! Instead ask in a nonchalant voice, "What software tools do you provide to your database developers and database administrators?" Think twice about accepting a job offer if none of the answers is Toad, because it is simply the best-of-breed tool for Oracle database development and database administration. Whether it's a simple task, such as formatting SQL code, or one that is more complex, Toad delivers with style and panache.

Give me Toad or give me a job in another company!

Toad Development Suite

As an Oracle database professional, you know that your environment undergoes constant change and, therefore, you rarely have total control over it. Changes typically result from new development projects or modifications to existing code, and functional or performance problems can arise when applications are deployed to production. Depending on the severity of the problems, the consequences can range from additional development cycles to downtime that negatively impacts the bottom line.



Quest Software understands the challenges that enterprise development teams face. Often, project teams are dispersed across many locations—some throughout the world—and comprise technologists whose skills and experience can vary a great deal. As development projects evolve, they become nearly impossible to manage effectively. This often leads to code inconsistencies, performance problems and bugs in production, where issues are significantly more expensive to fix than they would be in development.

Quest Software's new Toad Development Suite for Oracle enables database development best practices. It provides a consistent, repeatable, and measurable process for managing the database development process, while ensuring that the code being deployed to production is of the highest quality regardless of the users' skill set.

- Develop code faster, cleaner, and with fewer bugs using the first automated unit testing tool on the market, Quest Code Tester for Oracle.
- Optimize SQL code by scanning and rewriting it automatically, displaying the best-performing SQL alternatives.
- Validate code function, performance, and scalability under production-like conditions.

What if you and your team could employ a process which ensures that the code you deploy functions properly and has been debugged, tuned, and tested for production?

With the Toad Development Suite for Oracle, you and your project team have the opportunity to deliver just that. With Toad, you can build code faster and cleaner, with fewer bugs.

Prepare—You can define the measurements for code success by describing the tests required to verify that functional requirements will be met.

Write and Compile— Build the programs according to predefined project-coding standards and apply code formatting.

Test Code—Perform code testing to identify the gaps between the way the code is functioning and the way it should be running by executing automatically generated test code using Quest Code Tester.

Code Review—Using Code Xpert, you can check the code quality against a predefined list of rules that can be saved for each project on the network. Rules can be shared among the team members to ensure consistency.

Debug—With an advanced source code debugger, you can quickly identify and fix the lines of code that are causing issues.

Optimize SQL—With an automated SQL Scanner, you can scan code for problematic SQL statements that may cause serious bottlenecks in production, and with one click, you can find the most efficient SQL alternative.

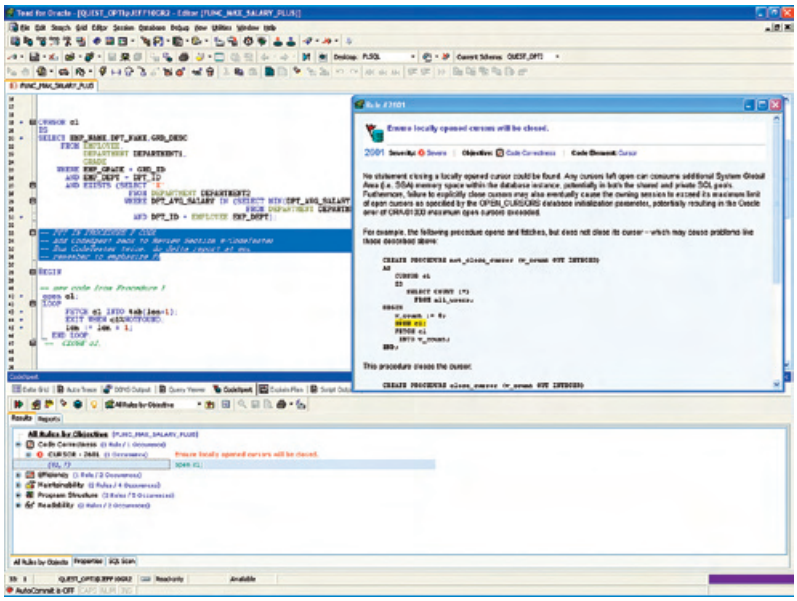
New Features of Toad for Oracle 9.1

Toad for Oracle 9.1 will help you implement database development best practices for Oracle within your organization so you can efficiently and accurately perform daily development and administration tasks.

Toad Group Policy Manager—The Toad Group Policy Manager provides a facility by which multiple copies of Toad for Oracle within an organization can share the same set of options. It consists of a Windows service that runs on a Windows server and publishes subsets of option data to defined groups of Toad users via TCP/IP, and an editor that is used to define option sets and user groups. Users can be restricted from changing published Toad options or permitted to alter them.

Using the Toad Group Policy Editor, policies and standards can be distributed throughout a group environment.

Action Recall—A versatile tool that ensures consistency and repeatability of operations for individual users or groups by recording actions in Toad and storing them for future use, sharing them with other Toad users, or scheduling them.



Recalling an Action is simply the ability to re-perform a distinct operation or sequence of operations in Toad, such as saving Oracle data to Excel, transferring a file via FTP, exporting DDL, or sending an email.

Toad Tips—A suite of context-sensitive pop-up windows that provide users with information on Toad as they work through the product. Each Toad Tip relates to a window in Toad and is designed to increase awareness and usage of important features, so users are better informed about the product and see the full value of Toad.

The display of Toad Tips can be customized by the user through a Notes page, selectively turned off, or turned off completely.

Code Xpert Scheduling—An enhanced automation mechanism within Toad that provides managers with the ability to simplify the preparation of a scheduled code review. Code Xpert Scheduling reduces the process of submitting code files for automated review down to just two clicks.

Code Xpert not only makes it easier for managers to ensure that code meets the required quality standards; it also enables the scanning of code for problematic SQL statements.

Smart Watches—A feature within the Toad debugger that makes debugging more efficient by helping developers better understand the location of PL/SQL variables in their program in order to set Watches on them.

Developers use Watches to see what happens to a variable's value as they run a PL/SQL program through the debugger. Since the debugger enables developers to execute PL/SQL code line by line, they can have a Watch display the current value(s) in a Watches panel.

Smart Watches allow Toad to find all of the available variables and list them in the Smart Watches panel. The developers can then decide which ones they want to watch and drag and drop them from the Smart Watches panel to the Watches panel, where they can be stored for future debugging sessions.

Toad World

Created just for Toad users, Toad World is a one-stop community portal where members can share ideas, ask questions, and learn from Quest experts such as Steven Feuerstein, Mike Ault, and Bert Scalzo; the Toad development team; and fellow database professionals. It offers community interaction, expert blogs, podcasts, user forums, tips and tricks, and more.

Toad World also keeps users up-to-date on the newest features of Toad. For example, Quest recently provided Toad World members with information on the latest release of Toad for Oracle, version 9.1: members could listen to a podcast on the latest features of Toad for Oracle 9.1 prior to its release.

Toad World is a free portal just for Toad users and can be accessed at **www.toadworld.com**.

Summary

Toad for Oracle has evolved over more than 10 years. With input from Quest Software's Oracle experts like Steven Feuerstein and Bert Scalzo, as well as contributions from its loyal user community, Toad has grown into the most advanced database development and administration solution on the market. ▲

Cecilia Weiss and Nicole Tamms are product marketing managers at Quest Software. For more information about Toad and other Quest products, visit **www.quest.com**.

Peekaboo!

by Iggy Fernandez

Jeepers Creepers, where'd ya get those peepers?
Jeepers Creepers, where'd ya get those eyes?
Gosh all git up, how'd they get so lit up?
Gosh all git up, how'd they get that size?
—1930s song

Iggy Fernandez

Case Study

A big website was repeatedly brought to its knees when the execution plan for a certain very common query would undergo a dramatic change and Oracle would begin using an unrestricted "Cartesian" algorithm to join intermediate results. The response time for the query would go from a few hundredths of a second to several minutes, and Average Active Sessions would go from the low single digits to several hundred.

Each time, the database administrators would bring the problem under control by rebuilding the statistics of the tables involved in the query. The situation would remain stable for about a week, at which point the problem would suddenly reoccur without any apparent triggering cause.

"Bind variable peeking upsets the conventional wisdom that query execution plans are stable if the data statistics do not change."

Investigation showed that data statistics were automatically regenerated every night. The first hypothesis was that the data was changing after the statistics were regenerated. However it turned out that the data was static and that it was not necessary to regenerate the statistics nightly.

Some other explanation was necessary and the problem was finally traced to the query optimizer's use of the "bind variable peeking" technique.

Theory Refresher

Execution plans for SQL queries are not hardwired into the Oracle database—they are only generated when the query is first encountered.[1] The query execution plan is then cached for as long as possible[2] and reused when an identical query is submitted.

This is certainly efficient since it is computationally expensive to generate a query execution plan. However there is an important catch: queries that use bind variables and only differ in the values of the variables are considered identical in order to promote reuse. But a query execution plan that works well for particular values may not work well for other values—the problem becomes acute if an execution plan is generated from particularly unrepresentative values.

Prior to Oracle 9*i*, the cost-based optimizer did not consider the values of the bind variables, but instead applied various rules of thumb to the data statistics to produce cardinality estimates that were equally good (or equally bad) for all values of the bind variables. Beginning with Oracle 9*i*, the cost-based optimizer began considering the values of the bind variables when generating query execution plans. For example, consider a restriction such as "AGE > :1."

Prior to Oracle 9*i*, the query optimizer might assume that 5% of the data will satisfy the restriction and apply this rule of thumb to available statistical information, such as the number of rows in the table, to estimate the cardinality of the resulting data set. Beginning with Oracle 9*i*, Oracle will consider the value of the bind variable and produce a more refined cardinality estimate if column histograms are available. This strategy is called "bind variable peeking."

Bind variable peeking upsets the conventional wisdom that query execution plans are stable if the data statistics do not change. A database that is quietly humming along can suddenly go quite berserk if a query execution plan is generated from unrepresentative values of bind variables—a children's nursery rhyme comes to mind:

There was a little girl who had a little curl
Right in the middle of her "forrid."
When she was good, she was very, very good,
But when she was bad she was horrid.

Smoking Gun

After much trial and error, the database administrators were able to reproduce the problem at will using a particular combination of query variables that did not match any data in one of the tables. This meant that the entire chain of joins in the query produced no data at all and, therefore, the use of the

[1] This is true even when using "stored outlines" or "SQL profiles"—they are collections of hints, not actual execution plans.

[2] Cached plans are managed using the "least recently used" algorithm and may be discarded to make room for others if not used frequently enough.

Cartesian join algorithm past a certain point did not degrade query performance.

Here is the report produced by the "tkprof" utility—like every query execution plan, it should be read top to bottom and right to left. The key step is highlighted in red and produces no data rows at all.

The report also shows that Oracle was smart enough to stop processing the query at that point, as evidenced by the number of consistent reads accumulated at the succeeding steps.

Oracle took a mere 3358 microseconds—less than a hundredth of a second—to process the query the first time, but disaster struck when Oracle cached the plan and began using it indiscriminately.

```
   Rows  Row Source Operation
-------  ---------------------------------------------------
      0  SORT ORDER BY (cr=9 r=0 w=0 time=3358 us)
      0   HASH JOIN (cr=9 r=0 w=0 time=3326 us)
      0    MERGE JOIN CARTESIAN (cr=9 r=0 w=0 time=2671 us)
      0     HASH JOIN (cr=9 r=0 w=0 time=2669 us)
      0      HASH JOIN (cr=9 r=0 w=0 time=2130 us)
      4       TABLE ACCESS FULL PC (cr=3 r=0 w=0 time=161 us)
      0       TABLE ACCESS FULL A (cr=6 r=0 w=0 time=305 us)
      0      TABLE ACCESS FULL PA
      0     BUFFER SORT
      0      TABLE ACCESS FULL P
      0    VIEW
      0     SORT GROUP BY
      0      HASH JOIN
      0       HASH JOIN
      0        TABLE ACCESS FULL PG
      0         TABLE ACCESS FULL P
      0        TABLE ACCESS FULL P
      0       TABLE ACCESS FULL PA
```

Here is the report for the now-cached execution plan but a more representative set of values of the bind variables. Just four rows were retrieved from table "A," but the Cartesian join operation produced a data set containing 1.65 million rows and the query needed 84.37 seconds of processing time.

```
   Rows  Row Source Operation
-------  ---------------------------------------------------
    208  SORT ORDER BY (cr=326 r=18450 w=18450 time=84376785 us)
    208   HASH JOIN (cr=326 r=18450 w=18450 time=84375997 us)
1659339    MERGE JOIN CARTESIAN (cr=158 r=0 w=0 time=1398500 us)
   1359     HASH JOIN (cr=146 r=0 w=0 time=28895 us)
      4      HASH JOIN (cr=9 r=0 w=0 time=1724 us)
      4       TABLE ACCESS FULL PC (cr=3 r=0 w=0 time=143 us)
      4       TABLE ACCESS FULL A (cr=6 r=0 w=0 time=286 us)
   1359      TABLE ACCESS FULL PA (cr=137 r=0 w=0 time=19562 us)
1659339     BUFFER SORT (cr=12 r=0 w=0 time=895771 us)
   1221      TABLE ACCESS FULL P (cr=12 r=0 w=0 time=405 us)
     52    VIEW (cr=168 r=0 w=0 time=35754 us)
     52     SORT GROUP BY (cr=168 r=0 w=0 time=35729 us)
   2248      HASH JOIN (cr=168 r=0 w=0 time=33368 us)
     52       HASH JOIN (cr=31 r=0 w=0 time=6874 us)
     52        TABLE ACCESS FULL PG (cr=19 r=0 w=0 time=845 us)
      1         TABLE ACCESS FULL P (cr=12 r=0 w=0 time=354 us)
   1221        TABLE ACCESS FULL P (cr=12 r=0 w=0 time=342 us)
  27556       TABLE ACCESS FULL PA (cr=137 r=0 w=0 time=8419 us)
```

Regenerating the data statistics did not change them (because the data was static) but had the side effect of invalidating the current execution plan and forcing a new one to be generated.[3] This time, Oracle did not use the Cartesian algorithm

[3] The default value of the NO_INVALIDATE setting used by the GATHER_TABLE_STATS routine is FALSE.

SQL Challenge

The last SQL challenge was to criticize the following script on as many grounds as possible—the data table in question contained approximately 100 million rows and each statement updates approximately five million rows.

```
UPDATE  /*+ PARALLEL  (promotion,  8) */ promotion p
   SET  promotion_id = 2412
 WHERE  promotion_id = 2182;
UPDATE  /*+ PARALLEL  (promotion,  8) */ promotion p
   SET  promotion_id = 2414
 WHERE  promotion_id = 2184;
UPDATE  /*+ PARALLEL  (promotion,  8) */ promotion p
   SET  promotion_id = 2416
 WHERE  promotion_id = 2188;
```

Chris Lawson wins an iPod Shuffle and a CD of Man of La Mancha for his detailed response.

"Although at first glance the SQL statements seem acceptable, this is not the case. There are serious problems that need to be addressed.

Syntax

- *Parallel DML must be enabled using 'ALTER SESSION ENABLE PARALLEL DML.'*
- *As written, the parallel hint will do nothing, since the table alias must be specified, not the table name. The hint should be /*+PARALLEL (p, 8)*/*

Performance

- *Parallel DML only provides limited speedup. That is, parallel DML does not scale nearly as well as parallel query. My tests show at best a 2.5x performance gain using parallel DML.*
- *Running separate threads can be far faster, and should at least be considered. For instance, if there is an index on promotion_id, we could run 15 threads, each of which updates a subset of the rows. Multi-threading is a frequent tactic in data manipulation in VLDB systems.*
- *If we stay with the parallelism/full-scan method, running three successive transactions (each of which performs a full table scan) is inefficient. It would be far more efficient to handle all three cases in a single scan, like this:*

```
   SET promotion_id = DECODE(promotion_id, 2182, 2412,
2184, 2414, 2188, 2416)
 WHERE  promotion_id IN (2182, 2184, 2188);
```

- *I don't see any disabling of foreign keys or indexes. When updating a huge number of rows, this should at least be considered.*
- *I likewise don't see any disabling of triggers. This should also be considered.*

Data Security

- *Changes in production tables should generally be preceded by a backup of the table(s) in question. I can't imagine updating millions of rows in a production database without a solid backup.*
- *Should we make an error, we need a back-out plan. This means having a script ready beforehand that will return the table to the correct state."* ▲

and performance returned to normal—only a few hundredths of a second were required each time.[4]

```
   Rows  Row Source Operation
-------  ---------------------------------------------------
    208  SORT ORDER BY (cr=326 r=0 w=0 time=56219 us)
    208   HASH JOIN (cr=326 r=0 w=0 time=55707 us)
    208    HASH JOIN (cr=314 r=0 w=0 time=53079 us)
   1359     HASH JOIN (cr=146 r=0 w=0 time=17952 us)
      4      HASH JOIN (cr=9 r=0 w=0 time=1808 us)
      4       TABLE ACCESS FULL PC(cr=3 r=0 w=0 time=127 us)
      4       TABLE ACCESS FULL A (cr=6 r=0 w=0 time=293 us)
   1359      TABLE ACCESS FULL PA (cr=137 r=0 w=0 time=13344 us)
     52     VIEW (cr=168 r=0 w=0 time=31360 us)
     52      SORT GROUP BY (cr=168 r=0 w=0 time=31334 us)
   2248       HASH JOIN (cr=168 r=0 w=0 time=29040 us)
     52        HASH JOIN (cr=31 r=0 w=0 time=4006 us)
     52         TABLE ACCESS FULL PG (cr=19 r=0 w=0 time=713 us)
      1          TABLE ACCESS FULL P (cr=12 r=0 w=0 time=305 us)
   1221         TABLE ACCESS FULL P (cr=12 r=0 w=0 time=353 us)
  27556        TABLE ACCESS FULL PA(cr=137 r=0 w=0 time=8269 us)
   1221    TABLE ACCESS FULL P (cr=12 r=0 w=0 time=456 us)
```

[4] The complete absence of index lookups and nested loop joins was caused by the absence of indexes, primary keys, and foreign keys. Their absence is a terrible design mistake of course, but the tables were small enough that adding indexes did not improve query performance.

Workarounds

Here are some workarounds to use if you encounter the problem—you pays your money and you takes your pick.

- Freeze the execution plan using a "stored outline." A stored outline is a comprehensive set of hints generated by Oracle and stored in the database. These hints will be automatically used by Oracle every time the query plan is regenerated and this ensures that the same query plan is generated each time.
- Embed your own hints in the query to force the use of preferred access paths. Common hints are ORDERED, LEADING, and INDEX.
- Don't use bind variables in the problem query and force the query optimizer to perform query optimization if the query parameters change.
- Disable bind variable peeking globally by changing "_optim_peek_user_binds" to FALSE. This will force the query optimizer to revert to Oracle 8*i* behavior. ▲

Iggy Fernandez is a database administrator for Verizon and can be reached at **iggy_fernandez@hotmail.com**.

Taking Control with CMDB

by Neeraj Nayan

Neeraj Nayan

Introduction

The IT Infrastructure Library (ITIL) is the gold standard for IT organizations all over the world. It provides guidance for all aspects of IT operations from Application Management and Service Level Management to IT Continuity Management (a.k.a. Disaster Recovery). The Configuration Management Database (CMDB) described in the IT Infrastructure Library is the linchpin of an effective IT organization.

History

ITIL is a set of publications providing detailed guidance (what to do, why you should do it, and how to do it) on the management of IT services. It was created in the late 1980s by the Office of Government Commerce (OGC) in the U.K. and quickly became the international standard for IT management in public as well as private-sector organizations on both sides of the Atlantic and all over the world. It is championed by large IT service providers such as IBM Global Services and EDS, software vendors such as BMC and Computer Associates, and Fortune 100 companies such as Procter & Gamble, Caterpillar, and Boeing.

"The Configuration Management Database (CMDB) described in the IT Infrastructure Library (ITIL) is the linchpin of an effective IT organization."

Benefits

- ITIL is a framework of best practices—a synthesis of ideas from international practitioners, not an academic theory of how things should work or a vendor's opinion on how to operate its products. It describes a proven and practical framework for the delivery and support of IT services.
- It provides nonproprietary and impartial guidance that is independent of the hardware and software being used.
- It is in the public domain and can be used without payment of any license fees.
- A global network of user groups provides peer support.
- It is strongly aligned with the ISO 9001:2000 quality standard and is compatible with the organization's quality management practices.

Structure

ITIL is divided into five main areas: Service Delivery, Service Support, Application Management, Infrastructure Management, and Security Management. Service Delivery covers Service Level Management, Financial Management, Availability Management, Capacity Management, and IT Continuity Management, while Service Support covers Incident Management, Problem Management, Configuration Management, Change Management, and Release Management.

Service Delivery[1]

Service Delivery covers Service Level Management, Financial Management, Availability Management, Capacity Management, and IT Continuity Management.

The goal of Service Level Management is to "*maintain and improve IT Service quality, through a constant cycle of agreeing, monitoring and reporting upon IT Service achievements and instigation of actions to eradicate poor service—in line with business or cost justification.*"

The goal of Financial Management is to "*provide cost-effective stewardship of the IT assets and resources used in providing IT Services.*"

The goal of Availability Management is to "*understand the Availability requirements of the business and plan, measure, monitor and continuously improve the Availability of the IT Infrastructure, services and supporting organization to ensure that these requirements are met consistently.*"

The goal of Capacity Management is to "*ensure that cost-justifiable IT capacity always exists and that it is matched to the current and future needs of the business.*"

[1] All the definitions in this section and the next are those found in the official OGC publications.

The goal of IT Continuity Management is to "*support the overall Business Continuity Management process by ensuring that the required IT technical and service facilities (including computer systems, networks, applications, technical support and Service Desk) can be recovered within required, and agreed, business timescales.*"

Service Support

Service Support covers Incident Management, Problem Management, Configuration Management, Change Management, and Release Management.

The goal of Incident Management is to "*restore normal service operation as quickly as possible and minimize the adverse impact on business operations, thus ensuring that the best possible levels of service quality and availability are maintained.*"

The goal of Problem Management is to "*minimize the adverse impact of Incidents and Problems on the business that are caused by errors within the IT infrastructure, and to prevent recurrence of Incidents related to these errors.*"

The goal of Configuration Management is to "provide accurate information on configurations and their documentation to support all the other Service Management processes."

The goal of Configuration Management is to "*provide accurate information on configurations and their documentation to support all the other Service Management processes.*"

The goal of Change Management is to "*ensure that standardized methods and procedures are used for efficient and prompt handling of all Changes, in order to minimize the impact of Change related incidents upon service quality, and consequently to improve the day-to-day operations of the organization.*"

The goal of Release Management is to "*design and implement efficient procedures for the distribution and installation of Changes to IT Systems.*"

Taking Control with CMDB

The Configuration Management Database (CMDB) is the linchpin of an ITIL-compliant organization. It stores information about the following:

- IT infrastructure components—information regarding servers, storage, desktops, and so on, as well as relevant information such as nature and configuration.
- Services—all information about services and how they are mapped to business processes and underlying applications, relationships, and configuration.
- Relationships—the map showing the link between services and the infrastructure components that they depend upon.

The CMDB is vital to gaining control over and staying in control of the database infrastructure and can serve many purposes including documentation, change management, governance, compliance, standardization, auditing, problem solving, and planning.

Getting Started with CMDB

Many software vendors offer tools to create and populate the CMDB; open source software is also available. Oracle's Remote Diagnostic Agent (RDA) can also be used to quickly and painlessly implement a CMDB of sorts that provides immediate and real value to Oracle database administrators and system adminstrators alike.

RDA was created by Oracle Support to aid in the diagnosis of database problems. It consists of Perl scripts that can be run on database servers or application servers to collect all the pertinent configuration information about the OS, network, and databases, as well as performance information (for the OS as well as the database) and log files. The collected information is stored in text files that could be saved in a CVS repository, but HTML files are also created for easy viewing (an example is shown in Figure 1). The entire discovery and publishing process can be easily automated, and all the information can be tied together using a simple spreadsheet of database names and URLs. You could also customize and extend the RDA code if you like.

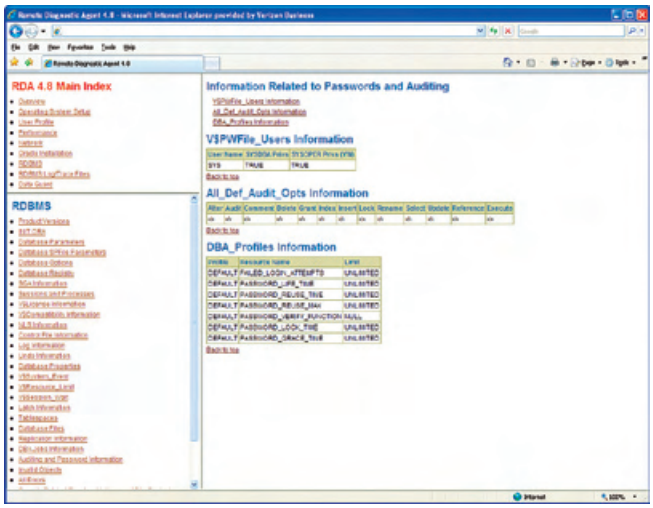


Figure 1: Sample RDA output.

Further Reading and Online Resources

The official OGC publications are the bible of ITIL practitioners but are fairly expensive. *Foundations of IT Service Management: based on ITIL* from Van Haren Publishing is an excellent and reasonably priced introduction to ITIL. The IT Service Management Forum (**www.itsmf.org**) is an international organization of ITIL practitioners with chapters in many countries. ▲

Neeraj Nayan has more than ten years of experience in the IT industry. He holds a Manager's Certificate in IT Service Management and is an IT service manager for Colt Telecom Group, a European provider of data, voice, and managed services. You can reach him at **neerajnayan@gmail.com**.

Streaming Performance!

by Chris Lawson

Row, row, row your boat/Gently down the stream
Merrily, merrily, merrily, merrily/Life is but a dream.
—Campfire song


Chris Lawson

Oracle Streams is simple in principle but has lots of "moving parts." The initial setup may look easy, but don't be fooled. There are numerous pitfalls, functional as well as performance, that can easily trap the unsuspecting DBA and cause massive performance degradation. In this column we'll review the most critical factors that can wreck Streams performance. To get started, let's review the main components in Streams. (Note that our recommendations are applicable to Oracle 10*g*.)

Overview

The three Streams components are called Capture, Propagate, and Apply. The capture process is simple in concept. Using LogMiner, Streams continually reads the redo log, checks if the transaction is part of the propagation rule set, and then stores the change on a queue in memory. These steps are all very fast, meaning that capture will seldom be a performance bottleneck. The second step, propagation, is also not usually a problem, as long as the network is fast.

In contrast to the first two steps, the apply process bears the brunt of the load. The apply processes must actually perform the transaction. This typically involves at least some disk I/O, and possibly much CPU time. Streams employs a clever way to protect the apply process from overload. When the apply process falls behind, the capture process throttles itself back, temporarily stopping capture. This state is called "Paused for Flow Control." This moderating effect is a smart move, because the apply process runs badly when overwhelmed. If Streams allowed the destination queue to fill up, the apply process would be forced to "spill" the excess transactions out of the queue and then re-read them later.

Let's now investigate some of the critical performance factors.

Number of Apply Processes

On busy OLTP systems with lots of users doing lots of transactions on the source database, you will need a commensurate number of apply processes at the destination. A reasonable starting point is to use the same number of apply processes as you have CPUs at the destination site. If the extra processes remain idle most of the time, no harm is done.

The number of apply processes is controlled via a special apply parameter called Parallelism. Note that an apply parameter is not the same as an init.ora parameter. To set an apply parameter, you use the procedure Dbms_Apply_Adm.Set_Parameter.

Commit Frequency

This is the single most important area to consider when working with Streams. Here's why: outside Streams, with regular batch jobs, the commit frequency used is not catastrophic, as long as one doesn't commit too often. So, if you are running an insert of 1,000,000 rows, you can commit every 1,000 rows, every 10,000 rows, or maybe wait until the very end, as long as your undo space is adequate. We normally just want to avoid excessive commits—(especially once per row).

With Streams, however, it is critical that you commit at moderate intervals—roughly every 1,000 rows. If you don't commit at regular intervals, you'll pay a double penalty. First, your propagated data (called Logical Change Records or LCRs) may overflow the buffered queue at the destination database. Copying LCRs back and forth causes large performance delays. Second, infrequent commits means that Streams can't take advantage of multiple apply processes. This is due to a Streams restriction that a single transaction must be handled by a single process.

Commit Serialization

Using this apply parameter, Streams allows you to control the commit order in the destination database. You can preserve the same order in which transactions committed in the original database (parameter set to full) or allow variation (parameter set to none).

Allowing unrestricted commit order improves performance somewhat because the apply processes never need to wait for any earlier transactions to commit. Nevertheless, the "none" setting should only be used if you are positive that changing the commit order will not cause problems in your application.

Checkpoint Frequency

A Streams checkpoint is not the same as a database checkpoint. Instead, this refers to how often the Log Miner process stores its current processing status in the table, System.Logmnr_Restart_Ckpt$. You might think that this checkpoint

data should not affect Streams performance, but that isn't the case. This table is actively used, with Selects, Inserts, and Updates occurring very often. Further, on a busy database, each checkpoint may contain several thousand rows, with each row containing two blob fields (plus other fields, such as SCN#).

By default, Streams writes checkpoints every 10 Mb of transaction data, but this is changeable using the capture parameter _Checkpoint_Frequency. In practice, 10 Mb is much too low for busy systems and causes the checkpoint table to grow quickly. A more appropriate value is 200.

Set this checkpointing parameter using the procedure Dbms_Capture_Adm.Set_Parameter. You can list the capture parameters by querying Dba_Capture_Parameters.

Checkpoint Retention

A related parameter that should be changed is Checkpoint_Retention_Time. This parameter controls how long checkpoint data is saved. In practice, the default value of 60 (days) is excessive; a better value is just 2.

An interesting quirk about this parameter is that you set it a bit differently from the other parameters. Checkpoint_Retention is set when a capture process is first defined. If you need to change it later, use the procedure, Dbms_Capture_Adm.Alter_Capture.

Other Critical Parameters

There are a few other key parameters that need to be changed:

- **_Hash_Table_Size**: This is another apply parameter. The recommendation from Oracle support is to use 10000000.
- **Aq_Tm_Processes**: This should be set to 1. (Note that the 10.2.0.3 Upgrade Assistant incorrectly changes this to 0!)
- **Txn_Lcr_Spill_Threshold**: This determines the maximum number of LCRs in a single transaction, after which the apply queue overflows. To avoid this, set this parameter slightly larger than the estimated maximum number of LCRs in any single transaction.

Note that you can check your current Streams parameters by querying the views Dba_Apply_Parameters and Dba_Capture_Parameters.

Monitoring Streams Performance

Finally, if you want a quick snapshot of how well Streams is working, Oracle provides an easy way. Here's my favorite Streams script for measuring the end-to-end response time. In the (real) example below, Streams captured, propagated, and applied the most recent transaction in 1 second.

```
COLUMN  APPLY_PROC FORMAT A12
COLUMN  LAT_SEC FORMAT 999999999
COLUMN  'Message Creation' FORMAT A17
COLUMN  'Apply Time' FORMAT A17
COLUMN  MSG_NO FORMAT 9999999999999

  SELECT APPLY_NAME APPLY_PROC,
(HWM_TIME-HWM_MESSAGE_CREATE_TIME)*86400 LAT_SEC,
TO_CHAR(HWM_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
"Message Creation",
TO_CHAR(HWM_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
HWM_MESSAGE_NUMBER MSG_NO
FROM V$STREAMS_APPLY_COORDINATOR;

APPLY_PROC  Lat Sec  Message   Creation  Apply Time              MSG_NO
----------- -------- ----------------- ----------------- -------------
APP_PRD           1  10:39:43  04/25/07  10:39:44 04/25/07  3316562531369
```

Concluding Remarks

Streams can be a great way to propagate information, but there are many complications to consider. In concept, Streams is simple, but efficient implementation takes great care. With a little foresight, and by following these basic suggestions, you can avoid the most serious pitfalls. ▲

Chris Lawson is an Oracle DBA consultant in the San Francisco Bay Area, where he specializes in performance tuning of data warehouse and financial applications. He is a frequent speaker at NoCOUG, and has written for a number of publications such as Oracle Internals, Exploring Oracle, SELECT, Oracle Informant, *and* Intelligent Enterprise. *Chris has held a variety of positions in the IT field—ranging from systems engineer to department manager—and is an instructor for the University of Phoenix. He can be contacted via* **www.oraclemagician.com**.

Many Thanks to Our Sponsors

NoCOUG would like to acknowledge and thank our generous sponsors for their contributions. Without this sponsorship, it would not be possible to present regular events while offering low-cost memberships. If your company is able to offer sponsorship at any level, please contact NoCOUG's president, Lisa Loper, at **lloper@dbspecialists.com**. ▲

Long-term event sponsorship:

- LOCKHEED MARTIN
- CHEVRON
- ORACLE CORP.
- PG&E

Thank you! Year 2007 Gold Vendors:

- BEZ Systems
- Confio Software
- Database Specialists, Inc.
- Embarcadero Technologies
- IT Convergence
- Network Appliance
- Princeton Softech
- Quest Software
- Roundstone Systems

For information about our Gold Vendor Program, contact the NoCOUG vendor coordinator via email at: **vendor_coordinator@nocoug.org**.

TREASURER'S REPORT

Jen Hong, *Treasurer*

Beginning Balance		
April 1, 2007		**$ 49,556.67**
Revenue		
Membership Dues	3,750.00	
Meeting Fees	420.00	
Vendor Receipts	10,300.00	
Miscellaneous	1,662.00	
Interest	59.60	
Total Revenue		**$ 16,191.60**
Expenses		
Regional Meeting	3,974.23	
Journal	5,628.26	
Membership	–	
Administration	–	
Website	350.00	
Board Meeting	722.61	
Marketing	121.39	
Insurance	–	
Vendors	–	
P.O. Box	–	
IOUG-Rep	650.00	
Total Expenses		**$ 11,446.49**
Ending Balance		
June 30, 2007		**$ 54,301.78**

Bag o' Tricks

by Danny Chow

Danny Chow

Database performance is impacted by various activities happening within the database, and one of these factors is the amount of data being changed in a given period of time. Data changes generate log switches and, therefore, log switch counts are a good indicator of the level of change activity in the database. It is very important to point out that redo volume and data change volume are related, though not by a fixed ratio—the details are too lengthy for this short article. The important point is to establish a redo log switch pattern in order to identify abnormalities. With that said, redo log volume is important when determining how fast archived logs are filling up disk space or being shipped and applied to standby databases, etc.

I once worked with a database into which we were loading large amounts of data in batches. Each batch was taking longer to complete than the previous batch, even though the data volume was the same in each batch. By comparing the number of redo log switches between batches, I found out that the number of log switches had increased significantly. It turned out that after each run some timestamp and status fields were updated, causing multiple deletes and re-inserts of the same records in lieu of updates in later runs.

Another time, I was asked to set up a weekly load job at 3 a.m. on Sunday morning, since conventional wisdom said that it would be a "quiet time" for the database. On collecting the log switch counts at regular intervals, we realized that everyone has been using this "quiet" time, and the period with least data change was actually 6 p.m. to 10 p.m. on Monday evenings!

I always schedule a query to collect redo log switch counts and insert these results to a table or a spreadsheet to establish a pattern. The query listed below uses the V$LOG_HISTORY view to accommodate databases running with NOARCHIVELOG mode.

```
redo_log_activity.sql
-- Script to count redo log switches
-- This is based on the first_time of each log;
this may not be accurate for an idle system
-- Verified to work with Oracle 8i, 9i, and 10g

break on report
compute sum label Total of CNT on report
column cnt heading 'Log Switches' format 99999
column bucket heading 'Interval' format A16

PROMPT
PROMPT Number of log switches within the interval
specified.
--
-- This query prints one line per bucket.
--
SELECT (CASE
  WHEN first_time between (sysdate - 1/96) and sysdate
       THEN '1: last 15 min'
  WHEN first_time between (sysdate - 1/48) and sysdate
       THEN '2: last 30 min'
  WHEN first_time between (sysdate - 1/24) and sysdate
       THEN '3: last 1 hour'
  WHEN first_time between (sysdate - 4/24) and sysdate
       THEN '4: last 4 hours'
  WHEN first_time between (sysdate - 1) and sysdate
       THEN '5: past 1 day'
  WHEN first_time between (sysdate - 3) and sysdate
       THEN '6: past 3 days'
  WHEN first_time between (sysdate - 7) and sysdate
       THEN '7: past 7 days'
       END) as bucket,
       count(*) as cnt
    FROM  v$log_history
   WHERE  first_time >= (sysdate - 7)
   GROUP  BY (CASE
  WHEN first_time between (sysdate - 1/96) and sysdate
       THEN '1: last 15 min'
  WHEN first_time between (sysdate - 1/48) and sysdate
       THEN '2: last 30 min'
  WHEN first_time between (sysdate - 1/24) and sysdate
       THEN '3: last 1 hour'
  WHEN first_time between (sysdate - 4/24) and sysdate
       THEN '4: last 4 hours'
  WHEN first_time between (sysdate - 1) and sysdate
       THEN '5: past 1 day'
  WHEN first_time between (sysdate - 3) and sysdate
       THEN '6: past 3 days'
  WHEN first_time between (sysdate - 7) and sysdate
       THEN '7: past 7 days'
       END)
ORDER BY BUCKET;
```

In the example listed below, a database was generating 21 log switches per hour versus the normal rate of 4 log switches per hour that had been previously recorded, and the abnormal pattern is what raised the red flag. I generally find the "one hour" figures most useful; however, feel free to tailor the interval to fit your need.

```
SQL> @redo_log_activity.sql

Number of log switches within the interval
specified.

Interval          Log Switches
----------------- -------------
1: last 15 min               9
2: last 30 min               9
3: last 1 hour              21
4: last 4 hours            122
5: past 1 day               48
6: past 3 days               1
                  -------------
Total                      210
```

Danny Chow is an independent Oracle consultant with many years of DBA and development experience, and holds certifications in Oracle 7, 8, 8i, and 9i. His email address is **dannychow@earthlink.net**.

NoCOUG Summer Conference

Session Descriptions

For more detailed descriptions and up-to-date information, see **www.nocoug.org**.

—General Session—

Oracle Database 11*g* Spotlight: Managing Change

Leng Tan, *Oracle Corporation* 9:30–10:30

Leng Tan will discuss the manageability features in Oracle Database 11*g*, which are designed to help organizations easily manage enterprise grids and deliver on their users' service level expectations. Oracle Database 11*g* introduces more self-management and automation that will help businesses reduce their system management costs while increasing the performance, scalability, availability, and security of their database applications. During this keynote presentation, Leng will detail many of the new manageability capabilities in Oracle Database 11*g*, such as automatic SQL and memory tuning, partitioning advisor for tables and indexes, and enhanced performance diagnostics for database clusters. Other highlights of this session include an overview of Oracle Database 11*g*'s unique new "change assurance" features, designed to help businesses significantly reduce the time, cost, and risk associated with system changes such as hardware or software upgrades as well as configuration changes.

*Leng Tan, Oracle vice president of database manageability and diagnosibility, has worked in the database development organization at Oracle Corporation and is a 10+ year veteran. She is in charge of development of mission-critical database capabilities including manageability and diagnostics. Leng recently hosted a podcast on Oracle Database 11*g *Change Assurance and Manageability.*

—Keynote—

Relational Database Industry and Technology Trends

Guy Harrison, *Quest Software* 11:00–12:00

RDBMS technology and the RDBMS market have undergone huge changes in past years with more change to come. What are the key industry and technology trends influencing the future directions for relational databases, and how do these trends affect RDBMS professionals? In particular, we'll examine the effect of grid technology, virtualization, changing software architectures, open source, outsourcing, and security considerations.

Guy Harrison is chief architect, database solutions, at Quest Software, Inc. He is a recognized database expert with over 15 years of experience in database administration, development, and performance tuning. Guy is the author of Oracle SQL High-Performance Tuning *(Prentice-Hall, 1997, 2000),* Oracle Desk Reference *(Prentice-Hall, 2000) and* MySQL Stored Procedure Programming *(O'Reilly, 2006) and is a regular contributor to database-related technical journals and a regular speaker at technical conferences. Guy was instrumental in creating Quest's popular Spotlight® product line and contributed to the development of other Quest products such as Quest Central and Foglight.*

—Room 1220—

Oracle Database 10g: Implement Streams

Daniel Liu, *Oracle Corporation* 1:00–2:00

This presentation provides an overview of Oracle Database 10*g* Streams technology. It offers an introduction to the basic concepts and architectures of Oracle Streams and discusses the following implementation topics: Basic Objects, Rules, Capture, Instantiation, Propagation, Apply, and administrating the Oracle Streams environment. This session also provides a step-by-step setup of an Oracle Streams environment and compares Streams technology with Data Guard and Advanced Replication.

Daniel Liu is a senior technical manager at First American Real Estate Solutions in Santa Ana, Calif., and co-author of Oracle Database 10g New Features: Oracle10g Reference for Advanced Tuning & Administration *by Rampant TechPress. His expertise includes Oracle database administration, performance tuning, Oracle networking, and Oracle Application Server. Daniel published articles with* DBAzine, Oracle Internals, Oracle Technology Network, *and* SELECT Journal. *He received the SELECT Editorial Award for Best Article in 2001 and was named Architect of the Week by the Oracle Technology Network in 2004. He has also given presentations at IOUG-A Live, LAOUG, OCOUG, NoCOUG, TOUG, OOUG, Wilshire Meta-Data conference, and Oracle OpenWorld. Daniel has served as a panelist on Oracles of Oracle at Oracle World and IOUG-Live.*

Back Porting ADDM, AWR, ASH, and Metrics to Oracle 9*i* and Oracle 8*i*

John Kanagaraj, *DB Soft* 2:30–3:30

Oracle Database 10*g* introduced a number of "automatic" features such as ADDM (Automatic Database Diagnostic Monitor), AWR (Automatic Workload Repository), ASH (Active Session History), and server-generated alerts (SGAs) that are based on server metrics. These features are actually based on internal infrastructure components consisting of V$ and X$ views that existed even in Oracle 8*i*. Once we have understood how Oracle Database 10*g* uses such components in these "automatic" features, it is a simple matter to "back port" much of this functionality to the older versions. In this presentation, we will see some practical examples of this back porting and point to other areas that can benefit from it. We will also list what features can't be "back ported" and why.

John Kanagaraj is a principal consultant with DBSoft Inc., and resides in the Bay Area in sunny California. Since 1984 he has been working with various flavors of Unix and Oracle as a developer, DBA, and system administrator. John is a frequent presenter at IOUG and OAUG, and co-authored Oracle Database 10*g*: Insider Solutions, *published by SAMS. As the executive editor of IOUG's* SELECT Journal, *John is always looking forward to developing and mentoring new authors. You can e-mail him at* **johnk@ioug.org**.

Simulating Oracle I/O Workloads to Accurately Configure Storage

James Koopmann, *LSI Corporation* 4:00–5:00

Using Oracle's ORION workload tool, we can effectively develop a workload that mimics and stresses a storage array in the same manner as a planned application with an Oracle backend database. Only after benchmarking potential storage can we gain a finer understanding for the performance capabilities of our storage and be confident in its ultimate deployment. This presentation discusses storage concepts such as RAID level, Host Bus Adapters, Controllers, and Oracle's ORION workload tool. Then the attendee will be walked through a variety of benchmark tests and results that show how to run the ORION tool and make decisions on how to configure storage for an Oracle database. This presentation will show the results of various workloads using RAID 0, 1, 10, and 5 on a 30 TB storage system. So come see how your favorite RAID level performed.

James F. Koopmann is dedicated to providing technical advantage and guidance to companies within information technology. Over the years James has worked with a variety of database-centric software and tools vendors as a strategist, architect, DBA, and performance expert. He is an accomplished author with technical writings appearing in publications such as Oracle Magazine, Oracle Professional, *and* Exploring Oracle, *and across the Web in many portals and newsletters.*

—Room 1240—

Systematic Oracle Performance Tuning

Guy Harrison, *Quest Software* 1:00–2:00

Oracle performance tuning has matured over the past decade with ad-hoc and inaccurate tuning approaches giving way to more empirical and effective profiling approaches. However, there are still many circumstances in which performance tuning addresses symptoms rather than the underlying causes of poor performance. In this presentation, a systematic approach is offered that focuses on systematically optimizing the Oracle RDBMS to avoid inefficient or needlessly costly tuning outcomes.

Guy Harrison is chief architect, database solutions, at Quest Software, Inc. He is a recognized database expert with over 15 years of experience in database administration, development, and performance tuning. Guy is the author of Oracle SQL High-Performance Tuning *(Prentice-Hall, 1997, 2000),* Oracle Desk Reference *(Prentice-Hall, 2000), and* MySQL Stored Procedure Programming *(O'Reilly, 2006) and is a regular contributor to database-related technical journals and a regular speaker at technical conferences. Guy was instrumental in creating Quest's popular Spotlight® product line and contributed to the development of other Quest products such as Quest Central and Foglight.*

Introduction to Oracle Database 11*g*

Penny Avril, *Oracle Corporation* 2:30–3:30

The year 2007 marks 30 years of Oracle helping businesses and governments manage their most valuable asset—information. It's the lifeblood of all our applications, from traditional transactional systems and large data warehouses to Web 2.0, XML, and other applications. During this presentation, Penny Avril will discuss today's IT challenges and how Oracle Database 11*g* can help manage users' service-level objectives while reducing the cost of computing.

Penny Avril is principal product manager in Oracle's Server Technology Division, where she is currently responsible for the deployment and R&D planning of database releases. Penny is a frequent presenter at regional user conferences and other industry-related events.

Average Active Sessions

Kyle Hailey, *Performance Vision* 4:00–5:00

Average Active Sessions, the golden mean of performance metrics, is a simple, powerful, and elegant way of displaying the multidimensional data found in Active Session History (ASH). Between the dense approximate samples of ASH and the broad accurate snapshots of STATSPACK, we find a simple metric that can easily determine database health. In this presentation find out how to determine Average Active Sessions from STATSPACK, AWR, or ASH, and how to use the metric with Enterprise Manager or by hand by mining ASH data.

Kyle Hailey has been in the industry for almost two decades, 10 years of which were spent at Oracle in support, porting, benchmarking, and kernel development. He has created tools to improve high-end performance monitoring such as direct SGA attach and interactive graphic displays of performance data. He recently redesigned the Oracle Enterprise Manager 10g performance pages to be graphically oriented and wait interface–centric. He now works as an independent consultant on Oracle performance issues and systems monitoring.

NoCOUG member Alex Wong smiles broadly after winning three prizes in the giveaway raffle at the Spring Conference. He must have been wearing his lucky shirt! Bring your business card with you to the Summer Conference and enter to win a full conference pass for Oracle OpenWorld 2007 worth almost two thousand dollars!

—Room 1130—

RMAN in the Trenches: Part II

Philip Rice, *UC Santa Cruz* 1:00–2:00

RMAN in the Trenches is a sharing of discoveries, as a result of implementing RMAN over a period of several years. It is not a high-level overview of features and it is not a how-to book. Intermediate topics are discussed along with some basics. Beginners can benefit by having a better idea of what to watch out for. Topics include corruption detection, metadata, flashback, and hardware implications. We'll also have a sampling of the good, the bad, and the ugly.

Philip Rice has been in the computer field since 1980 and began working with Oracle in 1991. He is now an Oracle database administrator at UC Santa Cruz, retaining a detectable Minnesota accent almost 8 years after moving from the land of long winters.

Tracing Individual Users in Connection-Pooled Environments with Oracle 10*g*

Editor's Pick

Terry Sutton, *Database Specialists.* 2:30–3:30

The SQL trace facility helps the DBA diagnose performance problems and make applications run faster. Tracing works well in client/server environments or two-tier architectures where there is a one-to-one correspondence between a database session and an end-user session. Unfortunately, the SQL trace facility never worked well in connection-pooled environments or architectures that use an application server to multiplex many end-user sessions into a smaller number of database sessions. Beginning with Oracle 10*g* we have the ability to trace an end user's session, even though connection pooling may cause each request by that user to be served by a different Oracle server process. In this presentation we will walk through a real-life example of tracing an individual end user's actions as they use a web-based application. Although each request by the end user might be handled by a different Oracle server process, and each Oracle server process will also handle requests from other end users, all requests by our target end user get traced and reported cohesively in one TKPROF report.

Terry Sutton, OCP, has been an Oracle DBA for 12 years and has worked in the information technology area for 19 years. Since 2000, Terry has been a senior staff consultant at Database Specialists, performing assignments ranging from production database administration to emergency troubleshooting, with a particular focus on Oracle database performance tuning. He has been a speaker at the RMOUG, NoCOUG, and IOUG-Live conferences, as well as the Hotsos Symposium and various local user groups.

SOA Project Methodology

Dr. Mohamad Afshar, *Oracle Corporation* 4:00–5:00

To successfully plan and execute an SOA project requires a methodology with SOA-specific techniques to complement your standard software delivery practices. This session highlights the unique aspects of a methodology for implementing SOA projects, including service portfolio planning, service design, and business process analysis. Learn how to maximize the benefits from your SOA projects with these techniques!

Dr Mohamad Afshar is a recognized expert in the field of SOA. His work on Oracle's SOA Maturity Model and Governance with SOA are widely recognized. Mohamad is currently director of product management for Fusion Middleware and has been leading the efforts on Oracle's SOA Methodology. He has a bachelor's degree from University of London and a PhD from Cambridge University. ▲

Usually, it's harder to pinpoint.

Amazing what you can accomplish once you have the information you need.

When the source of a database-driven application slowdown isn't immediately obvious, try a tool that can get you up to speed. One that pinpoints database bottlenecks and calculates application wait time ***at each step***. Confio lets you unravel slowdowns at the database level with no installed agents. And solving problems where they exist costs a ***tenth*** of working around it by adding new server CPU's. Now that's a vision that can take you places.

A smarter solution makes everyone look brilliant.

CONFIO SOFTWARE

Download your FREE trial of Confio Ignite™ at www.confio.com.

Oracle | DB2 | SQL Server | Sybase | J2EE

NoCOUG Summer Conference Schedule

August 16, 2007, at Chevron, San Ramon, CA

Please visit **www.nocoug.org** for updates and directions, and to submit your RSVP.
Cost: $40 admission fee for nonmembers. Members free. Includes lunch voucher.

8:00 a.m.–9:00	**Registration and Continental Breakfast**—Refreshments served
9:00–9:30	**Welcome**—Lisa Loper, NoCOUG president
9:30–10:30	**General Session:** *Oracle Database 11g Spotlight: Managing Change* —Leng Tan, Oracle Corporation
10:30–11:00	**Break**
11:00–12:00	**Keynote:** *Relational Database Industry and Technology Trends* —Guy Harrison, Quest Software
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #1**
	Room 1220: *Oracle Database 10g: Implement Streams*—Daniel Liu, Oracle Corporation
	Room 1240: *Systematic Oracle Performance Tuning*—Guy Harrison, Quest Software
	Room 1130: *RMAN in the Trenches: Part II*—Philip Rice, UC Santa Cruz
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #2**
	Room 1220: *Back Porting ADDM, AWR, ASH, and Metrics to Oracle 9i and Oracle 8i* —John Kanagaraj, DB Soft
	Room 1240: *Introduction to Oracle Database 11g*—Penny Avril, Oracle Corporation
	Room 1130: *Tracing Individual Users in Connection-Pooled Environments with Oracle 10g* —Terry Sutton, Database Specialists
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #3**
	Room 1220: *Simulating Oracle I/O Workloads to Accurately Configure Storage*—James Koopmann, LSI Corporation
	Room 1240: *Average Active Sessions*—Kyle Hailey, Performance Vision
	Room 1130: *SOA Project Methodology*—Dr. Mohamad Afshar, Oracle Corporation
5:00–??	**NoCOUG Networking and Happy Hour at Marriott San Ramon, 2600 Bishop Drive, San Ramon**

Oracle Performance Training Day—details on page 5.

Session descriptions appear on page 25.

RSVP online at www.nocoug.org/rsvp.html

The NoCOUG Journal design and production: Giraffex Inc. (www.giraffex.com).

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED

Win $2,000 in training—details on page 18.