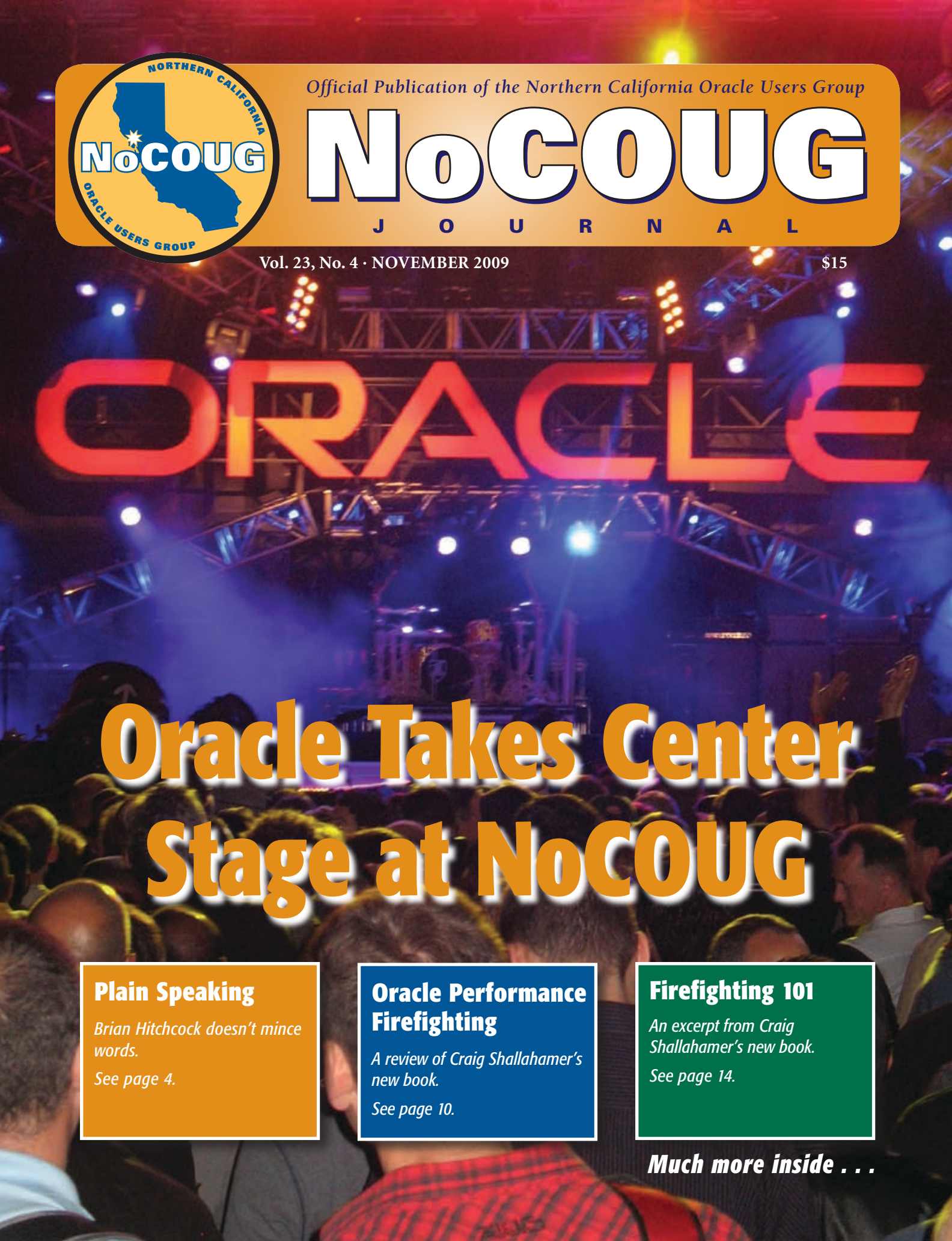
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 23, No. 4 · NOVEMBER 2009 $15



Oracle Takes Center Stage at NoCOUG

Plain Speaking

Brian Hitchcock doesn't mince words.

See page 4.

Oracle Performance Firefighting

A review of Craig Shallahamer's new book.

See page 10.

Firefighting 101

An excerpt from Craig Shallahamer's new book.

See page 14.

Much more inside . . .

Oracle Takes Center Stage at NoCOUG

NoCOUG's mission is to showcase the Oracle Database, and the *NoCOUG Journal* is a big part of that. In this issue, our veteran book reviewer, Brian Hitchcock, hands the baton to Dave Abercrombie and indulges in some plain speaking of his own. Rookie reviewer Dave Abercrombie does a great job of reviewing Craig Shallahamer's brand new book *Oracle Performance Firefighting*, and we top it off by printing an excerpt from the book. Norbert Debes returns with yet more Oracle secrets and John Weathington waxes eloquently about the need for a policy management system. You'll find all this and more inside.

Oracle Database 11*g* Release 2 will take center stage at the fall conference on November 13 at the Oracle conference center in Redwood Shores. Juan Loaiza will introduce Exadata V2, and a bevy of Oracle speakers will give you a glimpse into Oracle's latest achievements. It will be another great conference and I hope to see you there. ▲

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

2009 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 February 2010 issue is November 30, 2009. 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

We Have Turned a Corner

by Hanan Hit

Hanan Hit

Some of you might accuse me of being a naïve optimist when I say that a future of renewed opportunities is right around the corner. However it is my firm belief that technical excellence has always been and will continue to be the enabler of such prospects, and leading professionals will be the first to reap the fruits of economic recovery.

Most of us have felt the impact of the economic downturn in our respective organizations. In spite of the slowdown, NoCOUG has remained faithful to its mission of keeping its members on the cutting edge of technology innovations.

I want to thank all of our board members and volunteers for their dedication to helping others in the Oracle profession. Also, a special thanks to Nora Rosingana for taking care of our financial reporting.

I'm proud of what we achieved during the last year. We had four conferences with presentations by industry leaders. Our quarterly journal was packed with great content. We had our first International NoCOUG SQL Challenge, which was very successful. And last but not least, we'll be holding a special training event with Tanel Poder on advanced Oracle troubleshooting.

Our fall conference will be held on Friday, November 13 at the Oracle Conference Center in Redwood Shores. The day will be filled with top speakers giving fascinating presentations. Juan Loaiza, senior VP at Oracle, will present the amazing new Exadata V2 database machine; Kyle Hailey will tell you all about optimizer mistakes; Jeff Jacobs will discuss common SQL "anti-patterns" that cause performance problems; Tanel Poder will give presentations on capacity planning and troubleshooting tools; and Brian Hitchcock will give an introduction to MySQL. This is an excellent opportunity to meet the experts, network with industry colleagues, participate in book raffles, visit vendor exhibits, and enjoy plenty of food and refreshments. Get all of the conference details and submit your registration at the NoCOUG website, **www.nocoug.org**.

In closing, I encourage all of you to continue concentrating on new technical innovations as well as making sure that your current projects are successful. I believe the worst is behind us and I am very hopeful that in 2010 we will see a greater demand for applications and services that require our expertise.

I'll see you at the conference. ▲

Plain Speaking

An Interview with Brian Hitchcock

Brian Hitchcock needs no introduction to NoCOUG members; over the years, he's reviewed more than a dozen books for the NoCOUG Journal. *Brian's book reviews are very insightful; he uses them as an opportunity to reflect on the topic of the book and he doesn't mince words. For example, when reviewing* Database Hacker's Handbook *for the May 2006 issue of the* Journal, *Brian said: "Oracle is hard to secure because it is so big and so complex. A reasonable person would say that this argues for using simpler RDBMS software. Take MySQL for example. A simpler RDBMS to be sure, but what is the MySQL vendor doing? They're adding "features" as fast as they can, which adds to the size and complexity of their product. And why are they doing this? To meet market demands! Perhaps the enemy is us?"*

Hard to secure? The enemy is us? Them's fighting words.

Yes, we are the problem. We have created the problem and we continue to demand that the problem be made worse. Of course, as in all real stories, everyone has an agenda that is being served. The more complex Oracle becomes, the better for our job security.

We could, in a perfect world, demand that Oracle (and other vendors) stop adding features to their products and delay shipment of anything they sell until it was perfect. We could, hypothetically, demand that no software be sold to us until it was perfectly secure, performed flawlessly, and didn't need administration or support. But we don't. Why? Because all the complexity—which causes many of the security issues—is what we want and what we demand. Oracle knows what sells and that isn't always what we need.

We think we can get some sort of advantage over our competition if we just have somemore features in our business systems, and we assume that these business advantages will always outweigh the costs of actually implementing and supporting these new features in our systems. Consider the ERP system. How many stories have you heard where a business started to implement a large, complex ERP system because it would make them more profitable, and later on that same business has lost huge amounts of money due to project delays, bugs, etc.?

Because of our belief (we want to believe) that new and more complex is better, we buy shiny new stuff. Whether that stuff really works is a support detail left to those unlucky persons such as myself. Yes, it is new and it is shiny but it won't run on our servers. No problem, time for new servers! But what about the existing servers? Well, you must support all of them! No problem! And now we need to move all this into a cloud—wait, is that a private or a public cloud? Will any mobile device that our users find at Best Buy be fully interoperable with the new software? And we wonder why security is so hard.

We complain about it, but we really get what we really want—more interesting and more secure jobs. And for all the complaining, we will now spend lots of dollars traveling to the city by the bay to spend a week wallowing in all the new shiny stuff that will create lots of new problems—and maybe give our businesses some real competitive advantage.

This generates an IT system arms race—my company must keep up with our competition. Do any of our businesses really need a fully virtualized, cloud-based, mobile-enabled order-fulfillment system? I'm not sure, but we absolutely must not fall behind our competitor down the street (or in Redmond). The arms race supports itself. Instead of limiting the complexity to simplify our businesses, we seek out new and shiny and try to bolt it onto what we have. More complexity breeds more complexity, which needs more of us to support. Is everybody happy?

Actually, when you look at the big picture, we have a pretty sweet deal. We make our own mess and then try to clean it up, which creates a bigger mess . . . and we get paid all along the way.

My pointy-haired manager wants me to patch the database every quarter, but he's never heard of "attack surface area" and "surface area configuration." I want to blurt out that patching the database is like recaulking all the doors and windows but forgetting to close them. Are patches guaranteed to be free of regressions? Do I dare inject them into our mission-critical e-commerce databases without full-blown regression tests? Can we afford to patch? Can we afford not to patch? Do you patch?

First, my manager has the most wonderful hair I've ever seen. He made me say that. Really! I live in fear. He requires that we call him "Mr. Big." But his hair is just wonderful!

Let's start with the easier parts of the question. I support Oracle Applications environments and patching is a full-time job for an Oracle Applications DBA, so yes, I patch. But that doesn't really address the issue. Oracle Applications is just a more extreme case of an Oracle database environment. There are always new bugs coming to light with patches to fix them. And, there are new features that can be added by patching. So, back to the question, do you patch? Yes, but only with a really good reason. So what is a really good reason?

This is not a simple question. There are so many Oracle products with so many patches that I don't know that it is theoretically possible to be completely patched at any given point in time. And even if you were, it wouldn't last long. I think it is clear that no one is fully patched. Therefore, the real question is how do you decide what to patch and when. I think you should patch only when there is a clearly defined reason for taking the risk of patching. Examples: your system won't run without a patch—pretty easy, everyone from IT to Sales will agree you need that patch. Next, you must have a patch to have a feature that allows you to sell more stuff, again, pretty easy. How about patches because they are recommended by support? Not so easy. Again, I would want a specific reason to apply these patches. The phrase I really don't respond well to is "this patch fixes a lot of bugs." This means I am patching lots of stuff I may not be using and could break stuff I am using.

I don't see this as a good enough reason to patch. But reality plays a part in this. In the Oracle Applications world, there are many times when you have to apply a huge rollup (or family pack) patch just to fix the one bug you are hitting. You don't have a choice,

Oracle doesn't (and realistically they can't) offer all the bug fixes as individual patches. When looking at a patch I ask the following: What will I tell the business if this patch brings down production? If I have a clear answer (our system was down already, we wanted to sell more stuff . . .), then I am confident that the need to patch outweighs the risk.

Further, there is always risk in patching. Even if a patch fixes whatever the problem was that made you want to patch in the first place, you don't know, for sure, that the patch didn't mess up something else. This becomes complicated. Imagine all the products Oracle offers. We all have different combinations of those products and we all have a slightly different set of versions, release levels, patch levels, and options installed for all of those products. And they interact with the OS and the hardware and the network and who knows what else you have added over the years. How can Oracle test any patch to be sure it doesn't break something else? They can't. I'm not criticizing Oracle (full disclosure, I want to work for Oracle, anytime soon would be good!). But we have to keep in mind that Oracle can only do so much in the way of regression testing. When you patch, you take on risk. A good example of this is the dreaded "one-off" patch. The README files for these patches include a disclaimer that the patch hasn't been regression tested.

Truly bad science, but it is what we do. So, to answer another part of the question, no, patches are not guaranteed to be free of regression issues.

Which brings us to one of my favorite patching topics, CPU patching. These are bundles of patches to fix bundles of bugs (if there are gaggles of geese, what is a whole lot of bugs?) related to known and recently discovered security issues with all Oracle products.

From above, I think we need a really good reason to patch, but for CPU patches, what exactly is that good reason? Well, the obvious good reason is that we want to have a secure system. We all agree that is what we want, but do CPU patches really get us there?

If you read the CPU patching docs carefully, you will find that they don't do what you think they do. A reasonable person would assume that if you apply the quarterly CPU patches regularly and you keep up to date, your environment is secure from the known security threats as of the time of the most recent CPU patch. Not exactly. Each CPU patch represents Oracle's best effort to date to fix all of the known security issues. That doesn't mean the fix for any given security issue in a given CPU patch really fixes the problem. It may contain the best that Oracle has at the time, but more bugs are often found that get fixed in the next CPU patch. And you don't really know how Oracle decides which security fixes get into each CPU patch. Are you sure Oracle puts all the most important security patches into the latest CPU patch, or, do they put in the things they can fix before the next quarterly CPU patch release date? You *don't* know—there may well be a very serious, well-known security problem that didn't get fixed before the latest CPU patch was released. You think you are secure because you are up to date on CPU patches, but you may or may not be. Look at this from another perspective: How do you know you need any or all of the fixes in a CPU patch? Oracle is trying to fix as much as they can as fast as they can, but that doesn't mean they are fixing anything that affects you. For example, if your system is behind a firewall and does not interact with the Internet, do you need CPU patches at all? I don't know. You have to make all of these impossible decisions with imperfect data.

Given this, and given the risk of patching, how do we assess the CPU patching question? I think this is the same as any patch. You need to understand why Oracle does what it does, and you need to merge that agenda with that of your business. If your business would be harmed by press reports that you weren't up to date with all the vendor security patches, then you'd better be CPU patched all the time and learn to live with it. Always have a good reason to apply any patch. And I'm the first to say that political reasons are just fine. In all my experience, the political issues always trump the technical issues and technical persons do much better once they embrace this reality.

On to the next part of the question, can you "inject" them into a mission-critical database without full regression testing? I think you have to. You can't do full regression testing, it would take forever. And your business can't afford it. Clearly you don't "inject" patches into your production environment; you test them in dev or alpha and beta systems, etc. The problem with this is that only production has the most current set of data. Beta may be a month behind, and the data could be different enough that the same patch works in beta but not in alpha, depending on what the patch is changing. All in all, we take some level of risk to move forward. Want to be on the latest version of Oracle? Cool—then you take the risk of having the latest bugs. Want to avoid patching by staying on the old version of Oracle you have been running for years? No problem—you take on the risk of very old bugs that won't be patched because your version isn't supported anymore.

What would a reasonable person do? First, a reasonable person wouldn't become a DBA to begin with, but that is for another interview. You and your business will make some decisions. Most shops that I have been involved with went one of two ways: *all patching all the time* or *none at all unless abso-*

lutely required. Both of these approaches have their problems and benefits. I was asked if you can afford to patch, can you afford not to? My answer is "both." You can't support Oracle for very long and not apply some sort of patch somewhere. CPU patching alone requires quarterly patching at a minimum. Or, you could decide to do CPU patching once a year, but then you are exposed (in theory) to some of the known security issues for most of a year.

Finally, some examples from my experience, which may or may not have anything to do with anything.

CPU patching is complicated enough for the Oracle database, but for Oracle Applications, it is even more fun (and/or more job security, depending on how you look at it). For Oracle applications, CPU patching starts with the database but then moves on to the Oracle Application Server, the web and forms servers, various developer libraries, etc. And the best part? For Oracle Applications, the CPU patches are not cumulative. Think about that. Yes, CPU patches are cumulative for the database. No matter how far behind you are on CPU patching, you can apply the latest CPU patch to your database and be done. Not so for Oracle Applications. If you haven't applied any CPU patches ever, and you are running Oracle Applications 11.5.10.2, this means you have to go back 4 years and apply all of the CPU patches between then and now. No kidding. Once upon a time, I was asked to plan the CPU patching for such an environment, and after about a month of planning I offered this option: we are so far behind and the patch plan is so complex (300+ patches plus all their pre-req patches), it would be less effort to simply upgrade to Release 12 of Oracle Applications. And, with Release 12, the CPU patches for Oracle Applications are cumulative. Patching is a crazy business. But it pays well.

What about Oracle Database 11gR2 then? Should I upgrade this Oracle 7.3.4 database, which has been running without a hitch on Windows NT SP2 for years and years?

Many questions are posed in a way that masks their inherent assumptions. You are not running without a hitch. That is an illusion.

When you can't find any hardware that supports NT SP2 what will you do? Will that be a "hitch"? When you need to do anything to this system and you try to find people that can help you, will they be available and what will they cost? Will that be a "hitch"? There is no free lunch. Yes, you perceive that this database has been running without a hitch, but that isn't true. You have put this system into a deep freeze where time stands still, as if it were traveling to Mars. As long as everything is just perfect (not likely in the real world), yes, this system will continue to run. When something does happen, the options available to fix this system and the cost of these options will be very unpleasant. The longer you wait, the more painful the transition will be.

And by keeping this system frozen in the past, you add to the complexity of your environment. And that makes it more expensive to support, and so it goes. The logic is self fulfilling. If you wait long enough, your decision to keep this system in the past appears to make sense. You will be right in that it will be really expensive to bring this system into the present, but this is the illusion. You have to keep telling yourself that this situation is good because you need to justify your leaving this problem unresolved for so long. I get it, I really do. I support databases that really should be upgraded to a current version. When asked, the people who make these decisions say they "can't" upgrade these databases. The real reason is that they don't want to spend the money now, and that means they will spend lots more money later. Overall, this is good for me.

Thanks for taking the time. I wish I could ask you about medical insurance reform but let's not go there. But, before you go, I've just got to ask: what's with the Tiffany lamp on your Facebook and LinkedIn profiles?

That is precisely the problem with our health care system: no one wants to "go there." This ties it all up nicely. Just like the 7.3.4 Oracle database running on Windows NT SP2 mentioned before, there is a perception in our country that the health care system has been "running without a hitch." This isn't reality. And the painful changes needed to make our health care system viable for the long-term (that is, longer than the Boomers will be around) will only get more painful the longer we wait.

I don't claim to have all the answers, but I am very sure of this. I want everyone to have some level of health care because I want everyone working hard so they can pay my Social Security benefits. We are all in this together. Unfortunately, not all of us get that, and that is the real issue.

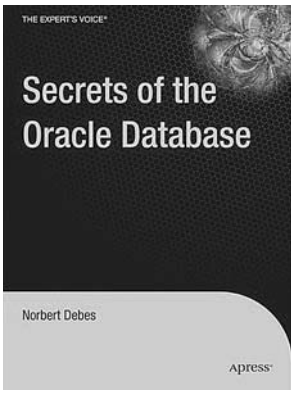
The Tiffany lamp was a seven-year project. I first took stained glass classes in high school and eventually even sold some work and did some installations. This lamp is my attempt at a Tiffany-style leaded-glass lamp. Tiffany produced many lamps using glass that his company produced to create the complex colors. This is the smaller version of the Wisteria lamp shade and is 10" in diameter. It is called the Pony Wisteria and has 873 pieces of glass, which are wrapped in strips of copper foil. The foil-wrapped pieces are placed on a mold to form the shape of the shade and soldered together. The full-size Wisteria lamp shade is 18" in diameter and has closer to 2000 pieces of glass.

An excellent book was published in 2007 discussing the woman who actually did the design work for many of Tiffany's most famous lamps, Clara Driscoll. The book is *A New Light on Tiffany, Clara Driscoll and the Tiffany Girls*. It describes the working environment in the late 1800s and relates many issues of the time that resonate today. Established workers threatened by new low-cost labor, management cost cutting, etc. Anyone interested in Tiffany lamps would find this book worthwhile.

Why did it take me seven years to complete? First, I had never done a lamp with such small, complex pieces of glass before, and I had to learn as I went. This meant I would stop for months while I figured out how to cut and grind such small pieces of glass. Similar delays occurred while I learned how to solder on a curved surface, and I have some scars where gravity and molten solder converged on my hands. Those that make Tiffany reproductions professionally could produce such a lamp in much less time, but then, this was my first and so far only. I want to do the larger version someday, but my job and NoCOUG have so far prevented that. I have all the materials but I haven't made the time so far. ▲

Interview conducted by Iggy Fernandez

All of Brian's book reviews are available at **www.brianhitchcock.net**.



More Oracle Secrets

by Norbert Debes

Norbert Debes

Valid node checking is an interesting security feature that protects DBMS instances from malevolent or errant Oracle Net connections over TCP/IP, without the need for a firewall or IP address filtering at the operating system level. It's a "poor man's firewall" under control of the DBA. The feature is available in Oracle9*i* and subsequent releases at no extra cost. Production DBMS instances may be separated from test and development instances without additional hardware or firewall software simply by specifying a list of nodes that may contact the listener. Valid node checking is documented, but it is undocumented that the parameters are fully dynamic in Oracle 10*g* and Oracle 11*g* such that the configuration may be enabled, changed, and removed without stopping and restarting the TNS Listener, rendering the feature much less intrusive. The feature is riddled by quite a few undocumented aspects and a poorly implemented security bug fix that may cause unplanned downtime.

Improved Security

Here's an anecdote that illustrates why valid node checking is a worthwhile feature. A production database that had several database jobs was copied onto a test machine. The database jobs started running on the test machine. Some of these jobs were using database links. Since the database link definition contained a full Net service name definition, instead of referencing a Net service name in `tnsnames.ora` (an undocumented feature), the test system was able to access a critical production system and caused deterioration in its performance. The administrators got off lightly, since the jobs were read only. Imagine what could have happened had the jobs modified production data. Correctly configured valid node checking would have prevented the issue.

The feature is controlled by the three parameters `tcp.validnode_checking`, `tcp.invited_nodes` and `tcp.excluded_nodes`, which are presented in Table 1 (at right).

It appears that the code path for valid node checking is always executed. As long as the feature is not in use, the lists for invited and excluded hosts are empty, thus allowing any client to connect. This assumption is based on the observation that the TNS Listener writes lines such as those in the excerpt below on each connect by a client to its trace file, irrespective of the setting of `tcp.validnode_checking`:

```
[12-JUL-2007 20:03:12:268] nttcnp: Validnode Table IN use; err 0x0
[12-JUL-2007 20:03:12:268] nttvlser: valid node check on incoming node
10.6.6.64
```

Only when the TNS Listener trace level is at least at level `ADMIN` and `tcp.validnode_checking=no` is set does the trace file contain evidence that valid node checking is switched off:

```
[12-JUL-2007 19:52:28:329] ntvllt: tcp.validnode_checking not turned on
```

Let's take a look at some examples. The prompts `dbserver$` and `client$` indicate where each command was run. The `listener.ora` and `sqlnet.ora` on the server are below:

```
dbserver$ head -200 listener.ora sqlnet.ora
==> listener.ora <==
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = TCP)(HOST
=dbserver.oradbpro.com)(PORT = 1521))
      )
    )
  )
trace_level_listener=admin
==> sqlnet.ora <==
NAMES.DIRECTORY_PATH= (TNSNAMES)
```

Valid node checking is currently switched off. This is evident from the configuration file `sqlnet.ora` reproduced above. Below is the `tnsnames.ora` on the client:

```
client$ cat tnsnames.ora
TEN_TCP.WORLD =
```

Table 1: Valid Node Checking Parameters

Name	Purpose	Values	Default
tcp.validnode_checking	Turns valid node checking on or off	yes, no	no
tcp.invited_nodes	List of nodes that may connect may connect to the TNS Listener	Comma-separated list of host names and/or IP addresses on a single line	empty list
tcp.excluded_nodes	List of nodes that are denied a connection to the TNS Listener	Comma-separated list of host names and/or IP addresses on a a single line	empty list

```
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST=dbserver.oradbpro.
com)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = TEN)
    )
  )
```

Let's start the TNS Listener and verify that the client can connect to it. For the sake of conciseness, `lsnrctl` output—which does not indicate whether valid node checking is configured—is omitted:

```
dbserver$ lsnrctl start
```

Since the above `sqlnet.ora` file does not contain any of the three valid node-checking parameters, the feature is disabled and any client can connect successfully:

```
client$ sqlplus -l ndebes/secret@ten_tcp.world
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
SQL> EXIT
Disconnected from Oracle Database 10g Enterprise Edition Release
10.2.0.1.0 - Production
```

Enabling and Modifying Valid Node Checking at Runtime

I claimed that valid node checking could be enabled dynamically in Oracle10*g* and Oracle11*g*, i.e., without stopping and restarting the TNS Listener. Note that this works only if the configuration file `sqlnet.ora` is present when the TNS Listener is started (undocumented). Let's verify my claim by changing `sqlnet.ora` as below on the server and running `lsnrctl reload`[1]:

```
dbserver$ cat sqlnet.ora
NAMES.DIRECTORY_PATH=(TNSNAMES)
tcp.validnode_checking=yes
tcp.excluded_nodes=(client.oradbpro.com)
dbserver$ lsnrctl reload
```

The output of `lsnrctl reload` does not indicate whether valid node checking is enabled or not. Now an attempt to connect from the client system client.oradbpro.com fails:

```
client$ sqlplus -l ndebes/secret@ten_tcp.world
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jul 13 02:06:33 2007
ERROR:
ORA-12537: TNS:connection closed
SP2-0751: Unable to connect to Oracle.  Exiting SQL*Plus
```

Of course, translation of the client host name to an IP address with DNS, NIS, or other method must be configured. IP addresses may also be used in the list of invited or excluded hosts. If the TNS Listener trace level is at least `USER`, an entry like the one below, which identifies the client that was denied, is written to the TNS Listener trace file:

```
13-JUL-2007 02:21:02:109] nttvlser: valid node check on incoming node
88.215.114.53
13-JUL-2007 02:21:02:109] nttvlser: Denied Entry: 88.215.114.53
```

[1] It appears that the LSNRCTL utility caches the TNS Listener configuration. When testing, you should always run lsnrctl from the command line, instead of leaving the utility open and running multiple commands at the LSNRCTL prompt. The latter approach may not pick up changes to `listener.ora`.

Setting the list of invited nodes in such a way that `client.oradbpro.com` is included and running another `reload` enables the client to connect again:

```
dbserver$ cat sqlnet.ora
tcp.validnode_checking=yes
tcp.invited_nodes=(client.oradbpro.com)
dbserver$ lsnrctl reload
client$ sqlplus -l ndebes/secret@ten_tcp.world
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
```

The successful connection by the client is logged as below in the TNS Listener trace file:

```
[13-JUL-2007 02:24:44:789] nttvlser: valid node check on incoming node
88.215.114.53
[13-JUL-2007 02:24:44:789] nttvlser: Accepted Entry: 88.215.114.53
```

If `tcp.invited_nodes` is set, any node not mentioned in the list is denied access:

```
dbserver$ cat sqlnet.ora
tcp.validnode_checking=yes
tcp.invited_nodes=(192.168.0.1)
dbserver$ lsnrctl reload
client$ sqlplus -l ndebes/secret@ten_tcp.world
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jul 13 02:06:33 2007
ERROR:
ORA-12537: TNS:connection closed
SP2-0751: Unable to connect to Oracle.  Exiting SQL*Plus
```

Of course, the denied hosts also include the system where the TNS Listener is running, such that subsequent LSNRCTL commands over TCP/IP fail. You need to include the local system in `tcp.invited_nodes` to allow LSNRCTL commands over TCP/IP. Another method is to use an IPC protocol entry as the first `ADDRESS` of the TNS Listener. This tells the LSNRCTL utility to communicate with the TNS Listener using IPC, which is obviously exempt from TCP/IP valid node checking. The next example shows a TNS Listener definition, which uses the IPC protocol in the first `ADDRESS` entry.

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = IPC)(KEY = TEN))
       (ADDRESS = (PROTOCOL = TCP)(HOST =dbserver.oradbpro.
com)(PORT = 1521))
      )
    )
  )
```

By default, an Oracle9*i* TNS Listener is unprotected against STOP commands from remote nodes. Instead of using a TNS Listener password to prevent someone who is logged in on another host within the network from shutting down the TNS Listener, you could also use valid node checking. The downside is that the list of invited nodes has to include all the machines, which may access the TNS Listener. These could still be used to remotely stop the TNS Listener but might be trusted systems. This is interesting news for installations that run clustering software, which protects the ORACLE TNS Listener against node failure but does not support TNS Listener passwords (e.g., VERITAS Cluster Server prior to release 4).

If you want to take a more relaxed approach, you may set only `tcp.excluded_nodes` and list systems that you are certain may not connect to the TNS Listener, and thus the

instance(s) served by the TNS Listener. All nodes not mentioned will be able to connect. Host names and IP addresses may be used at the same time.

There's no sense in setting both `tcp.invited_nodes` and `tcp.excluded_nodes` at the same time, since even nodes not mentioned explicitly as excluded nodes will still be excluded when `tcp.invited_nodes` is set. If a node name is contained in both `tcp.excluded_nodes` and `tcp.invited_nodes`, `tcp.invited_nodes` takes precedence and the node is allowed access.

In Oracle9*i*, if there is a single node name that cannot be resolved to an IP address, this error is logged to the trace file:

```
[12-JUL-2007 21:25:10:162] nttcnp: Validnode Table **NOT** used; err
0x1f7
```

Valid node checking is switched off when this error occurs. Unfortunately, the Oracle9*i* LSNRCTL utility does not write an error message to the terminal. In the presence of invalid host names, Oracle10*g* `lsnrctl startup` fails with "TNS-12560: TNS:protocol adapter error" and "TNS-00584: Valid node checking configuration error". Using `oerr` on TNS-00584 gives:

```
$ oerr tns 584
00584, 00000, "Valid node checking configuration error"
// *Cause:Valid node checking specific Oracle Net configuration is invalid.
// *Action:Ensure the hosts specified in the "invited_nodes" and "excluded_
nodes"
// are valid. For further details, turn on tracing and reexecute the operation.
If TNS Listener tracing is enabled, the trace file will contain a message
similar to the following:
[12-JUL-2007 23:27:16:808] snlinGetAddrInfo: Name resolution failed for
wrong.host.name
[12-JUL-2007 23:27:16:808] nttcnp: Validnode Table **NOT** used; err
0x248
```

A reload of configuration files with `lsnrctl reload` completes successfully in spite of name resolution failures, which are logged in the following format:

```
[13-JUL-2007 00:11:53:427] snlinGetAddrInfo: Name resolution failed for
wrong.host.name
```

However, the TNS Listener silently switches off valid node checking after encountering a non-resolvable host name. This is a serious bug. As far as I know there is currently no patch for this bug. After each reload, the DBA should check the listener trace file for any name resolution failures. It is required to set TRACE_LEVEL_<listener_name>=user for such errors to be logged.

No reverse address translation is performed on IP addresses. Thus, IP addresses that cannot be translated to host names do not prevent the operation of valid node checking. The operating system utility `nslookup` may be used to translate between Domain Name Service (DNS) host names and IP addresses, and vice versa. Keep in mind that `nslookup` does not read the hosts file (`/etc/hosts` on UNIX, `%SYSTEM_ROOT%\system32\drivers\etc\hosts` on Windows, where `SYSTEM_ROOT` is usually `C:\WINDOWS`). So the TNS Listener may be able to resolve a name or IP address by calling C programming language library routines (`gethostbyaddr()`, `gethostbyname()`), while `nslookup` may not.

I ran some tests to determine the undocumented maximum accepted length for the invited and excluded node lists. The maximum line length of the Vi editor I used was 2048 bytes[2]. Both parameters were still working fine at this line length. Assuming an average length of 30 bytes for a host name, this length would provide enough room for around 65 entries. If IP addresses were used, at least 128 IP addresses would fit. The list of valid nodes cannot exceed a single line, otherwise the error "TNS-00583: Valid node checking: unable to parse configuration parameters" is signaled and the TNS Listener does not start.

Caveats

Host names specified as either valid or excluded nodes are resolved once when the listener starts. No reverse lookup on incoming IP addresses is performed to match an incoming IP address with a configured host name. As a consequence, variable IP addresses will not work. If DHCP is used, provisions must be made such that the DHCP server will always assign the same fixed IP address based on the DHCP client's Ethernet MAC address.

Given the constraints imposed by the way the feature is currently implemented and the aforementioned bug that disables valid node checking on reload in case there is a non-resolvable host name, the only safe way to configure the feature is to use only IP addresses and no host names.

Beyond Valid Node Checking

Valid node checking works well when only a few clients need to connect to a DBMS instance, such as in a three-tiered environment where connections originate from an application server. In cases where a large number of clients need to connect, it is far better to run the listener on the loopback adapter (IP 127.0.0.1) or to specify only the local node as a valid node and have all other clients use Oracle Connection Manager (CMAN), which will relay clients to the TNS Listener. If you're running the TNS Listener on a clustered system, remember to add the host names of both cluster nodes to the valid nodes list.

Oracle Connection Manager

CMAN provides rules that allow entire subnets to contact it. This greatly alleviates the overhead of managing a large number of clients. With CMAN, you also have a choice of simply closing an incoming connection (rule action=drop), just like the listener does, or returning a proper TNS error message (rule action=reject) that lets the user know that a CMAN rule prevented the connection. The last rule in a CMAN configuration file should be a rule that drops or rejects all connections that have not been explicitly accepted.[3]

Another interesting property of using CMAN as the first line of defense is that you can disable any attempts to connect that request an `ORACLE_SID` by specifying `srv=<instance service name>` in all CMAN rules listed in cman.ora, where instance service name is one of the service names set with the initialization parameter `SERVICE_NAMES`[4]. Considering that

(continued on page 17)

[2] Other editors allow lines that are longer than 2048 bytes. Vim (an improved Vi) is an enhanced implementation of Vi, which supports a line length of more than 2048 bytes. It is available for free at **www.vim.org** and runs on UNIX, Windows, and Mac OS.

[3] (rule=(src=*)(dst=*)(srv=*)(act=reject))

[4] (rule=(src=<src_host>)(dst=<dst_host>)(srv=<instance service name>)(act=accept))

Oracle Performance Firefighting

A Book Review by Dave Abercrombie

Details

Author: Craig Shallahamer

ISBN: 978-0-9841023-0-3

Pages: 386

Year of Publication: July 2009

Edition: 1

Price: $49.95

Publisher: OraPub

Summary

Overall review: Craig Shallahamer has written a practical, informative, and very clear book that integrates important Oracle performance and scalability topics with valuable communication and problem-solving advice. His lively explanations of latching, wait events, the buffer cache, the shared pool, and redo mechanisms alone are worth the price of the book. Combined with excellent communication tips, a foundation of queuing theory, real-world examples, and a freely available toolkit, this book is truly invaluable.

Target audience: Both seasoned troubleshooting experts and newcomers to the field will gain tremendously from this book.

Would you recommend to others: Yes.

Who will get the most from this book: Oracle performance and scalability troubleshooters who desire to deepen their knowledge, sharpen their skills, and improve their communication.

Is this book platform specific: No.

Why did I obtain this book: While researching a particular nasty bug involving In-Memory Undo (IMU), I found a white paper written by Craig Shallahamer that clearly and concisely explained this feature. I was so impressed by the quality of this white paper that I went to his OraPub website to see what else might be available from Craig. I saw that this book had just recently been published and bought it instantly.

Chapter 1: Methods and Madness

Craig Shallahamer establishes a framework for firefighting in this chapter. His first point is to avoid panic. Of course, this is easier said than done, especially with pagers beeping and with frantic pleas from users and managers. Craig emphasizes the need for clearly established goals and good staff communication. Especially important are regular updates of progress, lack of progress, expectations, and time estimates. Such updates go a long way in demonstrating your care and concern to end users and can drastically cut down on interruptions. However, Craig does not explicitly mention the benefits of having a written emergency standard operating procedure that can guide you in this process, allowing you to focus on the problem rather than, for example, wondering who to call next.

Craig highlights the need for clear report writing. Although his emphasis on writing and communication skills might seem out of place in a very technical book, I am grateful for his focus on an oft-neglected topic. I am sure we all have examples of great ideas and solutions that were delayed or never implemented due to lack of a compelling and concise story. Craig's very practical advice goes so far as to include a report template integrated with tips on the report writing process.

Craig describes the OraPub "3-Circle" analysis, based on three overlapping circles: operating system, Oracle, and application. Although this perspective may seem obvious and trivial, deliberately using it as an active framework will reduce risk, strengthen analysis, and encourage a wider variety of solutions. Solutions will arise more coherently, be easier to communicate, and will encourage cooperation. Craig refers to OraPub's 3-Circle throughout the book. I have already adopted this perspective in my own work, and I can attest to these benefits.

Chapter 2: Listening to Oracle's Pain

Much writing on Oracle's wait interface begins with a description of the relevant views and wait classes. Craig approaches this topic from a refreshing and more easily understood angle: he walks us through an example of a team of programmers adding instrumentation into an existing application. Readers come to appreciate this wait event instrumentation from the perspective of Oracle kernel developers, where it is seen correctly as a major breakthrough in performance analysis.

Craig guides the reader through an operating system trace (Linux strace) in which we can follow an Oracle session as its instrumentation issues gettimeofday() calls both before and after a physical read request. I was amazed at how many basic insights flow from this simple example. His point that every Oracle process is either on the CPU or posting a wait event arises quite naturally from this example.

Craig leads the reader through the three main wait event views: v$system_event (cumulative red line system-wide), v$session_event (cumulative red line session-level) and v$session_wait (real-time snapshot), and brings in some example scripts from OraPub's System Monitor (OSM) toolkit (URL below).

Craig returns to the topic of response, service, and queue times, and it is much easier to place these into context now that we've been through an explanation of Oracle's wait event instrumentation. Craig's emphasis here again is communication. His examples break response time down to several successively greater levels of detail, and using the level most appropriate to

your audience can help you to get your point across more effectively.

Craig finishes this chapter with a tour of several of OraPub's ORTA reports. We go through instance and session level reports, and we just barely touch on the benefits of Oracle's End-to-End Metrics columns such as v$session.client_identifier. These OraPub ORTA reports are all available for download in the OSM toolkit.

Chapter 3: Serialization Control

Oracle protects its memory data structures with latch and mutex control structures. They are rather like tokens: "Once a process has the control structure, it is then, and only then, that it is allowed to access the structure." These control structures must be very fast, which Craig illustrates with a simple example: if Oracle took as long as 1 millisecond to see whether or not a block was cached, then a 5,000 buffer get query would spend an absurd five seconds just answering this question. Craig shows that Oracle's use of these control structures is very simple: a process either has control or not (although some can be shared), and there is no queuing. Craig points out the important distinction between "locking," which prevents memory changes, and "pinning, " which prevents only memory deallocation.

In general, latches are acquired within either shared or exclusive mode, and the shared mode is likely to be faster. Also, latch acquisition attempts are classified either as "immediate" or as "willing to wait." Craig beautifully illustrates a willing to wait attempt with a vivid example: "Picture a room full of people looking intently at the room's center. There is a hole at the very center of the room where the just-released latch shoots out like a rocket. All the people in the room are repeatedly yelling 'Give me the latch' and are swinging their arms as fast as they can, up to 2,000 times, hoping that their hand will move over the spot just as the latch is released. When they look in their hand, the contestants hope to see the latch. If they don't, they continue swinging their arms [up to _spin_count times, usually 2,000] hoping the next time they will be the lucky one." After two consecutive batches of 2,000 frantic spinning attempts, Oracle will insist that an unsuccessful process must sleep for 10 milliseconds before waking up for another set of 2,000 chances.

Craig highlights Oracle's methods for accounting for spinning and sleep time, and this is very important to understand when looking at its wait interface. It turns out that Oracle considers the spinning phase, which consumes CPU resources, to be service time, and it records the sleep time as latch wait time, considered as queue time. The user, of course, experiences the sum as response time. An operating system administrator looking at CPU utilization and run queue length would see service time problems but not have direct insight into the sleeping (latch wait) portion of a user's increased response time. A DBA looking at wait times would see details about why processes are sleeping during queue time without knowing exactly how many CPU resources were consumed beyond knowing that significant spinning service time has already occurred before the sleeping wait for the latch. Craig concludes this accounting discussion with a real-life graphic example that illustrates how well Oracle's latching mechanisms work under even extreme loads.

Chapter 4: Identifying and Understanding Operating System Contention

Craig leads us through the operating system with a goal of identifying bottlenecks relevant to his 3-circle analysis method. He discusses four subsystems in turn (CPU, memory, IO, and network) while emphasizing data readily available from within Oracle and through the use of standard, freely available operating system tools.

Craig starts with CPU, since its behavior closely follows queuing theory predictions for response time and queue length. He demonstrates the CPU subsystem using a simple graphic depicting four cores and one queue. This diagram is generally very helpful, and Craig uses it often throughout the book. Unfortunately, due to its extreme simplicity, this diagram cannot distinguish between two very different situations: 1) a very idle CPU subsystem with one active transaction, and 2) a long running transaction holding serialization locks and so preventing other processes from making use of the CPU.

Craig explains how CPU time is commonly divided into the three or four categories of "user," "system," "idle," and often "IO-wait." Craig warns us that "not all IO-subsystems will show wait for IO time, even when there are significant IO wait issues." He defines utilization here as the fraction of available time spent within user and system categories (over a defined averaging interval, of course). Craig demonstrates the use of operating system tools like vmstat, sar, and the /proc filesystem to gather CPU utilization details. Craig also brings our attention to the very useful Oracle view v$osstat that brings these CPU utilization details within reach of simple SQL. This v$osstat view is extraordinarily valuable if you lack convenient shell access to the database host itself.

Craig walks us through a very clear demonstration of how response time curves vary depending on the number of CPU cores. Any queue-based system, including a CPU subsystem, contains a "knee" in its response time curve. As arrival rate (i.e., workload) increases, response time remains almost constant, rising only very gradually. Eventually, a point is reached where queue time begins to dominate response time. As the arrival rate workload increases further, queue time ramps up, response time quickly deteriorates, and the system hits the wall. The location of this knee depends on the number of CPU cores: a large number pushes the knee to the right (higher CPU utilization), with a steeper wall, whereas a small number of CPUs brings the knee to the left (lower utilization) with a gently slopped wall.

Craig points out a much underappreciated distinction between online transaction processing (OLTP) and batch processing systems. Users of OLTP systems value quick response time; they therefore want the system to run at low utilization, thus minimizing transaction queue time. On the other hand, users of batch systems want the system to run at high utilization, using as many CPU cycles as are available, thus maximizing throughput and minimizing idle time (but without letting the CPU run queue to get too long, which would incur more process management overhead costs). This can become a delicate balance in those systems that do both OLTP and batch processing.

Craig briefly goes over operating system memory issues. He

starts, and ends, with advice regarding communication: never use words like "bottleneck" or "swapping" with operating system administrators; instead use terms like memory "pressure" or the more generally useful claim that "Oracle's memory requirements exceed available capacity." Craig leads us through the bewildering categories of memory, such as real, virtual, shared, nonshared, private, code, data, stack, resident, etc., and he relates them to Oracle's categories like PGA and SGA. He continues with practical example usage and interpretation of commands such as vmstat and sar.

Craig then returns to his simple diagram to contrast the IO queuing model with the CPU model. Since a specific IO device can service only those requests for data on that device, an IO subsystem is best modeled with a separate queue for each device. This is quite a bit different from a CPU, which can generally service any incoming request. We therefore talk about balancing an IO subsystem, but we never talk about balancing a CPU subsystem. This realization leads directly to understanding why IO subsystems start queuing well before reaching high overall arrival rate workloads.

Craig walks us through the Oracle wait interface, pointing out the common physical read related wait events that can help to diagnose IO bottlenecks. Of course, his OSM toolkit includes queries that go against relevant views such as v$system_event and v$event_histogram. Interestingly, although this section again stays clear of mentioning Oracle's ASH, which can provide remarkable insight here, Craig does include discussion of AWR. He also leads us through examples of the use of operating system diagnostic tools such as sar, iostat, and NetApp's sysstat. In line with Craig's emphasis on effective communication, he reminds us not to tell the operating system administrator that "the disks are slow," instead recommending less confrontational language such as "Oracle's IO requirements exceed the disk's capacity."

Craig continues with a brief summary of network issues. Network latency can be diagnosed through use of commands like Oracle's tnsping, which better simulate SQL*Net packets than the classic ping command. Craig recommends gathering hundreds or thousands of observations at gentle rates of once every 30 or 60 seconds. He points out that network collision rates and dropped packet rates can be measured with commands such as ifconfig and netstat. However, he does not delve into the challenges of interpreting wait events such as "SQL*Net message from client."

Chapter 5: Oracle Performance Diagnosis

In this chapter, Craig integrates and expands the concepts introduced in previous chapters, setting the stage for explanations of Oracle internals to be presented in the following chapters. He starts by explaining limitations of Oracle's visibility into the operating system. For example, he points out that Oracle's instrumentation depends on asking the operating system "What time is it?" During periods of extremely high CPU utilization, (90% or greater) answers to this question might be delayed, leading to significant reporting error.

Craig contrasts the older performance statistic views, such as v$sysstat and v$sesstat, with the newer "time model" views v$sys_time_model and v$ses_time_model. One important difference is that the older views gather timing information only at the end of a call, but the time model views update about every six seconds (while a session is "active"). This improvement provides much greater insight into long-running statements. In general, the older views and the time model views will show slightly different timing values; Craig recommends use of the latter.

Craig points out that these performance statistic views, both old and new, do not provide service time (i.e., CPU) details as rich as provided by the wait interface. While Oracle's wait interface shows details for more than several hundred types of queue time waits, the performance statistic views split service time into just a handful of categories, and these are not always mutually exclusive. These categories do clearly distinguish between "server processes" (users' SQL) and background processes (e.g., the database writer). They all also show values for parsing CPU time and recursive SQL CPU time; however, these are not mutually exclusive. For example, the parsing CPU time values include both background and server processes, and both recursive and non-recursive SQL. Also, the recursive SQL CPU time values include both background and server processes as well as recursive SQL parsing time. Craig also explains Oracle's definition of recursive SQL: it includes "any SQL with a depth greater than one." So, for example, an application SQL statement run inside of a PL/SQL loop will be seen by Oracle as recursive.

Craig discusses an example of the "ghost IO bottleneck." Here, a wait event report shows physical reads as the top wait event, but the average wait time is only a very fast 0.6 milliseconds. He points out that you should first see whether application users are even noticing delays, since IO does not appear to be a real problem in this example. But assuming that performance is indeed unacceptable here, Craig shows potential solutions in all three circles: increase buffer cache (Oracle), tune SQL (application), or solve the CPU bottleneck (OS, not uncommon with these symptoms).

Craig emphasizes that looking only at average values can mislead. Looking at the distribution of wait times may reveal important clues. His OSM script swhistx.sql computes deltas for the v$event_histogram view, and this can show skew, i.e., non-uniformity, in the wait times. In some cases, the average wait time might be low, but looking at these histograms might show that a significant number of waits take far longer than expected.

Craig dispels several common wait event myths. For example, decreasing wait time does not always improve performance: decreasing latch queue time by increasing spinning service time can increasing overall response time. Another reason that decreasing wait time does not always improve performance is that the database is often only a small fraction of overall response time.

Craig illustrates the use of Oracle-supplied package DBMS_MONITOR to profile sessions or even groups of specific sessions. He shows how to use this tool to gather performance statistics that appear in views v$service_stats, v$serv_mod_act_stats, and v$client_stats. Craig also demonstrates the use of DBMS_MONITOR to trace sessions. Effective use of this tool requires the use of Oracle "End to End Metrics" (JDBC)

or DBMS_APPLICATION_INFO (PL/SQL), but Craig just scratches the surface of this wonderful application-level instrumentation API.

Craig introduces Oracle's Active Session History (ASH): Oracle records key facts about active sessions about once per second and maintains a historical buffer of its observations in a table called v$active_session_history. This buffer may retain these observations for only a few hours, depending on activity. However, a view in Oracle's Active Workload Repository (AWR) called dba_hist_active_sess_history retains one-tenth of these observations far longer, perhaps a week or a month or more, depending on how it has been configured. Craig highlights some of the amazing insights that these tools provide and demonstrates his OSM scripts ashrt.sql and ashsqlpcte.sql.

Chapter 6: Oracle Buffer Cache Internals

The real strength of this book is its clear, concise, and relevant description of Oracle internals. Craig never digs deeper into details than is necessary for solving real problems. He hits the sweet spot just right, providing a clear, intuitive, and most importantly, a practical understanding of Oracle's internal mechanisms. This chapter on buffer cache internals sets the standard for clarity and appropriate level of detail. Once again, this review cannot hope to do this rich material justice, so you will just have to read the book.

Chapter 7: Oracle Shared Pool Internals

Craig next walks us through the contents of Oracle shared pool, which contains a wide variety of memory objects, including SQL cursors, PL/SQL code, the library cache (metadata about tables, indexes, etc.), the row cache (data dictionary rows, not blocks), in-memory undo, and many others. Many of these memory objects are interdependent, and so a change in one object often cascades through many others.

Craig explains that a cursor is an executable representation of a SQL statement, created via a hard parse, drawing from data in row cache, library cache, and optimizer statistics. He leads us through cursor pinning and locking, pointing out how even pinning a cursor in shared mode is a serialized process.

Craig continues with an explanation of the library cache by using an analogy to an actual library. You begin your search for a book by going to the relevant card catalog (hash bucket), waiting in line (library cache latch), and sequentially scanning for the book's entry (searching a hash chain). You find the book's card (library cache handle) and then you walk over to the book and begin reading (cursor access). This analogy provides a firm foundation for applying Craig's practical advice on identifying and resolving library cache latch and mutex problems.

Craig goes over shared pool memory allocation and deal location issues. He concludes this chapter by walking us through Oracle's In-Memory Undo (IMU) feature, which was added in 10g Release 2. Once again, his advice on identification and resolution of these issues is easy to understand after his clear explanations of the underlying mechanisms.

Chapter 8: Oracle Redo Management Internals

Craig's final chapter on Oracle internals describes redo management. After having seen how the buffer cache and shared pool operate, redo management appears rather straightforward. However, Craig points out that solving redo problems is usually quite difficult, since changes in the application or IO subsystem are often required.

Craig whets our appetite by highlighting little-known facts about redo. For example, when undo segments change, these changes need to be included in the redo stream. Craig explains that redo-based recovery consists of two core phases: Oracle first applies redo to roll the database blocks forward in time. When this phase is complete, the database blocks contain both committed and uncommitted changes. Oracle then applies undo to reverse any uncommitted transactions. With this explanation, one can see that the redo log must also contain undo information, in order to reverse these uncommitted transactions during recovery. Another little-known fact illustrated by Craig is that non-DML queries can generate redo: Oracle often delays one session's post-DML block "cleanup" until the next time a buffer block is read by another session, and the ensuing block changes (not involving user data values) must be recorded in the redo stream.

Craig illustrates Oracle's redo architecture with simple and clear graphics. Starting with 9i Release 2, Oracle introduced multiple "strands" of redo, each with its own latch. Craig's graphics demonstrate quite clearly the benefits of this improved architecture. Craig shows the three fundamental stages in redo activity: space must first be allocated in the log buffer, redo "vectors" must then be copied into the buffer, and finally, the log writer flushes the buffer into the redo log file. With these internal mechanisms in view, Craig's practical advice on spotting and solving redo bottlenecks such as "log file sync" and "log file parallel write" does not appear so mysterious.

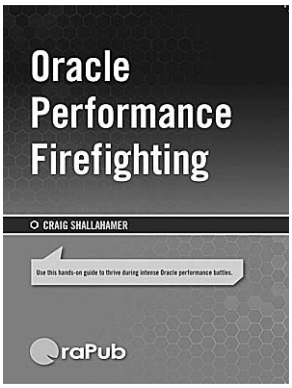
Chapter 9: Oracle Performance Analysis

Craig's concluding chapter revisits response time analysis in greater depth. He begins by going into more detail regarding arrival rate, suggesting use of Oracle's 'user calls' metric to measure workload. He defines utilization as requirements divided by capacity, using an analogy of a pitcher full of water (the requirements) and a cup (the capacity): will the cup overflow? Craig discusses several metrics that can be used for utilization, including CPU seconds, SQL executions, commits, redo bytes, and logical IOs. He suggests that we avoid a single rigid definition; we should remain flexible and be prepared to use a metric appropriate to the situation.

Craig goes over the places we can find Oracle's CPU-based and IO-based requirement metrics within STATSPACK and AWR reports as well as within the underlying views: v$sysstat is fine for IO but v$sys_time_model is better for CPU metrics. Regarding capacity, Craig suggests using the number of CPU cores to derive a time-based capacity metric such as total "CPU seconds per hour." Craig points out that IO capacity is a little harder to come by, requiring a visit with the IO administrator. Craig's ORTA methodology also makes use of operating system CPU utilization, and he explains how this can be derived from v$osstat.

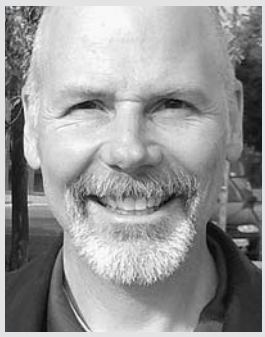
He then describes how to calculate Oracle's service time, queue time, and response time. This information can then be

(continued on page 17)



Firefighting 101

From *Oracle Performance Firefighting* by Craig Shallahamer



Craig Shallahamer

Over the years, I've noticed a pattern or characteristics of firefighting situations. I think we can learn a lot about prevention if we think about the common elements that help stoke the fire:

The critical production system is involved. This one is kind of obvious because, if the situation were not critical and a production system were not involved, then it wouldn't be classified as firefighting. But it's a common element.

Technical promises are made by nontechnical people. When a performance situation escalates, it's common for management to get involved. While this can bring calm to the situation, if management desires peace at any price, it can lead to promises that are technically unfounded and put unnecessary stress on the technical team. You need to be diplomatic in these situations. Never agree to something you know is false and cannot be realistically accomplished. In the end, you could be held accountable simply for agreeing.

The timing is unfortunate. How about Friday afternoon just before you are about to leave on a holiday—sound familiar? It has happened to everyone.

Significant change occurred. It may seem obvious while reading a book, but when you're in the battle, changes like an operating system patch, an Oracle database patch, an application patch, or even an additional module or application can escape everyone's attention. Think about what could have possibly changed, because it will help you understand what you are technically observing.

Surprise! Change is fine as along as we anticipate it. Even unanticipated change is OK if we allow room for the unexpected. But what really causes problems is a complete surprise that no one was expecting. That punch-in-your-gut surprise when you first arrive in the morning can be enough to throw everyone into a tizzy. If there is no surprise, people have enough time to prepare and respond to the situation, which is then less likely to become an intense firefighting situation.

Key business processes are slow. When business processes are slow, people get irritated, whine, and complain. But when the process directly affects the business, managers become concerned and feel they must step in. For example, if your business cannot recognize revenue until the product ships, the product shipping business flow must be smooth and uninterrupted. Problems will directly affect revenue, the stock price, people's bonuses and paychecks, and employees' jobs. So when people are complaining, ask yourself if their activity is key to the successful operation of the company. If so, be sensitive and careful about how you proceed. If not, delicately turn your focus elsewhere.

Applications are generally slow. If the situation is intense, usually the root cause is harsh enough to affect nearly all application areas on the system. So not only do you have people screaming about specific key business processes, but there is also a grumbling application user chorus. This is when you must prioritize and communicate to people what you are doing and why.

Extreme pressure exists. In a firefighting situation, it is not uncommon for jobs to be on the line. When this occurs, people freak out. Their fear turns to anxiety, and they panic. People express this in different ways. I've seen people break down and cry. I've seen a distraught chief information officer (CIO) physically grab another person and drag him across the room. People become unstable. You need a way to methodically move forward to avoid becoming trapped.

Olympic-level finger-pointing occurs. Oracle database administrators (DBAs) are not trained like salespeople. Salespeople are amazing at deflecting responsibility. If a salesperson tells you the problem is with a module, you may say something like, "I can look into that." And to your surprise, the salesperson's response is, "Oh, so it's your fault! We all hope you can fix it." See how easily the situation turned from your willingness to help to being your fault? This is what I'm talking about. Try to clarify what you say, so the expectations of the company do not rest on your shoulders.

Every time I encounter a firefighting situation, I follow a consistent series of steps. Not only does this list guide my actions, but it also lets others know my plan. If people know you have a plan and understand what that plan is, they are more likely to leave you alone. They will have more trust in you, which is important in these situations.

Don't Panic

Never panic. You are a highly paid professional! Numerous DBA jobs are available, especially if you are willing to visit other places in the world. So, even if you completely screw up, there is a very good chance you can get another job. And you

may be paid more! So relax and enjoy one of the unique and challenging aspects of our line of work. I tell a lot of stories when I teach, and the best stories come from the most intense firefighting situations I've encountered.

Define Your Objectives

Make sure you and everyone else involved agrees with what you are trying to accomplish. Get it down in writing. I have been to countless companies where the DBAs were trying to solve an undefined or shifting problem. Eventually, they became so frustrated that some of them quit their jobs. Usually, this is a management problem, but you need to take control of the situation and clearly outline what you intend to do, thereby also identifying what success means. Don't settle for a moving target, because you'll never win!

Establish the Scope and Get Reinforcements

The ancient Spaniards had a saying that goes, "He was a brave man that day." In other words, you may be awesome, but you're not awesome every day! So get other people involved, just in case you can't do it all.

I take a pragmatic view regarding people who say they can do it all: either they are lying or they have never been involved in an intense firefighting situation. Either way, you don't want to work with them.

Figure out what skills are needed and get them. This is not always so simple, because sometimes the skills are diverse and not available in your group (and perhaps, they should not be available in your group). For example, the typical DBA group will not have operating system tuning experts and application functional experts. The key is clearly identifying the required skills, regardless of where the skills reside. First deal with the scoping, or the definition, and then work to secure those skills.

Most firefighting situations require a very diverse set of skills. For example, you may need Oracle diagnostic, SQL tuning, and operating system tuning skills. But I'm willing to bet you also need someone who is an expert on how to use the application. If your users are new to the application, it's a good bet they are trying to use the new application in the same way as they used the old application. While they may get the work done, it might not be the best way to use the new application's capabilities. This directly impacts the system, because the users may be requiring the system to do more work than necessary. The problem is that you and your management team won't know this, but a functional expert will. So consider investing in someone who can help your users. The reduction in system workload can have a dramatic impact on performance.

Consider at the outset of the crisis that this might be a long ordeal that continues through the day and night, and back into day again, for 16, 30, 40, or more hours. No one person, however heroic, should expect to endure, or be expected to endure, for the entire duration. One of my first considerations is who will be available within 8 to 10 hours to take over for me for 8 hours. If you are the most capable person to begin the firefighting effort, then find the next best person to take over from you, and plan on at least an hour of "handover" overlap as well.

Get a Baseline of Current Performance

Related to a well-defined objective is an actual sample of the key problem area response times. This will become your baseline. A baseline is used for comparison. For example, when someone says, "the query now runs 50% faster," you still want to know if the query runs in 2 seconds or 2 hours. It's similar to a commercial stating something contains "35% less fat!" A statement like that screams for a baseline, and is clearly intent on deceiving and manipulating. Don't fall into the same trap.

If you are not sure what the key business transactions are, then ask your manager. Even better, set up a meeting with the user representatives and ask their priorities. Keep it limited to just a handful of transactions. Let everyone know that initially your team must focus on the absolute business-critical transactions and nothing else. Reassure them that after these transactions are running well, you will move on to another tier of performance analysis.

After selecting the transactions, take multiple samples. It is best to have users run the transactions and have them watch you write down the timing result. This will increase their trust, which at this point is probably at an all-time low. You don't need to take 30 samples—3 to 5 will be enough to give you a good idea of the situation. Then write these down and perhaps post them so everyone knows the baseline.

Install Your Tools

Regardless of which tools you use, you will need interactive and historically focused tools. Each has a distinct focus, and most vendor-supplied tools naturally fall into one of these categories.

Interactive tools operate in real time. When you log in to Oracle using SQL*Plus and run a script, you are being interactive. Interactive tools have the distinct advantage of going technically very deep. All the data is available because the activity is happening now. Hyper-focused interactive tools usually have dazzlingly graphics. However, be aware that interactive tools do have limitations. During a firefight, there is usually a lot of chaos, and while you are focusing on one problem, another problem is occurring. If you are limited to interactive tools, you could miss a key investigative opportunity.

Historically focused tools periodically collect data and then allow you to "look back in time," as they say. Examples of historical tools are Statspack and AWR. Historical tools are wonderful for analyzing a problem after it happened (like while you slept last night), when multiple problems need to be analyzed, or if you need to reference a past situation (this occurs more often than people think). Historical tools usually provide trending and graphing capabilities, which are especially helpful in communicating with nontechnical people.

Historical product vendors have a very technically challenging problem. They want to gather as much data as possible and as frequently as possible. However, vendors know the more frequently they gather data and the more data they gather, the more their collection activities will impact the system. This is one reason why good historical products seem expensive. In Chapter 5, which covers Oracle performance diagnosis, I will address data-collection challenges.

Firefighting vendor products shine when they blend interactive and historical capabilities. Regardless of the tools you use, ensure you have both interactive capabilities and historical capabilities. Always have your tools installed, running, and ready.

Develop a Simple Communications Strategy

If performance is disastrous, put simply, users assume you're an idiot. They believe you do not understand there are problems, you don't know what the problems are, or you're not fixing the problems. Unless you have done something to change their minds, that's what they will think.

How could users be so cruel? Look at it from their perspective. They may believe it is solely your responsibility to keep the system running smoothly and that you get paid very well to do this. So if performance is unacceptable, it follows that you are not doing your job or are incapable of doing it well. That may be completely incorrect, but unless you do something quickly to break that thought process, it will become truth to them and, unfortunately, perhaps to others.

Breaking this destructive thought process is easier than you may think. If you have met with users about the key business processes and have timed those processes, you have already demonstrated you care and know what is important to them. If you have not met with them, then schedule a meeting right away. Time is of the essence.

Next, create a way for users to monitor your progress whenever they wish. This could be as simple as a basic web page, with both numbers and graphs, or a notice with the same information periodically posted in a location convenient to the users. Keep it simple and to the point. If your posting becomes confusing to them, not only will they continue to interrupt and distract you from doing your work, but you will have also failed to break the destructive "you're an idiot" cycle.

Pick the Low-Hanging Fruit

I have rarely been to a customer site where there was not an immediate and relatively easy performance improvement opportunity. It is in everyone's best interest to perform a very quick analysis looking for these opportunities. While my analysis is performed quickly, I still follow my methods, but I don't spend as much time gathering perfect data and waiting for the "ah ha!" moments to occur. It's more of a "ram the analysis through my brain as fast as I can" exercise. Then I discuss the options with my client. Some changes may require an instance cycle; others may not. I also will try to give input, quantifying if possible, on which solution will impact performance the most. We look at combining uptime requirements with potential performance gains, and together, my client and I determine how to implement my recommendations.

Finding the low-hanging fruit not only improves performance, but also gives everyone involved confidence and hope. Because these situations are so charged with emotion, hope and confidence are desperately needed.

Conduct a Deep Performance Analysis

At this point, I should have good performance data available, some user hope and confidence, the objectives clearly defined, and the performance baselines established. Now, I very methodically perform my analysis, implement the solutions, give the system time to stabilize, gather more data, and repeat until it's time to stop.

Students and clients frequently ask me how I know when to stop improving performance. The answer, while not providing the warm, fuzzy feeling people may want, is straightforward. I keep working on improving performance until there is no time remaining, no budget remaining, or performance is acceptable. This is a very simple statement, but obviously there is much more to it.

At some point, it just doesn't make sense to keep chipping away at performance. Diminishing returns are always a factor. For example, the first tuning cycle may improve performance by 50%, the second by an additional 10%, and the next by 5%. Should we keep tuning? Perhaps other options should be considered, such as increasing system capacity or altering the workload. Regardless of how you disguise the situation, it always comes down to time, money, and performance.

Get a Final Baseline of Current Performance

Since you are periodically collecting key business process timing information, gathering the final baseline sample should be trivial. Still, write it down and publish it. Let people know how awesome a job you and your team have done! If performance went from a query taking 12 hours to that query running in 2 seconds, tell them. You don't need to be an egomaniac about it, but at the same time, people need to know they can count on you in the future. Use this opportunity to further instill confidence and hope.

Document Your Success

Closely tied to the final performance baseline is documenting your work and its subsequent success. The focus now turns to documenting what just happened to gain a deeper understanding, informing and training others, and to prevent a similar situation from recurring.

When I worked for Oracle Consulting, I required my team to complete an on-site client report, which we called our engagement summary, and distribute it to the entire group. While traveling, we commonly read each other's engagement summaries. This dramatically improved the technical competence of our group. It also created a deep sense of community and camaraderie. When the group is geographically dispersed, as was the case with my team, this is especially valuable to the members.

Celebrate!

Oracle performance firefighting is a unique time in your life. Most people never get a chance to experience the highs and lows of firefighting. How many times in your career will you be able to celebrate a dramatic and company-critical performance firefighting success? Probably not that many. So when you have survived a firefight, take some time out and celebrate.

When the ancient Greeks went to war, their campaigns were brief. The armies operated in the summer, because they needed to be home again in time for the harvest and gathering of grapes and olives. One battle always settled the business. A war

Firefighting 101

(continued from page 16)

was specifically designed to be of limited scope and be done with quickly. At that time, governments couldn't run a deficit or borrow money to finance a long, drawn-out campaign. If wartime spilled into harvest, both sides could starve! Everyone knew this, and their tactics were based on doing only what was absolutely necessary.

Just like the ancient Greeks, DBAs should use performance firefighting techniques only when necessary. You don't need to go through deep firefighting performance analysis everyday. If you try to do this, you will probably quit, change groups, or become a manager. Take time to celebrate and time to rest. Then your performance firefighting career will be long and glorious! ▲

Craig Shallahamer is a researcher, writer, and teacher for thousands of Oracle professionals on six continents and 23 countries. His expertise is Oracle performance management in which he has published more than 20 technical papers and authored two books Forecasting Oracle Performance *and* Oracle Performance Firefighting.

More Oracle Secrets

(continued from page 9)

most hacking tools tend to use `ORACLE_SID`, this may help defend against hackers.

The advantage of forcing clients to connect with `SERVICE_NAME` instead of `ORACLE_SID` is that `V$SERVICE_STATS` will give you detailed information about the load caused by connections to each service. This is very useful in configurations in which multiple applications connect to the same DBMS instance. Of course, you need to create a separate instance service name for each application—or even for online and batch operation of the same application—by setting the initialization parameter `SERVICE_NAMES`. CMAN processes rules in the order listed in the cman.ora file.

I've been using CMAN on top of the TNS Listener in production with good results for about one year now and have noticed that it is very picky about indentation in cman.ora. I strongly suggest that you start with the sample cman.ora file in Metalink note 579660.1. ▲

Norbert Debes has more than 13 years experience as an Oracle database administrator. For over 6 years, he held different positions and technical roles at Oracle Germany. He is the author of Secrets of the Oracle Database *(Apress, 2009).*

Oracle Performance Firefighting

(continued from page 13)

pulled together to construct response time graphs. Craig provides a five-step plan: 1) know the system bottleneck, e.g., CPU or IO, 2) pick an appropriate unit of work, 3) determine service time and queue time, 4) derive and review utilization values, and 5) create a response time graph. This may sound complicated, so Craig walks us through examples.

Of course, the point of this effort is to increase understanding of the situation and, just as importantly, to clearly communicate it to others. One major advantage of this response time approach is to be able to predict performance improvements from proposed solutions. Craig again emphasizes the need for clear communication, pointing out that "simplification is the key to understanding, and to communicating technical concepts . . . we want to make our performance presentation memorable . . . to motivate change." Craig also cautions us that this response time approach is simplified and does not include validation or multiple data points. Craig provides references to more detailed and rigorous approaches, and he ties this all together with an example performance analysis involving three full iterations. ▲

Each day, Convio sends tens of millions of emails and then tracks the ensuing transactions and activities, such as donations, advocacy, and click-throughs. Dave has honed his troubleshooting and scalability skills keeping these very busy databases happy. He can be reached at **dabercrombie@convio.com**.

SQL Blog

(continued from page 21)

In the manner of Stephane Faroult, I confidently declared that the rewritten query would be much faster than the original version if the tables were well indexed. I was right; the rewritten query completed in a very small fraction of a second. Stephane was right too! ▲

Iggy Fernandez is an Oracle DBA with Database Specialists and has more than ten years of experience in Oracle database administration. He is the editor of the quarterly journal of the Northern California Oracle Users Group (NoCOUG) and the author of Beginning Oracle Database 11*g* Administration *(Apress, 2009). He blogs at* **iggyfernandez.wordpress.com**.

Building a Policy Management System

by John Weathington

John Weathington

If you don't know where you're going, any road will get you there. Policy marks the destination, and without strong policy in place, your company will not know the difference between compliance and noncompliance. And, without strong policy management, your organization cannot prove whether or not it's in compliance.

Policy is the keystone of compliance, which is why it deserves the most attention when organizing a technological solution for compliance. If policy is the keystone of a doorway, then what you must build in your technology organization is the architectural support to make the doorway functional and reliable.

There's a lot under the surface of a good policy management system that most companies don't appreciate until their practices get put to the test with an audit. As children growing up in California, we were taught to stand under a doorway in case of an earthquake, because this is one of the strongest areas of the house. But it's not the keystone that keeps you safe, it's the underlying structure. This is also the case with your organization. To survive an audit, your company must be able to prove its innocence, and it's the infrastructure you build that will make the difference.

Keeping Track of Your Policies

I deal with all flavors of compliance, from SOX to HIPAA to PCI and beyond, so I've seen every incarnation of law, standard, and guideline that you can imagine. What are interesting are the common threads that weave their way through all sorts of compliance. Good policy is one of them.

Policy is a way to demonstrate that your organization understands and respects the law or standard that needs to be complied with. In theory, if you're in compliance with your policy, by extension you're in compliance with the law that prompted the creation of your policy. The reason that policies are necessary is because, generally, when laws, standards, and guidelines are created they are vague and unspecific. Don't ask me why, they just are.

So probably the very first thing an auditor will ask for when conducting a friendly audit or assessment is a copy of your organization's policy. Not being able to produce that policy quickly is the biggest mistake your company can make. These are the things that can turn an assessment into an investigation.

It's probably not your job to create policy, but you will be involved in storing it somewhere. In this day and age it's somewhat ridiculous to rely on printed documents and filing cabinets for your policy storage system (although it's a pretty good backup), so your first job will be to create a simple document management system.

At a minimum, you will need to image the document and store it in a BLOB format or similar. This is not an option, this is the minimum. You don't know how many companies go through great efforts to create elaborate policy management systems, and they don't store the physical document. This is a bad idea; you need the imaged document stored in permanent record. You should not use an obscure image format. TIFF (tagged information file format) is probably the safest bet; however, PDF (portable document format) is also acceptable (but not recommended—I have my reasons).

In addition to the document image, at a minimum you should store the title of the policy, the version, and the date when the policy went into effect. Additionally you may want to add some notes highlighting the differences from one policy version to the next. The grain of your policy table should be policy version, meaning that you should have one row in your table for each version of the policy. This is very important for two reasons. First, policies are never static; they're constantly being reinterpreted, so they're constantly changing. Second, policy is always tied to a time period; if they investigate your company's practices from two years ago, you'll need to know the policy that was in effect two years ago.

Who Knows About Your Policy?

If step one is having a policy, step two is telling your employees about your policy. Another common thread with all regulations and standards is that the policy must be communicated to people. It seems obvious, but once again it's not about the deed, it's about the proof.

Your company will undoubtedly create a compliance training program to inform people about the new compliance procedures. I recently started an engagement with a large e-commerce client, and before I could even start becoming productive, compliance training on issues like anti-money laundering was mandatory. It was a CBT (computer-based training) with a test at the end to make sure I understood the material.

I'm not sure what they did with the record of my score, but surprisingly this is an area where I've seen weakness in compa-

nies, primarily because they try to handle the record keeping in functions outside of the corporate IT department. This is another bad idea.

Content creation is the responsibility of the policy department, format is the responsibility of the training department, but record keeping should be the responsibility of the IT department. You should take a holistic view of training, as opposed to what format is used. For instance, it doesn't matter if the employee took the training through an online course, a home study course, or instructor-led training. The fact that training was done is what needs to be recorded.

You can set up a simple system that's tied to your HR database and/or identity management system; for example, your LDAP system. This is important, as it ties your employee training to a larger infrastructure that will track peripheral but sometimes important information about the employee that took your training. If your HR system doesn't have good change management around your employees, consider this in the design of your tracking system. For instance, some policy might require that a certain procedure be performed by a director or vice president. If your HR system doesn't record point-in-time information about roles and titles, you'll need to store that in your system.

In your training records system, record the employee and other supporting information as necessary (like title or position), the policy and version in effect when the training was taken, the date and time when the training was taken, the format (e.g., online, instructor-led), the instructor if applicable, and the employee's score if a test was administered. Most training programs have versions of their own, so it would be good to record that information as well as the actual course outline and materials.

Don't Shoot the Messenger

If your company is involved in a potential compliance violation, having someone report the issue can only help matters. Unfortunately, people sometimes feel uncomfortable reporting violations as there may be some unpleasant reverberations as a result. Have you ever seen a TV show where the detective goes around asking people in the neighborhood what happened and nobody seems to have witnessed anything?

Your company's compliance team will definitely set up a whistleblower program where complaints can be properly managed. Again, I don't understand why companies don't properly engage IT in these undertakings. As with record keeping for training, not engaging IT is putting the company at great risk. Even if the whistleblower program is effective, if you can't prove what's going on, it doesn't matter (I hope you're picking up on the theme here).

Whistleblowers need special protection, so the architecture needs to be handled that way. Consider anything submitted by a whistleblower as highly sensitive data. Keep it encrypted in the database under lock and key, and keep good records of who gets access to this information. You may also need to support the anonymous submission of a complaint, but that's more on the operational side.

Each and every complaint needs to be acted upon, so you'll need to store good records on how the complaint was followed up. You should design your system just like an issue-tracking system. You'll need to engage your whistleblower response team on their follow-up process, and you might even need to train them on good practices. Of course this comes naturally for us as issue tracking is built into our DNA, but your company's response team might not have those habits honed yet.

A good practice for whistleblower programs is to follow up with the submitter periodically to ensure that nothing negative has happened as a result of filing the complaint. Having this policy in place is great; however, having recorded evidence that these checkups are being performed is awesome. A key concern of most auditors is that some form of retaliation will result from a complaint. Demonstrating with evidence that this is not the case will bode well for your chances of receiving a passing grade. Even in the absence of a formal follow-up, you might be able to indirectly collect information in this regard from your HR system. For instance, if you have record of a complaint in March by an employee that was terminated in July, you might have a problem! Don't wait until the audit to catch problems like this; you can easily get a feed from your HR system that will highlight this situation.

Information Technology as a Compliance Partner

What we've done here just scratches the surface on how you can help; it's certainly not a comprehensive guide to policy management. The key to making it work is in the engagement. You should be working with your compliance department as a partner not an order taker. They know what needs to be done but they don't always know the best way to get it done, especially when technology is involved (which is 98% of the time).

Policy management should be the first concern within your partnership, as it sets the stage for everything else that follows. Start with the basics of simply recording each policy (in image format) with good change management around different versions. Then build good record keeping around your compliance education system, taking great care to store who took what training and when. Finally, build out your whistleblower program to make sure you can prove that employees are protected and potential violations are diligently followed up on.

Even when Alice figures out where she needs to go, she still needs a good road to get there. The technology organization is ultimately responsible for making it all happen. It starts by having an honest conversation with your compliance department. Set up a meeting today to discuss how you can reinforce your company's compliance program. ▲

John Weathington is a management consultant who helps companies dramatically improve efficiency and avoid penalties and fines. His San Francisco Bay Area–based company, Excellent Management Systems, Inc., has helped companies all over the world like Sun Microsystems, Cisco, and PayPal (an eBay company). He is a Project Management Professional (PMP) and a Six Sigma Black Belt, as well as an Oracle DBA and Business Intelligence Architect. He recently helped a large technology firm fortify a $100 million government contract with the implementation of a custom compliance data warehouse. For more information, please access his website at **www.excellentmanagementsystems.com**.

SQL Blog

by Iggy Fernandez

Iggy Fernandez

July 13, 2009 The Curious Case of the Cartesian Join

The curious thing was that the SQL query produced the correct results in spite of the Cartesian join! Of course performance was not optimal because a full table scan was required. Here's how it all went down.

Here is a query on the Employees and Departments tables. There are individual restrictions on each table but there is no joining condition involving both tables.

```
SELECT
  e.department_id,
  e.first_name,
  e.last_name
FROM
  employees e,
  departments d
WHERE
  e.department_id = 60
  AND d.department_name = 'IT';
```

And here are the results. They are correct!

DEPARTMENT_ID	FIRST_NAME	LAST_NAME
60	Alexander	Hunold
60	Bruce	Ernst
60	David	Austin
60	Valli	Pataballa
60	Diana	Lorentz

Here is the query plan. Notice the Cartesian join operation and the full scan of the Departments table.

Id	Operation	Name
0	SELECT STATEMENT	
1	MERGE JOIN CARTESIAN	
2	TABLE ACCESS FULL	DEPARTMENTS
3	BUFFER SORT	
4	TABLE ACCESS BY INDEX	EMPLOYEES
5	INDEX RANGE SCAN	EMP_DEPARTMENT_IX

The reason why the query produces the correct results is that the restriction on the department_name column of the Department table produces exactly one row.

Here is the corrected query; the correct join condition has been added.

```
SELECT
  e.department_id,
  e.first_name,
  e.last_name
FROM
  employees e,
  departments d
 WHERE e.department_id = 60
  AND d.department_name = 'IT'
  AND e.department_id = d.department_id;
```

The results did not change but the query plan is now more conventional; the Cartesian join operation is no longer present.

Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
3	INDEX UNIQUE SCAN	DEPT_ID_PK
4	TABLE ACCESS BY INDEX ROWID	EMPLOYEES
5	INDEX RANGE SCAN	EMP_DEPARTMENT_IX

You can try the above queries in the HR sample schema if it is installed in your database.

September 22, 2009 Which Query Is Better?

In general, there are lots of ways of expressing a particular query requirement in SQL with implications for query performance. For example, which departments have employees with salaries greater than a certain cutoff? Here are two ways to express this query requirement in SQL. The first uses a conventional correlated subquery, while the second uses ANSI join syntax. Which is better? From a theoretical perspective? From a practical perspective? In certain situations? Since I can only pick one, which one should I pick?

```
VARIABLE salary_cutoff NUMBER

-- Correlated subquery

SELECT d.department_name
FROM departments d
WHERE EXISTS (
  SELECT *
  FROM employees e
  WHERE salary > :salary_cutoff
  AND department_id = d.department_id
);

-- ANSI join syntax
```

```
SELECT d.department_name
FROM (
  SELECT DISTINCT department_id
  FROM employees
  WHERE salary > :salary_cutoff
) e
JOIN departments d ON e.department_id = d.department_id;
```

Inquiring minds want to know!

P.S. The above queries will work unmodified in the HR sample schema.

October 1, 2009 Throw Away That Execution Plan

I was asked to tune the following SQL query; it usually retrieved less than 100 rows but took several minutes of execution time. Table names and column names have been changed in the interests of confidentiality.

```
SELECT DISTINCT
  o.order_id,
  o.order_date
FROM
  orders o,
  order_items ol
WHERE
  LOWER(o.order_state) = : order_state AND
  ol.order_id = o.order_id AND
  ol.item_id IN (
    SELECT item_id
    FROM items
    WHERE item_state = : item_state
  )
ORDER BY o.order_date;
```

The first thing that most of us would want to do is to look at the execution plan generated by EXPLAIN PLAN or DBMS_XPLAN.DISPLAY. However, watch this fascinating YouTube video, *Rewriting SQL Queries for Performance in 9 Minutes* (**www.youtube.com/watch?v=ZVisY-fEoMw**) by Stephane Faroult, in which he claims that he doesn't find execution plans very useful in improving queries. Stephane's point seems to be that if you write high-quality SQL queries and your tables are well indexed, you can be reasonably confident that your queries will perform well. The analogy he gives is the problem of translating from one language to another using computer software; instead of constantly wondering how the language translation engine works and why it is producing poor-quality output, focus instead on writing well-structured and unambiguous input for the engine.

The first thing that I noticed about the problem query was that the FROM clause included tables from which nothing was being selected in the SELECT clause. This is a classic performance trap. For example, consider the query: Which departments have at least one employee? It can be answered in the following ways:

```
SELECT DISTINCT d.department_name
FROM departments d, employees e
WHERE e.department_id = d.department_id;

SELECT d.department_name
FROM departments d
WHERE EXISTS (
  SELECT *
  FROM employees e
  WHERE department_id = d.department_id
);
```

Even though these two queries mean exactly the same thing and produce the same results, Oracle does not process them in the same way. In the first case, Oracle compares every record in the Departments table with every record in the Employees table, thus collecting duplicate department names that have to be eliminated (SELECT DISTINCT). In the second case, Oracle performs an existence check (semi-join) for each record in the department table; no duplicates are produced.

The second thing that I noticed about the problem query was the use of the LOWER function: it would prevent Oracle from using an index on the order_state column. I questioned whether there was any need to use the LOWER function and whether there was a "function-based index" on the expression LOWER(order_state).

The third thing that I noticed was the use of the preposition "IN" to perform an existence check instead of "EXISTS." We need to be very careful when using "IN," "ANY," "SOME," and "ALL" in SQL queries. They behave in unintuitive ways in the presence of nulls (unknowns). For example, the value of the Boolean expression "1 IN (2, Null)" is not False as one might expect; it is considered to be unknown. Similarly the value of the expression "1 NOT IN (2, Null)" is not True as one might expect; it is also considered to be unknown. Finally—since an object either belongs or does not belong to a set—one might expect that the value of the expression "(1 IN (2, Null)) OR (1 NOT IN (2, Null))" is True but that is not so; it is also considered to be unknown. None of this is very intuitive, to say the least. However, refer to the section titled *Use of EXISTS versus IN for Subqueries* (**download.oracle.com/docs/cd/B19306_01/server.102/b14211/sql_1016.htm#i36215**) in the *Oracle Database 10g Performance Tuning Guide* (which suggests that, in certain circumstances, it is better to use IN rather than EXISTS). A similar note appears in the Oracle Database 9*i* guide but I could not find anything like it in the Oracle Database 11*g* guide; presumably the Oracle Database 11*g* query optimizer is smarter than previous versions.

It was time to rewrite the problem query. This is what I wrote:

```
SELECT
  order_id,
  order_date
FROM orders o
WHERE order_state = : order_state
AND EXISTS (
  SELECT *
  FROM order_items ol
  WHERE order_id = o.order_id
  AND EXISTS (
    SELECT *
    FROM items
    WHERE item_id = ol.item_id
    AND item_state = : item_state
  )
)
ORDER BY order_date;
```

(continued on page 17)

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, Hanan Hit, at **hanan.hit@enteros.com**. ▲

Long-term event sponsorship:

Chevron

Oracle Corp.

Thank you! Year 2009 Gold Vendors:

- Burleson Consulting
- Confio Software
- Database Specialists, Inc.
- Enteros
- Precise Software Solutions

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

TREASURER'S REPORT

Naren Nagtode, *Treasurer*

Beginning Balance		
July 1, 2009		**$ 31,407.51**
Revenue		
Membership Dues	1,250.00	
Meeting Fees	190.00	
Vendor Receipts	1,000.00	
Advertising Fee	125.00	
Training Day	–	
Sponsorship	2,000.00	
Interest	4.86	
Paypal balance	–	
Total Revenue		**$ 4,569.86**
Expenses		
Regional Meeting	4,681.04	
Journal	4,257.35	
Membership	–	
Administration	1,029.95	
Website	–	
Board Meeting	534.33	
Marketing	100.00	
Insurance	–	
Vendors	–	
Tax	–	
Training Day	–	
Accounting	–	
Miscellaneous	–	
Total Expenses		**$ 10,602.67**
Ending Balance		
September 30, 2009		**$ 25,374.70**

NoCOUG Fall Conference

Session Descriptions

For the most up-to-date information, please visit **www.nocoug.org**.

Keynote

Introducing Oracle Exadata V2
Juan Loaiza, *Oracle Corporation* 9:30–10:30

The Sun Oracle Database Machine combines industry-standard Sun hardware, Oracle Database 11*g* Release 2, and Oracle Exadata Storage Server software to create a faster, more versatile database machine. It's a completely scalable and fault-tolerant package for all data management, including data warehousing and transaction processing applications. The Sun Oracle Database Machine also includes Sun's new FlashFire technology to cache "hot" data for dramatically improved transaction response times and throughput. This keynote presentation is your opportunity to learn how to consolidate all of your database applications, store up to ten times more data, search data up to ten times faster, and make faster business decisions in real time without making changes to applications.

Auditorium

Extreme Performance with Oracle Database 11*g* and In-Memory Parallel Execution
Maria Colgan, *Oracle Corporation* 11:00–12:00

Introduced over a decade ago, parallel execution is one of the fundamental technologies that enable organizations to manage and access tens and hundreds of terabytes of data by taking full advantage of the I/O capacity on a system. Oracle Database 11*g* Release 2 delivers a milestone for parallel execution, with new groundbreaking in-memory capabilities along with a new automated technique for determining the optimal degree of parallelism for every operation. This session will discuss Oracle's parallel execution architecture, its new groundbreaking in-memory technology, and other enhancements in Oracle Database 11*g* Release 2.

Practical Oracle Capacity Planning
Tanel Poder 1:00–2:00

This session will give you some simple techniques for Oracle capacity planning and performance trend analysis.

With only one hour, we will not go into advanced topics such as queuing theory and scalability modeling but rather introduce simple regression analysis techniques on existing STATSPACK or AWR data, which are easy to apply yet give reasonably good results.

The idea is to give you the core knowledge and readily useable scripts and tools that will allow you to start performing common capacity planning and performance analysis tasks immediately after the conference.

We will be forecasting future CPU utilization, memory usage, and I/O rates; also, we will cover how to handle performance analysis of diverse database workloads, changes in execution plans, and constant data change.

Zero Slides: The Scripts and Tools That Will Make Your Life Easier and Allow Better Troubleshooting
Tanel Poder 2:30–3:30

As the title says, this presentation has no slides at all!

Instead Tanel will demonstrate some of the most useful tools he uses for accurate Oracle performance troubleshooting. The toolset ranges from plain SQL statements to Unix scripts and some custom-built GUI tools—and, of course, Tanel's Oracle Session Snapper will be covered.

During the session you will be taken through a few troubleshooting case studies following a systematic troubleshooting methodology. For anyone interested in Oracle internals and performance tuning, this session should be a fun learning exercise!

Best Practices for Data Loading with Oracle 11*g* Release 2
Oracle Corporation 4:00–5:00

Be ready to get your hands dirty! This session starts with providing an understanding of hardware needs for data loading and sizing of the hardware. Based on that foundation we will discuss the technologies available for fast bulk loading of a data warehouse. We will reserve a special place for flat files and how to load them quickly and efficiently, using the DBFS infrastructure new in 11*g*R2. With the release of OWB 11*g* Release 2 we will also have a quick look at what cool new features are brought to the table with that product. At the end of this session you will have a complete picture of how best to cater to your data-loading needs for Oracle data warehouses. You will have practical and real examples of how to use and leverage these tools for your specific needs.

Room 101

What to Do When the SQL Optimizer Goes Wrong **Editor's Pick**
Kyle Hailey, *Embarcadero Technologies* 11:00–12:00

This session covers how to identify an optimizer problem, why the optimizer makes mistakes, and what to do about it. The session includes modern and relevant SQL performance topics such as:

- VST—Visual SQL Tuning
- TCF—Tuning by Cardinality Feedback
- RTSM—Real-Time SQL Monitoring
- Hints, Profiles, and Query Transformations
- Extend Row Source Statistics
- Plan Stability

(continued on page 26)

Explaining the Explain Plan
Oracle Corporation 1:00–2:00

The execution plan for a SQL statement can often seem complicated and hard to understand. Determining if the execution plan you are looking at is the best plan you could get—or attempting to improve a poorly performing execution plan—can be a daunting task even for the most experienced DBA or developer. This session examines the different aspects of an execution plan, from selectivity to parallel execution, and explains what information you should be gleaning from the plan and how it affects the execution. It offers insight into what caused the Optimizer to make the decision it did as well as a set of corrective measures that can be used to improve each aspect of the plan.

Database Testing: Introducing DB Infrastructure Level Change with Full Confidence
Oracle Corporation 2:30–3:30

Real Application Testing is designed to simplify the testing of all DB infrastructure–related changes, including DB versions/patches, H/W, O/S, and Real Application Clusters (RAC) by capturing real production workload and replaying the load against a test environment. The replay of real production workload against a test environment enables a DBA to identify and tune all SQL performance regressions and provide analysis on the overall performance impact of change(s) introduced prior to production rollout. This method of workload capture and replay ensures the highest-quality testing for Oracle DB infrastructure, reduces testing time, and lowers testing cost.

Deployment Agility Through Tier Testing
Hanan Hit, *Enteros* 4:00–5:00

Oracle's Real Application Testing, which is part of the 11*g* release, allows businesses to quickly adopt new technologies while eliminating the risks associated with system changes. Real Application Testing combines workload capture and replay features with a SQL performance analyzer. This helps identify performance changes by replaying real-life workloads in the new database environments as well as fine-tune these environments before they are deployed to production.

This presentation will explain the significance of Tier Testing in analyzing the behaviors of the database that cannot be detected with traditional user experience testing products. We describe how to use Snapshots Standby database for load testing. We will explore I/O testing with Oracle ORION as well as Availability load testing. We will conclude with a comparison to other database tier load testing.

Room 102

SQL Performance Hero and OMG Method vs. the Anti-Patterns
Jeff Jacobs. 11:00–12:00

Most presentations on SQL Query performance assume that the query developer is highly experienced in SQL and Oracle and focus on tracing and other techniques. However, many query performance problems are the result of common "anti-patterns" typically used by developers inexperienced in SQL and Oracle. This presentation will teach the attendee how to identify and refactor these anti-patterns, generally without the need to resort to tracing, changing init.ora parameters, or other actions requiring DBA privileges. Each solution will cover identifying the problem, understanding why the anti-pattern causes a performance problem, the "fix" for the problem, and how the fix works. Where appropriate, the root cause of the problem will also be discussed.

Physical Data Storage for the Application Developer
Dave Abercrombie, *Convio* 1:00–2:00

A basic understanding of how Oracle stores and retrieves data is essential for understanding performance issues. Such knowledge enables the developer to design scalable, high-performance application code and data structures. By addressing scalability at the earliest stages of development, you can prevent expensive problems down the road. This presentation is geared toward the developer rather than the DBA.

After this presentation, you will be able to answer questions such as the following:

- Will an index help and, if so, what kind of index do I need?
- Why isn't my index being used?
- Can't I just iterate through my result set in Java, since it is so much easier than learning SQL?
- Why does my query work great on my development box but it's lousy in production?
- What is the best way to measure query performance?
- Since I'm not sure which columns I might need, why shouldn't I always just "select *"?

Transparent Application Failover for Application Developers
Mark Harrison, *Pixar* 2:30–3:30

TAF (Transparent Application Failover) is an excellent Oracle RAC feature that allows applications to survive RAC node failures. While there is quite a lot of information regarding TAF configuration and operation on the server side, there is much less information regarding client-side TAF programming. This presentation covers:

- A quick guide to configuring TAF. Everything can be done in TNS—there's no database reconfiguration necessary!
- The nine cases where TAF can affect your client-side code.
- What TAF gives you "for free" and the cases where you must account for TAF conditions in your code.
- TAF-related database and listener error codes and how to handle them.

MySQL: Would You Like Transactions with That Table?
Brian Hitchcock, *Sun Microsystems*. 4:00–5:00

MySQL is becoming more and more popular as many businesses employ this open-source RDBMS in their IT infrastructure. For DBAs that have spent years working with Oracle, what are the big differences with MySQL databases? MySQL allows the DBA to make choices that Oracle doesn't allow and, therefore, those with Oracle experience will need to adjust their expectations as they start to support MySQL. Assuming that MySQL provides the same features as Oracle can cause many problems—problems that can be avoided by knowing about these issues beforehand. ▲

NoCOUG
P.O. Box 3282
Danville, CA 94526

NoCOUG Fall Conference Schedule

November 13, 2009, at Oracle Corporation, Redwood Shores, CA

Please visit **www.nocoug.org** for updates and directions, and to submit your RSVP.
Cost: $50 admission fee for non-members. Members free. Includes lunch voucher.

8:00 a.m.–9:00	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Hanan Hit, NoCOUG president
9:30–10:30	**Keynote:** *Introducing Oracle Exadata V2*—Juan Loaiza, Oracle Corporation
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Auditorium: *Extreme Performance with Oracle Database 11g and In-Memory Parallel Execution* —Maria Colgan, Oracle Corporation
	Room 101: *What to Do When the SQL Optimizer Goes Wrong* **Editor's Pick** —Kyle Hailey, Embarcadero Technologies
	Room 102: *SQL Performance Hero and OMG Method vs. the Anti-Patterns*—Jeff Jacobs
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Auditorium: *Practical Oracle Capacity Planning*—Tanel Poder
	Room 101: *Explaining the Explain Plan*—Oracle Corporation
	Room 102: *Physical Data Storage for the Application Developer*—Dave Abercrombie, Convio
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Auditorium: *Zero Slides: The Scripts and Tools That Will Make Your Life Easier and Allow Better Troubleshooting*—Tanel Poder
	Room 101: *Database Testing: Introducing DB Infrastructure Level Change with Full Confidence* —Oracle Corporation
	Room 102: *Transparent Application Failover for Application Developers*—Mark Harrison, Pixar
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Auditorium: *Best Practices for Data Loading with Oracle 11g Release 2*—Oracle Corporation
	Room 101: *Deployment Agility Through Tier Testing*—Hanan Hit, Enteros
	Room 102: *MySQL: Would You Like Transactions with That Table?*—Brian Hitchcock, Sun Microsystems
5:00–	**NoCOUG Networking and Happy Hour hosted by Oracle.**

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

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo: Aerosmith concert at Oracle OpenWorld 2009. Photo by Erasmo Chiommino