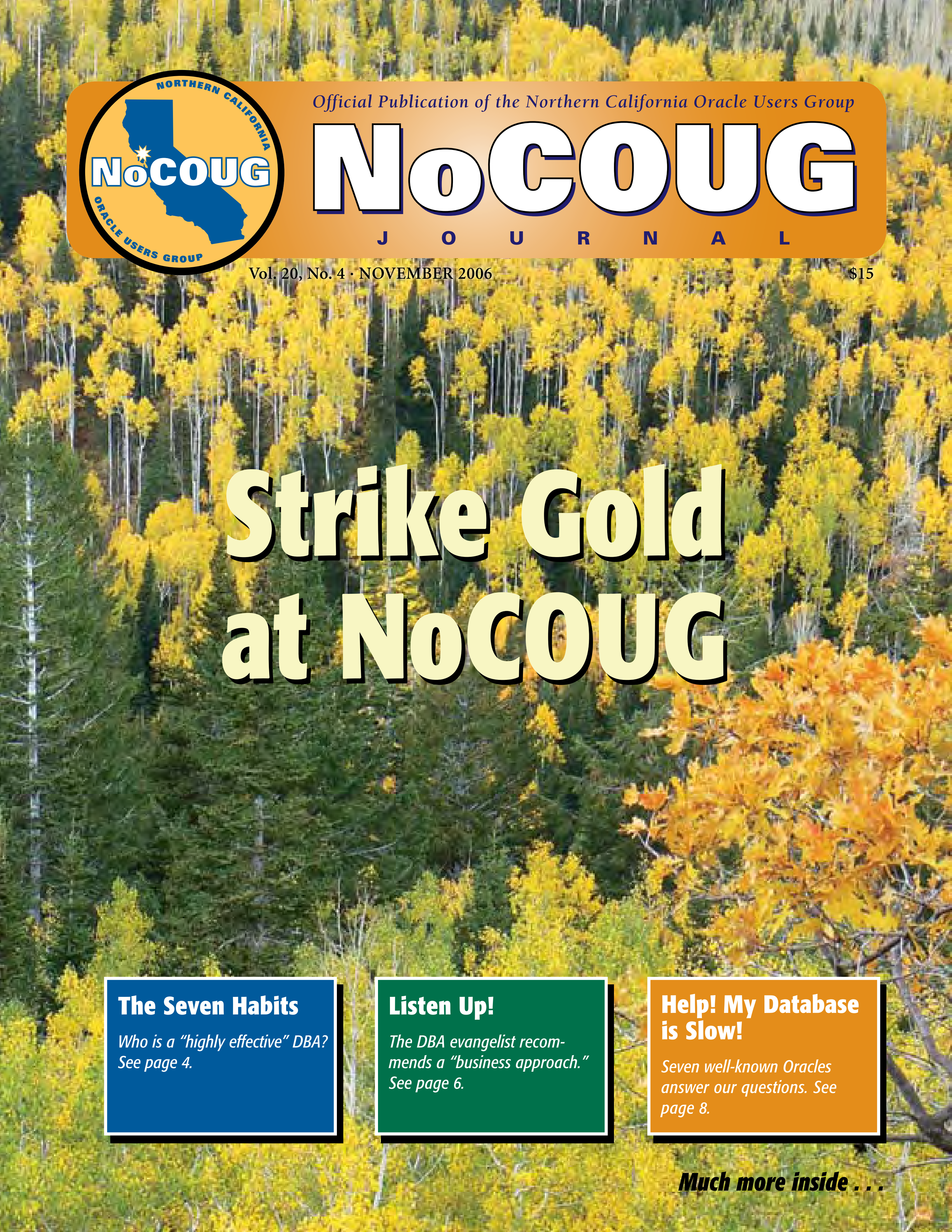
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 20, No. 4 · NOVEMBER 2006 $15



Strike Gold at NoCOUG

The Seven Habits

Who is a "highly effective" DBA? See page 4.

Listen Up!

The DBA evangelist recommends a "business approach." See page 6.

Help! My Database is Slow!

Seven well-known Oracles answer our questions. See page 8.

Much more inside . . .

How I Became the Editor of the *NoCOUG Journal*

I became a member of NoCOUG in 2004 and went to the Summer Conference at the Chevron campus in San Ramon. In his opening remarks, Roger Schrag, then president of NoCOUG, encouraged members to become volunteers. Afterwards, I mustered the courage to approach him and gave him my email address. *The very next day*, I received a welcoming email message from Roger:

Hello Ignatius,

It was nice to meet you at yesterday's NoCOUG conference. Thank you for expressing an interest in helping. We are a volunteer organization—and so it is the help of people like you who make the events a success.

Our next board meeting will be on Saturday, September 11, in Pleasanton from 9:00 am until noon. I hope you can join us as a guest. This will allow you to learn more about what goes on in the planning of conferences and events, and you can see if you might want to become a volunteer. As the date draws closer, I will email you exact directions, and a meeting agenda.

Roger Schrag, NoCOUG President

To Be Continued . . . —Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

NoCOUG BOARD

Publication and Submission Format

The *NoCOUG Journal* is published four times a year by the Northern California Oracle Users Group approximately two weeks prior to the quarterly regional meetings. Please send your questions, feedback, and submissions to the *NoCOUG Journal* editor at **journal@nocoug.org**.

The submission deadline for the upcoming February 2007 issue is December 1, 2006. Article submissions should be made in electronic format via email if possible. Word documents are preferred.

NoCOUG does not warrant the NoCOUG Journal *to be error-free.*

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

Strike Gold at NoCOUG

by Darrin Swan

Darrin Swan

The year 2006 is my sixth year of serving NoCOUG as a board member, and this is my second and last year as NoCOUG president. I have had a lot of fun and thoroughly enjoyed the role. One of my favorite parts about being NoCOUG president is being able to meet with NoCOUG members and learn about their projects and initiatives. The other is working with a great group of board members. Our conferences wouldn't be a success without their dedication.

We're in the home stretch now and, I'd have to say, I'm very happy with the outcomes the year has produced so far. We've accomplished a lot while trying new things such as Training Days and new venues. NoCOUG members learned from a top-notch lineup of Oracle experts, including many new faces. NoCOUG's incredibly dedicated volunteers and board once again organized four successful conferences while maintaining the strong financial position necessary to continue the work in 2007 and beyond. And they had a lot of fun while doing it.

But 2006 still has a lot to offer—much more education for Northern California Oracle professionals. Oracle OpenWorld begins on October 22 and NoCOUG's Fall Conference takes place on November 2.

The theme of Oracle Open-World 2006 is "Better Information—Better Results." I'm always amazed at the number of presentations at Oracle OpenWorld, and this year it will boast as many as 1,400—wow! Sir Elton John will perform live—pretty cool.

I'd like to think that NoCOUG's Fall Conference is equally exciting but on a more intimate scale. PG&E has graciously offered their facilities in San Francisco to NoCOUG for the conference, and we too have an exciting line-up of learning opportunities. Highlights include a keynote address and technical presentation by Cary Millsap. There will also be three technical tracks, vendor exhibits, book raffles, and more.

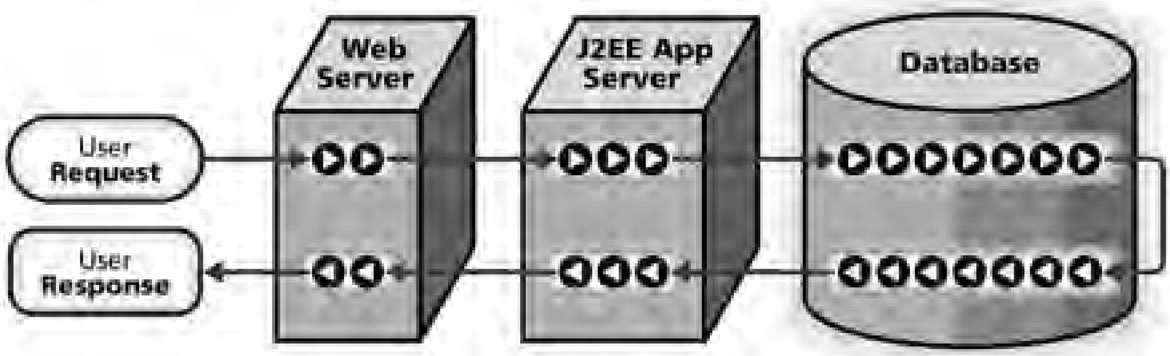
As always, a NoCOUG conference is a great opportunity to network with like-minded Oracle professionals from Northern California. And you can't beat the price—as usual there is no fee for NoCOUG members!

I hope you enjoy this issue of the *NoCOUG Journal*, and I look forward to seeing you again on November 2! ▲

Step-by-step transaction visibility for Oracle and Java.

Application Performance Management from Confio Software

Ignite™ for Oracle • Ignite™ for Java



Transaction steps: Wait-Event or Method Call

- Automates best-practice Wait-Time analysis.
- Visibility from Oracle up through J2EE Applications.
- Lightest load on monitored servers.
- Fastest to install, fastest to results.

Download free trial at www.confio.com and begin monitoring in 30 minutes.

CONFIO SOFTWARE

www.confio.com
info@confio.com
303.938.8282

Ignite IT Performance™

[illegible]

The Seven Habits of Highly Effective DBAs

by Venkat S. Devraj

Venkat Devraj

Have you seen a highly effective database administrator in action? No, I'm not referring to a DBA with the word "senior" somewhere in his title, someone who wows peers and managers periodically with his grasp of arcane command-line syntax, or someone who seems to be able to save the day with his superior performance tuning or disaster recovery skills with just the right mouse-clicks across half-a-dozen GUI tools.

I'm instead referring to the DBA whose environment rarely needs a display of his firefighting prowess. He doesn't have to get creative or think on his feet at 3:00 a.m. on Sunday. He follows a set number of processes for handling day-to-day tasks, and leverages his creativity in converting more and more ad-hoc reactions to well-defined processes. And he continues to sharpen and leverage his core skills to instrument database behavior to anticipate and avoid bottlenecks. He doesn't have to go from caffeine to crisis and back to get through the day or night. In spite of the fact that he doesn't burn 60+ hours a week to help users, they still love him. How is this possible?

Gone are the days when the DBA could sit in his cave, I mean his cube, and shoo away the users, stating that the problem "doesn't appear to be in the database."

There are certain traits that set this relatively new breed of superstar DBAs apart. Slowly but relentlessly, they are changing the norm of how DBAs are supposed to function and upgrading the mainstream perception of what makes a "good DBA." Such DBAs quickly develop a vision of what the ideal database environment should look like and work toward achieving that goal. Fortunately, being a highly effective DBA is a skill that can be acquired by today's overworked and somewhat ineffective DBAs. In a similar vein to the business blockbuster from the illustrious Stephen Covey, here are seven behavioral attributes to cultivate to gain entry into the DBA Hall of Fame.

Be an "expert generalist"

These days, you can't afford to know just the database to the exclusion of other areas. Without a doubt, you have to occasionally get deep inside the database, but in addition, you have to know all the components that the database is related to—the hardware, the operating system, the network, the storage sub-system, the application tiers—in order to understand the different maladies that each tier may face, and interpret how those maladies can adversely affect the health of the database. DBAs are expected to produce metrics across multiple tiers in order to conclusively prove a theory and avoid finger-pointing or emotional arguments. They need to know their way across Unix, Linux, and Windows. These platforms aren't mutually exclusive any longer—companies pick and chose platforms to best match specific business needs, and mixed environments are the norm. So continue to learn all you can about the database, but don't take your eyes off the other tiers. Besides basic systems administration, data modeling and development skills are almost mandatory to operate at a higher level, especially writing and tuning SQL, along with knowledge of a relevant database 4GL such as PL/SQL. If you have the opportunity, pick up skills across database platforms. Platform-independent scripting skills via Perl or Python also often come in handy.

Understand the business and user expectations

One would think this is a no-brainer, yet gaining knowledge and keeping abreast of the business often slips to the bottom of many DBAs' list, who then flounder as they face unanticipated database use and growth patterns. Get beyond a superficial level of understanding to map database transaction and workload patterns to different user activities. Such knowledge helps in the following ways:

- For new systems, it allows you to model systems infrastructure to ensure capacity is available to accommodate regular use as well as spikes.
- For existing systems, it helps you build a baseline of what the database is supposed to look like when it is

working "normally" and identify any deviations. Accordingly, you can configure alert thresholds to avoid false alerts, as well as begin proactive diagnosis and action to prevent long-term performance disruptions.

Strike up conversations with users about how they view the database and what they need to be successful in their jobs. Only after you begin to see the database from the users' eyes can you begin to work in conjunction with them and proactively address their needs. Gone are the days when the DBA could sit in his cave, I mean his *cube*, and shoo away the users, stating that the problem "doesn't appear to be in the database."

Build granular service-level agreements via quantitative and easy-to-understand metrics

Once user expectations are understood, regardless of whether there is an enterprise SLA or not, it is desirable to document them via simple but quantitative metrics such as Mean Time to Respond (to user calls and environment exceptions), corresponding status update intervals, predefined maintenance windows, response time for key application screens and reports, successful completion of the nightly load process, etc.—metrics that leave little room for vagueness.

The best-laid SLAs have user-facing "system performance criteria" that do not rely on the technical jargon that often makes SLAs useless to anyone but the person writing them. Such SLAs begin with the application and span all areas of the infrastructure stack that feed the application. The DBA has to contribute to the enterprise SLA from a database perspective. You should describe, in the users' lingo, what the database (as a component of the overall application stack) needs to be capable of doing for them at different times of the day for it to be considered business-worthy. Provide a user dashboard that is easy to use and captures these metrics effectively in near real time. That simplicity will serve you well because it will allow both the users and you to measure value on an ongoing basis, and generally be on the same page about areas for improvement without ambiguity or frustration.

Break down the environment into smaller, more manageable pieces

Even as recently as a couple of years ago, DBAs viewed change control and related processes as something that cramped their natural style. Since then, especially thanks to the growing adoption of ITIL and similar standards, highly effective DBAs are beginning to see the value of having robust processes for change control, release management, and configuration management.

However, for environments to be stable, the DBA has to get rid of the ad-hoc tendencies that prevail. Anything that touches a database with real users (production *and* non-production) within or outside the company needs to have these processes in place. Don't be content with labeling your environment merely as "production," "QA," "development," etc. Within each environment, categorize the different databases based on characteristics that are unique to your business—either based on the user base, the SLAs, or the applications that run on it. Keep the number of categories small (say, no more than half-a-dozen).

Once the categories are accepted by all relevant parties (say, the change control board, the configuration manager, etc.), standard operating procedures can be applied to each environment to dictate access restrictions, maintenance policies, standard reactions to problem events, monitoring poll frequencies, alert/escalation policies, and so on. This reduces the amount of ad-hoc decisions and guesswork during problem diagnosis and resolution.

Having such categorized policies in place pays rich dividends in terms of more controlled/stable environments, as well as reducing the labor overhead typically associated with security, compliance, and audits. For instance, for access policies to be effective in production, it would involve determining what to lock up and what to leave accessible using the right level of privileges and roles. Once the policy is implemented, there are fewer challenges in troubleshooting and figuring who did what in the database, without having to enable auditing in the database and incurring a performance penalty.

Similarly, be the gatekeeper for all SQL that gets deployed —do SQL audits via random sampling of pre-production code. For instance, search for any SQL that accesses larger data-sets (say, any table with more than a million rows) and run a quick EXPLAIN PLAN on it to ensure that the execution path is optimal. Either tune those rogue statements yourself, teach the developers how to do it, or have the Development DBA (if there is one in your environment) tune them. This activity can be backbreaking (all the more reason for teaching the developers so it gets done right the first time), but will result in happier users and fewer emergencies eating into your leisure hours.

Be focused on the task, not necessarily on how to accomplish it

If any of you are former C programmers, remember the API stack you would use to interface with that printer device? You didn't know or need to know exactly how the printer worked or what commands made it behave the way you wanted it to. All you needed to know was what API function calls to make to get the printer plugged in to your application, and those functions did the remainder of the job. Wasn't that easy? Did that not increase your productivity? Would you have been able to crank out as many applications as you did if you had to spend time learning the internals of the printer in order to get it to interface with your application? Someone else (an expert on printers) had already taken the trouble to learn all about it and code the API stack for you to use. Highly effective DBAs are similarly task focused. They learn what actions are required for a given environment, and they focus on procuring and configuring those via reusable components built by a third-party vendor (who is capable of supporting it), or by someone else within their team or in the overall DBA community. If the required components don't appear to exist, they take the time to define what they need and either build it themselves or assign the build part to others in their

(continued on page 22)

Listen Up!

The DBA Evangelist Recommends a "Business Approach"

Steve Lemme

Steve Lemme is the author of Implementing and Managing Oracle Databases *and a columnist for* Database Trends and Applications. *An Oracle Master DBA with over 15 years of experience in mission-critical Oracle architecture, Steve travels the country speaking on database management best practices to address regulatory compliance.*

Steve's day job is Director of Database Management Solutions for Computer Associates, one of the world's largest IT management software providers. NoCOUG invited Steve to speak at the Spring Conference, and I got to sit down with him at Lockheed Martin and listen to him prognosticate what the future holds for Oracle DBAs.

From DBA to Visionary

If you've ever been to Oracle OpenWorld or a large Independent Oracle Users Group conference, chances are you've had the opportunity to attend Steve Lemme's presentations. He's lively, approachable, and full of enthusiasm for Oracle technology, preaching the importance of the database and the critical role of DBAs within a company. After many years of involvement in the Oracle community, it's not surprising that he eventually found himself on the board of directors for the Independent Oracle Users Group.

Authoring a book, writing columns, giving presentations, and volunteering for the IOUG clearly takes a lot of his time. What motivates him? "I do it for two reasons," he says. "The first is my love for sharing information with others. If someone can learn from me and avoid the same mistakes I may have made, that's great. Or, if someone attends one of my sessions and then shares their perspective and I can learn something, that's great. Too often, I see the wheel being re-created by DBAs."

The second thing that motivates Steve Lemme is his need to help DBAs understand their role and importance in business. He says, "Business executives are seeking more learned DBAs who understand business—not the ones that sit and watch a console. Those are the DBAs being outsourced. In the end, profitability and problem solving matter. That is a gap that motivates me. My role at the IOUG is as an Oracle evangelist."

If someone can learn from me and avoid the same mistakes I may have made, that's great. Or, if someone attends one of my sessions and then shares their perspective and I can learn something, that's great.

There couldn't be a more perfect role for Steve. "Oracle is my hobby," he says, when someone questions if he's got other interests.

And he really means it.

"It's part of who I am," says Steve. "But, when I'm not focusing on Oracle, I'm consumed with my family. I have four kids under age 12, three girls and one boy. "

"I learned about one pregnancy when I was at an Oracle conference. I was at a UKOUG conference when one of them was born [two weeks prior to the due date]. I was joking that I might have to leave the conference any minute to get on the Concorde." His wife even came to a conference with him once, though "it's unlikely to happen again," he readily admits with the joking tone that maybe he took it too far.

All the kids have their own computer systems. They help proofread Steve's work, including the Oracle-related poster that CA produces. "Some of them might even grow up to be DBAs," he says proudly.

Steve himself has had a lifelong interest in computers. Originally interested in the manufacturing process using computers, Steve studied manufacturing engineering at Arizona State, with a minor in robotics. At some point, Apple Computer held a sneak preview on campus of the Apple Lisa, and shortly thereafter, Steve became the Apple student representative. "I was like an ambassador on campus," says Steve, who eventually got a job at Apple Computer in 1989, managing their enterprise data center.

"I learned early on that I didn't want to be a programmer. I like to manage things, not necessarily invent the scratch code," says Steve. However, working at Apple gave him an exposure to Oracle and other databases. "I picked Oracle because it needed the most help with care and feeding. It was the most complex," he says.

"I left Apple when the industry was changing and moved to Motorola in Arizona to architect the first billion dollar system. I was brought in to architect mission-critical systems and created the Oracle Center of Excellence. Downtime cost $100,000 per hour."

There will always be a need for DBAs—but DBAs will need to change with the times.

Steve later moved to a tools company named Platinum, which was acquired by CA. Steve enjoys his position as director of database management solutions for CA because he has to continually focus on the toolset, roadmap and strategy.

"I like my team. I like that customers are always challenging me with new problems. It makes the job interesting and exciting," he says. The technology itself is something he enjoys, along with the business changes. According to the company's website, "CA solves complex problems." It offers best-of-breed management solutions for the majority of database technologies in use today. CA is also involved with DB2 and SQL Server and offers a comprehensive and integrated solution for those databases. You can download the CA integrated database command center for free from **www.ca.com/unicenterdcc**.[1]

Oracle DBAs are beginning to be required to learn SQL Server and DB2.

"There will always be a need for DBAs. The role may not always be called a DBA," says Steve, ever the Oracle evangelist. But DBAs will need to change with the times. "If a DBA keeps doing the same things they did as an Oracle 7 or Oracle 8 DBA, they become expendable," says Steve.

Because CA is involved in SQL Server and DB2, in addition to Oracle, I asked Steve about them. "There are striking similarities in their challenges," he says. No one has a database that is shrinking in terms of size and complexity. No company wants to hire more DBAs to manage these growing systems, and DBAs who specialize in one RDBMS are starting to be required to learn other software. Oracle DBAs are beginning to be required to learn SQL Server, and DB2. A growing number of companies are using all three of these technologies, and DBAs need to broaden their scope.

"But, people, by human nature, are resistant to change," he continues. He wants to know how many people in IT are afraid of change and how many will step up? "A lot of people take the approach of Dilbert: Keep your head tucked and down at your cubicle and hope that no one notices you."

You can work to get 99.9999% availability, but it's meaningless if the network goes down."

But the Oracle evangelist insists that DBAs need to look outside the chain. "You can work to get 99.9999% availability, but it's meaningless if the network goes down."

"People need to think out of the box and take off the Oracle blinders. There is the Fusion Strategy. Oracle Applications now runs on other databases like DB2 and SQL Server. DBAs need to learn other database technologies, maybe not today, but in the next few years. The database is just a piece of the stack."

Clear business championship trumps any IT role.

At NoCOUG, Steve spoke on "Religion, Revelation, Revolution! Best Practices and Projects for Managing Databases." In this presentation he said, "Clear business championship trumps any IT role." I asked him what he meant by that. "The DBA that takes the initiative that helps to improve the business should be the champion. Since the DBA is so integral, their role touches so many both internally and externally. Anything that goes wrong with the database can touch hundreds or thousands of people. Some DBAs don't connect with that. For those that do, management takes notice."

Speaking of taking notice, how did Steve climb the ranks and get noticed? "I've always been a technologist and not afraid to get my hands dirty. I've managed mission-critical systems that cost hundreds of thousands of dollars for downtime. I realized that there is not a lot of recognition for DBAs, and if I could educate others on how to add value to business, I could make more of an impact." In the early days, Steve said he was reminded of Rodney Dangerfield: DBAs got no respect. "I wanted to help DBAs understand their own connection to the business and how critical their role is. That's how I moved into the management role."

It's a bird . . . it's a plane. It's Steve Lemme, "Oracle evangelist," out to motivate, share information, and help DBAs whenever he can. ▲

Interview conducted by Lisa Loper

[1] CA's Unicenter Database Command Center (Unicenter DCC) provides a unified browser-based user interface that enables DBAs to perform a variety of administration tasks across a range of database platforms, including Oracle; DB2 Universal Database (UDB) for Linux, UNIX, and Windows; DB2 UDB for z/OS; and Ingres. Unicenter DCC delivers a wide range of high-value administrative capabilities, including centralized schema management—which allows DBAs to create, alter, and rename objects specific to each type of database from a common interface. It also delivers unified cross-platform account management for granting and revoking access privileges and assigning user roles.

Help! My Database is Slow!

Ask the Oracles!

Cary Millsap: "Help! My database is slow!" There's a secret behind this statement, and it's monstrously important. Do you know what it is?

It's this: When a user says a word like database or system, she's not talking about a database or your system. She's talking about a *task*.

When a user says a word like "database" or "system," she's not talking about a database or system.

Here's the problem: To a DBA or analyst, a database is a complicated network of interrelated components. The most common metaphor for organizing those components into something comprehensible is the "performance dashboard," which technical people try to rely on especially heavily when they hear the words database and slow in the same sentence. Dashboards show all sorts of statistics about your system, like CPU utilization, I/O latencies, cache hit ratios, latch miss ratios, and redo generation rates.

When a user says a word like "database" or "system," she's not talking about a database or system.

Analysts think about this stuff all the time. But users never do.

The first step is obvious to almost anybody who hasn't "benefited" from Oracle training: Observe the user.

The first step is obvious to almost anybody who hasn't "benefited" from Oracle training: Observe the user.

When Nancy in AP says, "My database is slow," what she really means is, "When I click here, it takes too long before I get what I need."

The first step for diagnosing Nancy's problem is obvious to almost anybody who hasn't benefited from years of classical Oracle training: Observe Nancy while she executes the task she's complaining about. Classically trained Oracle analysts tend to consult their dashboards.

Now, maybe something on your dashboard will light up when Nancy does the slow thing. And maybe that light will actually reveal the cause of Nancy's problem. But sometimes, nothing lights up when she does the slow thing; and sometimes, the wrong thing lights up when she does the slow thing. In *Optimizing Oracle Performance*, there's a story on page 327 of one time when the wrong thing lit up.

Cases of wrong things (or no things) lighting up on dashboards happen more often than you think. I can't quantify exactly how often, but I can tell you that nearly every dollar I've earned since 1995 is the result of people who went the dashboard route and failed. One message I have for those who will listen is that they should stop looking at dashboards.

One message I have for those who will listen is that they should stop looking at dashboards.

One message I have for those who will listen is that they should stop looking at dashboards.

So, what should you do instead? As I mentioned, you should look at Nancy. In the case where I met my real-life Nancy, we fixed her problem without ever touching a system. It was one of those, "Ah! Don't do it that way—do it this way" problems. Of course, more often, there's something legitimately wrong with how the system responds to Nancy's click, so you'll need more data to figure out what it is.

The surprising thing about profile data is that you really don't need anything else.

The surprising thing about profile data is that you really don't need anything else.

The data that you need is profile data: the kind of stuff C programmers get when they use "gcc -pg," and it's what you get when you use Oracle's "10046 Level 12" trace facility. Raw profile data is a detailed log of how code path has consumed time. You usually don't look directly at raw profile data; you usually look at it through software called a profiler (like "gprof" and my company's *Hotsos Profiler*), which filters and aggregates profile data in useful ways to help you make quick sense of what took so long.

An application's ability to generate good profile data about itself determines how easy it's going to be.

The thing that surprises most database professionals about profile data is that you really don't need anything else to solve the vast majority of performance problems. The trick about profiling is that some applications make it difficult to generate good profile data. Oracle makes it easier with every database release, but an application's ability to generate good profile data about itself—its performance instrumentation—is a key feature that determines how easy it's going to be to make your "database" fast for all of your users. ▲

Cary Millsap is the chief technology officer of Hotsos Enterprises, Ltd., where he serves as an author, teacher, consultant, designer, and developer within the company's portfolio of software products and educational services. He wrote Optimizing Oracle Performance, for which he and co-author Jeff Holt were named Oracle Magazine's 2004 Oracle Author of the Year.

Prior to co-founding Hotsos in 1999, he served for ten years at Oracle Corporation as one of the company's leading system performance experts. At Oracle, he founded and served as vice president of the 80-person System Performance Group. He has educated thousands of Oracle consultants, support analysts, developers, and customers in the optimal use of Oracle technology through his commitment to writing, teaching, and speaking at public events.

Mogens Nørgaard: Have you ever said things like "*Oracle Support says* that version 10g R2 is full of bugs . . . ," "*Denmark loves* President Clinton," "*Microsoft says* that Linux sucks," and such?

I know you haven't. You know very well—as a thinking person—that organizations and countries can't speak (imagine what it would sound like). Individuals speak.

Whenever someone tells me things like "Oracle Support says . . . ," I ask them: "*Who* exactly says this?"—because it makes a difference whether it's Richard Powell (from UK Support) or a new trainee in Ulan Bator who writes something in an SR (or TAR or whatever).

When someone tells me their database is slow, I ask them: Is it truly everything that is slow?

And when someone tells me their database is slow I ask them: Is it *everything*—truly everything—done to the database by all applications—that suddenly runs slower? A few times in my career the answer has been a surprising "yes." In that case I know what to look for: Hardware that has stopped working, disks that have left the building, networks that have changed to half duplex, i.e., things that have a global, massive impact on your poor system.

The application decides what to execute, while the database decides how it is executed.

As I'm writing this, Anjo Kolk—the CEO of Miracle Benelux—is sitting beside me here at my oak table. Here's his reaction to the above statements:

"Remember that the application decides *what* to execute, while the database decides *how* it is executed. If the *what* is wrong, don't worry about the *how*."

Your database or your system or even your whole application is most likely not slow. The database simply decided to change the *how* for one or more SQL statements.

Why did Oracle change the *how*? Because its information and/or its rules changed. Oracle (really—its optimizer code) depends on information from five sources: rules in Oracle's source code, parameter settings, statistics (on object and system level), the data dictionary, and hints. If any of these five sources change, Oracle might (or might not) change the *how*. It's that simple.

You cannot avoid changes in Oracle's hard-coded rules and fuzzy logic—that happens when you patch and upgrade. For instance, a *lot* changed between 9.2.0.4 and 9.2.0.5. You can avoid changing parameters, statistics, and hints, thus creating greater stability. You can avoid changes to the Data Dictionary by not creating, dropping, or changing indexes, IOTs, etc. And of course hints should only be used as a very last resort anyway.

Changes for the better are good—but what if nobody asked for it? And what if they're changed back to what they used to be on the following day? Changes for the worse are bad—and nobody asked for it.

You can easily monitor changes in Oracle's *how* (a.k.a. execution plans) and compare the elapsed time spent and all that jazz. Nobody is doing it currently, so imagine the potential for cool research and fame. One of my Miracle guys—Torben Holm—has created a free tool called MirPlan

Cary Millsap will be delivering the keynote address, titled ***Why You Can't See Your Real Performance Problems****, at the Fall Conference on November 2. He will also be delivering a technical session, titled* ***Questioning Method R****. Please RSVP at* **www.nocoug.org/rsvp.html** *if you plan on attending.*

that will send you an email if a SQL statement gets a new execution plan, including information about time, number of logical IOs, plans before and after the change, and all that. It just takes snapshots of V$SQL and such. Simple stuff. Amazing insights.

Focus on avoiding changes to the how, and have the ability to see when and why the how changes.

In my opinion you should focus on avoiding changes in Oracle's how, thus creating the stability that users are craving—and at the same time have the ability to see when (and perhaps even why) Oracle changed the how—and whether it was for the better or worse. That's Change Management at the right level. And your database will "never" be slow again! ▲

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

Jeremiah Wilton: For this to be any more familiar a refrain to most DBAs, it would need only to be rephrased as "Hey! Your database is slow."

A shocking number of DBAs approach system slowness with a litany of possible causes and solutions. They check for high CPU utilization, high load averages, low free memory, locking, latches, bad explain plans, etc. The list goes on and on. I call this method the "guessing method." To hear the guessing method directly from one of its proponents, simply conduct job interviews with ten DBA candidates. Eight will be active users of the guessing method.

Using the guessing method, how can a DBA possibly check for every one of the many things that could possibly go wrong? Take the "bad explain plans" laundry list item for instance. How will the DBA know which of the thousands of cursors in the SQL area to explain? Once explained, what criteria will the DBA use to declare with certainty that the plan is "bad?" Simply put, the guessing method lacks any ability to empirically determine the cause of a problem, and serves only to make the DBA look busy during a crisis.

There is no excuse for employing the guessing method because Oracle has great instrumentation.

In the Oracle world, there is no excuse for employing the guessing method. Oracle is among the best instrumented

There is no excuse for employing the guessing method because Oracle has great instrumentation.

commercial software products available. Oracle's wait events allow DBAs to instantly see if there is a problem with response time and trace it to a specific cause. With the Active Session History (ASH) feature in 10*g*, wait information can be queried and trended over recent time to show when a problem began and ended, and also how it affected the database. ASH allows DBAs to effectively respond to that other familiar refrain, "Hey, your database was slow!"

In the majority of cases where the problem in fact lies with the database (and usually with a subset of sessions and not the whole database), simply looking at the current and recent session waits will point to the problem with no additional searching required. "Broken" plans performing full table scans will show up as excessive "db file scattered read" waits. Locking will show up as "enqueue" waits, and so on. For those aficionados of the guessing method, this begs the question: why spend your time poking around when you can go straight to the source?

There are cases of slowness where wait events can be less useful in solving the problem. Host resource starvation (CPU, memory) is one variable to rule out while investigating slowness. However, automated monitoring should always alert someone long before resource starvation affects application users. If staff is spending time looking at CPU and memory on the hosts, better-automated monitoring is probably needed.

Developers should take a page out of Oracle's book by including good instrumentation and timing.

Developers should take a page out of Oracle's book by including good instrumentation and timing.

In fairness, not all application slowness (only most) is Oracle's fault. To make rapid determinations of time spent on an application, developers should take a page out of Oracle's book by including Oracle-like instrumentation and timing. Applications and middle tiers that log timing information can be consulted during slowness to see where the application is spending time. If it is spending time querying Oracle, then the DBA can look into Oracle waits. If it is spending time on some other task, then engineers can follow the arrows to the appropriate service.

In cases of tuning slowness, an alarming lack of empiricism plagues a wide swath of Oracle DBAs. By using timing-based analysis, you can answer the simple question that is begged by every instance of slowness: "Where is all the time going?" ▲

*Jeremiah Wilton (***jeremiah@ora-600.net***) was the first DBA at Amazon.com. He is now principal consultant at ORA-600 Consulting in Seattle. An occasional speaker at NoCOUG and other conferences, Jeremiah specializes in complex recoveries, high availability, and scalability. His white papers on these and other topics can be found at* **www.ora-600.net***.*

Kyle Hailey: "Why is my database running slow?" is my favorite Oracle question. I love the challenge and pressure because I work well under pressure. The pressure can be tremendous at big Oracle sites where slowdowns can cause incredible amounts of revenue loss in mere moments. Thus anyone who can solve the slowdown quickly and efficiently steals the show and usually earns lots of money, and on top of that, it's fun.

The steps to addressing a performance problem are amazingly simple, just two steps:

1. Determine if it *really* is the database that's slow.

2. If it really *is* the database that's slow, find the bottleneck (either CPU or a Wait) and tune it. If it's a CPU bottleneck, then tune the high CPU SQL. If it's a wait event bottleneck, find and tune the specific wait event.

Determining whether the database is really slow can save much wasted time and effort.

Step 1, determining whether the database is really slow, or if it is actually the application that is slow, is extremely important and can save much wasted time and effort. The database is always the first to be blamed for a slowdown, even though most of the time the problem lies elsewhere. Proving that it's not the database has been difficult in the past, but now methodology is actually clear and simple. To prove that the database is running fine or even idle, compare the available CPU to the CPU used by Oracle, as well as the time waited in Oracle on wait events. If CPU used and time waited are both well below available CPU, then the slowdown is outside the database.

Determining whether the database is really slow can save much wasted time and effort.

Calculating the above comparison was tedious before 10*g*: install Statspack, set Statspack up to run automatically, run reports on the problem time period, slog through the 1000+ lines of data to get CPU usage (which often wasn't reliable), and sum all the waits All you need to do in 10*g* is to start up OEM and look at the performance chart. If the amount of CPU used is hitting the amount "maximum CPU" line, then the CPU is maxed out and the application needs to be tuned. To tune the application, track down the highest CPU SQL and

- Analyze performance by workload/application
- Predict performance shortfalls in the next 12 months
- Understand the effects of change: new apps, 10g, more users, new hardware...in seconds

BEZ systems The Prediction People™

quovera

Oracle Consulting Solutions Specialists

With over 10 years of Oracle consulting excellence, Quovera has performed more than 300 engagements for over 200 clients. Quovera's specialized Oracle consulting services include:

- 11i E-Business Suite implementations and upgrades
- Full lifecycle custom and web development and data warehousing
- Business intelligence, reporting and business performance management

Quovera also provides expert mentoring and training on JDeveloper, Developer and Designer with its team of renowned Oracle Press Authors and Certified Masters.

Quovera brings the experience of its Oracle consulting staff to work with you to improve your Information Technology processes and achieve your business goals.

Quovera, Inc.
800 West El Camino Real, Suite 100
Mountain View, CA 94040
Tel. (650) 962-6319 · www.quovera.com

tune them. The list of high-usage CPU can immediately be obtained by drilling down on the CPU used value in the performance chart.

Likewise if the amount of time waited in the database is large, then waits should be tuned. Large wait time is any time the wait time is as much as or higher than the amount of CPU being used. Tuning waits can be complicated and esoteric so Oracle introduced ADDM. ADDM automatically runs an analysis on Oracle bottlenecks and identifies problems and solutions. The output from ADDM can be obtained by clicking on the folders that appear at regular intervals below the performance chart.

ADDM automatically runs an analysis on Oracle bottlenecks and identifies problems and solutions.

For those of you who want to tune the wait events yourselves, either because you don't have ADDM or because you don't trust ADDM, you will need to know what each of the Oracle wait events is. Documentation on the wait events is sparse and esoteric, and even rarer to find are the methods for analyzing and correcting wait event bottlenecks. I've put together what I know on **www.perfvision.com/waits.** ▲

Kyle Hailey has been in the industry for a decade and a half. He worked for Oracle for over 16 years doing support, porting, benchmarking, and kernel development. He created tools for high-end performance monitoring such as "direct SGA attach" and interactive graphic displays of performance data. He recently redesigned the Oracle Enterprise Manager 10g performance screens, making them more graphical and wait interface centric. He is now an independent consultant on Oracle performance issues.

James Koopman: It's the Database! For sure! Always has been, always will be! Well that's what we DBAs have always been *told*. And you know something? *It always will be.* At least until we can prove otherwise. Sure, we can point our fingers at the application. *In fact*, if there weren't any applications our databases would run *super fast*! The problem isn't the database but in fact is having all these silly applications around. Add to the mix users and you have a *real* problem. I don't care what hardware you have or how you have configured your database, you can always grind it to a halt with the right query.

No matter how you have configured your database, you can always bring it to a halt with the right query.

No matter how you have configured your database, you can always bring it to a halt with the right query.

CPU taxed? Add another. Swapping memory? Add some more. Filesystem busy? Add another spindle. This is sort of the "Lone Ranger" approach. Let's just fire that silver bullet and add to the system what is lacking. The problem with this approach is that while we see the negative impact on our systems being played out in CPU, memory, or disk usage, we still haven't properly detected the root cause of why these resources are being overused. So enter, stage left, the wondrous database monitoring tools. After all you have a database problem and you need someone to tell you what it is. Am I right?

You cannot tune a system properly with just resource consumption as your criterion.

Don't get me wrong here, tools are great. I even use some now and then. So you bring up your newly purchased monitoring tool and it pinpoints a SQL statement (or two) that is either consuming resources or is contending for resources. Let me be *very* clear here. You need both! You cannot tune a system properly with just resource consumption. *And,* you cannot tune a system properly with just resource wait events.

You cannot tune a system properly with just resource consumption as your criterion.

For a simple example, let's just say that you look at your resource wait events and you see nothing at all. You might think you have unlimited availability and nothing is wrong. But in fact you might have a CPU, memory, or disk subsystem that is completely consumed. Yes, you will see wait events skyrocket if you put another application into the mix. But isn't that too late?

Okay. You have used your database monitoring tool. Great! Now go fix it . . . wait . . . fix what? . . . I'm scared . . . UGH!

If a subsystem is already consumed, waits will sky-rocket if you add another application—too late!

If a subsystem is already consumed, waits will sky-rocket if you add another application—too late.

This is the trap that is quite often found when I am pulled into "tuning opportunities." There is a perceived problem with a database system but we are uncertain as to what we should do. We are apprehensive because we don't really know if any particular SQL statement is good *or* bad. We typically don't have any historical evidence that verifies this SQL to be good *or* bad. We are apprehensive because we have no idea what other applications or processes have or should be running at the same time as this SQL statement. We don't know if something else is running poorly and impacting this SQL.

Likewise we don't know if something is running better and impacting this system. You see, any application may run faster or slower at any given time and be at different code points. Find two incompatible code points across your applications and you have disaster.

What does all this mean? Know thy system and application! ▲

James F. Koopmann is founder of Pine Horse, Inc. **www.pinehorse.com**. *James is an accomplished author and has worked with a variety of database-centric software and tools vendors as strategist, architect, DBA, and performance expert. James can be contacted at* **jkoopmann@pinehorse.com**.

Guy Harrison: It's commonplace for Oracle DBAs to be called in to fix a vaguely defined database performance problem—"the database is slow." While we might dream of a world in which all applications and databases were instrumented or monitored so that users were able to precisely identify exactly which transaction or resource was slow, Oracle performance specialists have to be prepared to deal with general assertions about database performance.

We dream of a world in which applications were instrumented so that we can identify the slow resource.

"The database is slow" most often means that some application that uses the database is slow and the end user has reached the conclusion that the database is the root cause. Indeed, our experience tells us that the database is as likely as not part of the problem, but there is always the possibility of an innocent database unjustly accused. In short, don't assume that the database is the cause of all application performance problems.

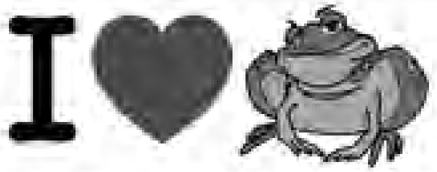
We dream of a world in which applications were instrumented so that we can identify the slow resource.

Your approach to dealing with a database performance problem may differ depending on whether the "slowness" is reported for isolated activities or whether all database activities show a slowdown. Isolated performance problems suggest issues with individual SQL statements and tracing may be the preferred diagnostic tool. Problems with all database activities may indicate a global issue with database load or contention that might require a wider diagnostic approach.

Your approach depends on whether the slowness is reported for isolated activities.

Your approach depends on whether the slowness is reported for isolated activities.

(continued on page 20)



We Know you Love Toad™

Now fall in love with our other Oracle solutions.

Whatever you need for Oracle — from development to performance tuning and management — Quest Software has an industry-leading solution for you.

So if you love Toad, we know you will adore our other Oracle solutions too.

Download our new technical brief, "Quest database management: At the center of the Oracle universe" at **www.quest.com/NoCOUG** and get a **free "I 'Heart' Toad" bumper sticker.**

QUEST SOFTWARE®

[illegible]

VERICENTER®

Your On Demand IT Resource.

VeriCenter and Our Application Partners Deliver Mission Critical Oracle Applications in a Hosted Model with Enterprise SLAs.

Enterprise Hosting | Infrastructure Outsourcing
Messaging & Collaboration | Colocation
Utility Computing | Disaster Recovery

West Coast Regional Director, Rahul Bakshi: 415-568-2182 or email at rbakshi@vericenter.com

www.vericenter.com | info@vericenter.com

Atlanta • Boston • Dallas • Denver
Houston • Minneapolis-St.Paul • San Francisco

Ignite for Java

by Jay Bright and Iggy Fernandez

Introduction

At the IOUG conference in Orlando, FL, in 2005, Cary Millsap delivered an important talk, titled "How to Make an Application Easy to Diagnose." You can download the white paper from **www.hotsos.com**, and this is what it says in conclusion:

> *You can't manage what you can't measure. For many applications, this means that **you can't manage performance**. [Emphasis ours] It can be easy to measure application performance **if** [emphasis ours] the application participates in the measurement. . . . By attacking the measurement problem intelligently in the application 3GL code, the application performance measurement problem goes away, yielding **faster** [emphasis ours] software that's cheaper to diagnose and repair, which is good for customers and vendor alike.*

If you don't understand why instrumenting application code will make software faster, not slow it down. consider this passionate plea by Tom Kyte (**tkyte.blogspot.com/2005/06/instrumentation.html**):

> *To the developers that say "this is extra code that will just make my code run slower" I respond "well fine, we will take away V$ views, there will be no SQL_TRACE, no 10046 level 12 traces, in fact–that entire events subsystem in Oracle, it is gone." Would Oracle run faster without this stuff? Undoubtedly–not. It would run many times slower, **perhaps hundreds of times slower**. [Emphasis ours] Why? Because you would have no clue where to look to find performance related issues. You would have nothing to go on. Without this "overhead" (air quotes intentionally used to denote sarcasm there), Oracle would not have a chance of performing as well as it does. Because you would not have a chance to make it perform well. Because you would not know where even to begin. So, a plea to all developers, get on the instrumentation bandwagon.*

But can't you get the profiling information you need from Oracle itself? In the same blog entry, Tom says:

> *Commonly requested information–Tom, can you tell me what my average transaction response time is? Answer: **nope, no clue, not a single clue**. [Emphasis ours] The only thing I can tell you from the database is on average how long individual bits of SQL might have taken. I don't know what a transaction is to you and besides, I would tell you only about the database component–no network, no application time, just the database.*

The Confio Solution

Confio Ignite™ for Java, part of the Confio Igniter™ Suite, identifies and exposes performance problems across the Web, Java, Oracle, and application tiers. Ignite for Java gives DBAs, application owners, and Java/J2EE system administrators a deep view of performance and of interaction between the J2EE application server and the Oracle database.

> *Ignite for Java helps the IT team find the most critical problems that keep end users waiting on their web requests, and identifies exactly where in the Java application or Oracle database the problem resides.*

Ignite for Java uses Response-Time and Wait-Time methodology, the industry's best practices for performance optimization. It focuses on time, not system operating statistics, as the most important indicator of performance. While memory and CPU utilization are important for operating a Java server, they do not directly affect the end-user experi-

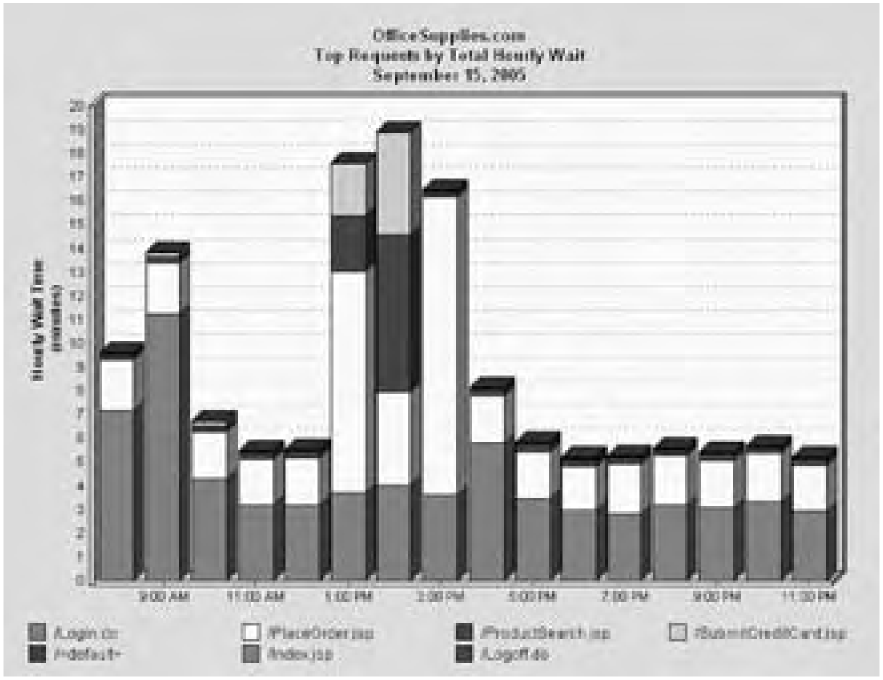


Figure 1—Web Requests Showing Accumulated Wait Time

ence. Focusing on time to complete specific web requests, and breaking the request time down into individual Java method calls, JSP page requests, and Oracle SQL, gives the IT user an exact picture of where and why an application is performing poorly.

Wait-time analysis using the example in Figure 1 illustrates the value in finding problems that affect end-user service. In this example, the application owner for the "OfficeSupplies.com" online store would see that accumulated wait time peaked in the 1:00 to 3:00 period, and then subsided. A specific problem arose with the SubmitCreditCard screen. From here, the performance analyst could drill down to find the source of the specific problem that occurred during this period and whether it is caused by lengthy SQL processing time (see Figure 2 for an example).

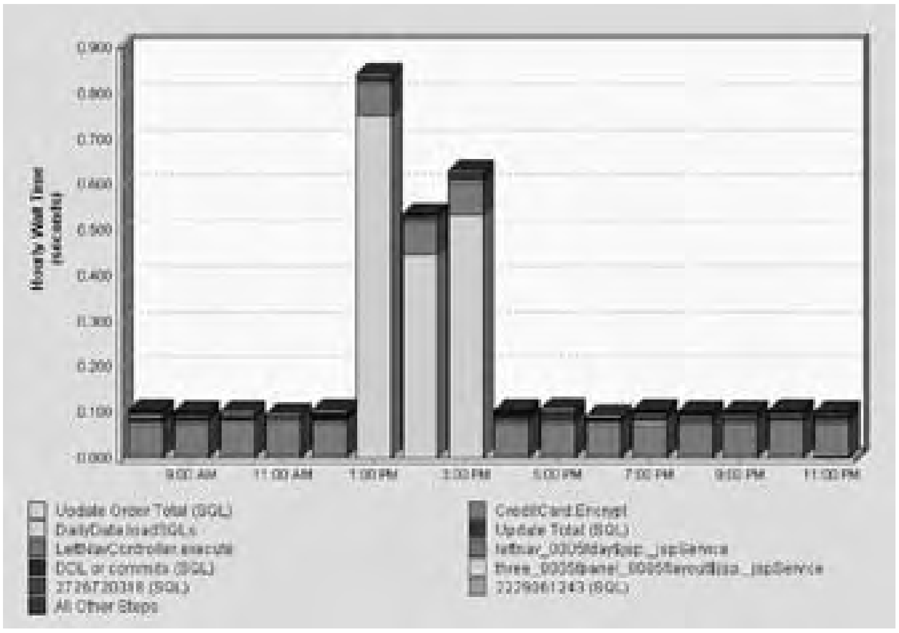


Figure 2—Response-Time components for a specific Web Request

Ignite for Java Capabilities

- ➤ Capture response times for each Java method, number of times called, and average response time.
- ➤ Use a simple drill-down interface to follow each Java request and determine the Oracle response time when Java JDBC invokes SQL or PL/SQL.
- ➤ Capture end-to-end response time details from Web Server requests–down through the Java application layers and all the way to the Oracle database.
- ➤ View the Java call stack showing how the Java application's time is divided into context, class, and methods, and where methods are dependent on Oracle to complete the request.

Byte Code Instrumentation

Confio's Ignite for Java uses Byte Code Instrumentation (BCI) to gain visibility in the J2EE application layers. BCI is a mechanism used by Java tools such as compilers, optimizers, obfuscators, code generators, verifiers, and analyzers.

In BCI, tracker bytes are inserted into the chosen compiled Java methods to signal the start and completion of execution. By watching these trackers, Ignite for Java is able to identify exactly how long each method requires to complete and how long it is waiting on JDBC calls to the database. No source code is viewed or modified. The method trackers are inserted into selected components of the compiled application at the level of the jar, ear, or war file.

- ➤ Instrumentation can be inserted into selected compiled Java methods by choosing specific class files.
- ➤ Instrumentation can be set at the class, jar, war, ear, or directory levels.
- ➤ Instrumentation is self-calibrating, allowing Ignite for Java to identify which Java methods need to be explicitly tracked and which can be included in the wait time of the parent method to eliminate the load on the application server. Auto-throttling for low risk and low impact on the Java Application Server. Appropriate for 24x7 Production Monitoring.
- ➤ Java method execution is correlated with JDBC/SQL calls.
- ➤ Class file pre-processing. No runtime intrusion and startup delays.
- ➤ No real-time load is imposed by dynamic runtime insertion of bytes.

Summary

Ignite for Java is designed for DBAs, Java application owners, and developers to gain deep visibility into their applications. It uses an intelligent instrumentation method, Byte Code Instrumentation, that can identify wait time in the application and still be light enough to use in production environments. ▲

To obtain a free trial copy of Ignite for Java, submit a request at **www.confio.com/javatrial** *or contact a Confio representative directly. Once your monitors have run for several days, Confio provides a free performance consultation to examine your collected data and highlight performance changes and specific expected improvement based on the Ignite for Java analysis.*

SQL Sucks! – Part III

by Iggy Fernandez

Then felt I like some watcher of the skies;
When a new planet swims into his ken;
Or like stout Cortez when with eagle eyes
He star'd at the Pacific–and all his men
Look'd at each other with a wild surmise–
Silent, upon a peak in Darien.—**John Keats**

Iggy Fernandez

Recap

In the first installment of this multi-part essay, we explained that SQL is what is called a "non-procedural" language; i.e., an SQL query simply identifies a subset of data in the database without specifying *how* to go about extracting that data. This has led to the pervasive belief that application programmers have no responsibility for SQL performance. In the second installment of this essay, we explored the theoretical underpinnings of the SQL language–an understanding of which is crucial to taking responsibility for SQL performance. In this installment–the last–we consider our options.[1]

The Horseless Carriage

In Part II of this essay, we listed a number of "dangerous" beliefs centered around the theme that application programmers have no responsibility for the efficiency of the SQL statements they write. An example of a belief that we labeled dangerous is the pervasive belief that application programmers may deploy SQL statements without carefully tailoring them to make use of existing indexes and that database administrators should set traps for SQL statements that have been deployed and that are performing badly and determine what new indexes should be created to make them perform better. Of course, I have to admit that I have been as guilty as anyone else in this regard. I am trying hard to break the habit but very old habits die very hard. I have many long years of experience in writing SQL statements that look like "pretzels created by somebody who is experimenting with hallucinogens"[2] but very little in writing SQL that is guaranteed to perform well.

I now see that writing SQL statements without regard to indexes and other access paths is like unyoking the trusty steed from the carriage and trusting that Oracle will find us a speedy stallion when we need one–not the old gray mare who "ain't what she used to be many long years ago." In the pre-relational days of yore, programmers were keenly aware of the available indexes. In fact the indexes were built into the data files in a manner similar to Oracle's Index Organized Tables (IOTs), making it extremely difficult to add new index paths. Here is an example of a COBOL data file that is indexed by EmployeeID and EmployeeLastName.

```
SELECT EmployeeFile
 ASSIGN TO "EMPLOYEE.DAT"
 ORGANIZATION IS INDEXED
 ACCESS MODE IS DYNAMIC
 RECORD KEY IS EmployeeId
 ALTERNATE RECORD KEY IS EmployeeLastName
WITH DUPLICATES.
```

There are only three things you can do with the file described above. You can perform a full scan of the file, you can retrieve the unique record whose EmployeeID value matches the value you specify, and you can retrieve records whose EmployeeLastName value is greater than or equal to a value you specify–we won't bore you with the details.

Fast-forward to the wonderful world of relational databases where programmers have been raised to ignore such niceties as performance. Edgar Codd, the inventor of relational theory, insisted on "Physical Data Independence"–but what exactly did he say? Referring to his magnum opus *The Relational Model of Database Management–Version 2,* we find the following rule in Chapter 20–*Protection of Investment.*

RP-1 Physical Data Independence–The DBMS permits a suitably authorized user to make changes in storage representation, in access method, or in both–for example, for performance reasons. Application programs and terminal activities remain **logically** [emphasis ours] *unimpaired whenever any such changes are made. (This feature is Rule 8 in the 1985 set.)*

In Chapter 21-*Principles of Database Design*, we find the following rule.

RD-10 Changing Storage Representation and Access Options–Commands must be available to the DBA for dynamically changing the storage representation and access method in use for any base relation without causing **logical** [extra emphasis ours] impairment of any transaction in **source code form** [extra emphasis ours] *(whether already compiled or not), or any noticeable*

[1] You can download Part I and Part II of this essay from **www.nocoug.org/presentations.html**.

[2] A colorful description coined by Steven Feuerstein in an interview published in May 2006 in the *NoCOUG Journal.*

delays in the execution of the transactions in progress or of transactions waiting to be processed.

My interpretation of these rules is that Codd would not object if the relational language allows the programmer to explicitly specify an access path (such as an index) as long as the relational language program remains *logically* unimpaired if the access path were to suddenly become unavailable[3]–in fact Codd seems to anticipate that relational languages might allow the programmer to do so. As a historical note, we might add that Codd was not involved with the development of SQL and has many critical things to say about SQL in his book.

All Oracle Hints meet Codd's requirement that an SQL query not be invalidated if an explicitly specified access path suddenly becomes unavailable. The purist in us might object to referring by name to an index that might be dropped at a future point in time and this is no longer necessary in Oracle 10*g* which allows us to specify an indexed access path by listing the *columns* which the index contains instead of the *name* of the index.

A Delicate Question

Does the optimizer need a hint? Experience tells us it often does. In Part I of this essay, we discussed the problematic assumption of attribute independence and the unsatisfactory assumptions that the Oracle optimizer is forced to make to determine how many rows will result from the joining of two tables. Another hard problem for the optimizer to solve is that of determining the equivalence of SQL statements. If two SQL statements are equivalent then the query execution plan that is optimal in the case of the first is also optimal in the case of the second. However, if Oracle cannot recognize their equivalence it is less likely to identify the same plan in both cases. In a 1988 paper,[4] Fabian Pascal wrote a query in seven different ways and checked how long it took Oracle to retrieve the data in each case. These were his results.

ACCESS METHOD	COMPLETION TIME
JOIN	19 seconds
IN1	22 seconds
ANY1	20 seconds
IN2	525 seconds (25X)
ANY2	525 seconds (25X)
EXISTS	525 seconds (25X)
COUNT	1818 seconds (90X)

I repeated the same tests recently and obtained the following results. Completion times were predictably faster because of faster hardware but the variations remained, indicating that the problem remains as hard as ever.

[3] See, however, Rule RD-2 in Chapter 21—*Principles of DBMS Design* read together with Codd's remarks on optimizability in Chapter 26—*Advantages of the Relational Approach.*

[4] **www.dbdebunk.com/page/page/1317920.htm**.

Lies, Damn Lies, and SQL!

In the last issue of the *Journal* we offered a prize of a SanDisk Sansa M240 1 GB MP3 Player for the best solution to the following puzzle sent to us by Sumit Sengupta from Columbus, OH.

Describe a combination of circumstances in which the COUNT function could produce the anomalous results seen in the example below.

```
SQL> DESCRIBE employees;

Name                  Null?     Type
--------------        --------  ---------------
NAME                  NOT NULL  VARCHAR2(25)
SALARY                NOT NULL  NUMBER(8,2)
COMMISSION_PCT        NOT NULL  NUMBER(4,2)

SQL> SELECT COUNT(name) FROM employees;

COUNT(NAME)
-----------
          3

SQL> SELECT COUNT(salary) FROM employees;

COUNT(SALARY)
-------------
          2

SQL> SELECT COUNT(commission_pct) FROM
employees;

COUNT(COMMISSION_PCT)
---------------------
          1

SQL> SELECT COUNT(1) FROM employees;

COUNT(1)
----------
          0
```

Chris Lawson and Roger Schrag submitted correct solutions to the puzzle. Chris Lawson's solution took advantage of Oracle's read consistency scheme and posited that a second session deleted one record (and committed its work) after each of the above queries. Roger Schrag submitted a solution that uses materialized views and does not require coordination with a second session—though it does require that all the rows in the table be deleted. The original solution provided by Sumit Sengupta does not require the deletion of rows and is based on the column level access control provided by Oracle 10*g* R2.

Roger Schrag wins the prize and we will discuss all three solutions in detail in the next installment of the SQL Corner. ▲

ACCESS METHOD	COMPLETION TIME
JOIN	0.0167 seconds
IN1	0.3959 seconds (25X)
ANY1	0.3907 seconds (25X)
IN2	0.4110 seconds (25X)
ANY2	0.4140 seconds (25X)
EXISTS	0.7291 seconds (40X)
COUNT	0.6991 seconds (40X)

The Robust Plan

But doesn't the best choice of query plan depend on the values of the bind variables? Yes, of course it does. However, remember that Oracle caches query plans and reuses them whenever possible, even when the values of the bind variables have changed–the aim being to avoid the overhead of repeated parsing of similar statements. Oracle constructs a query plan using the values submitted in the very first invocation of the SQL statement–this is called "bind variable peeking."[5] Obviously a query plan constructed using one set of bind variables is not necessarily ideal for all other values of the bind variables. In his book *SQL Tuning*, Dan Tow develops the theory of the "robust" execution plan. Such plans have only moderate sensitivity to the values of the bind variables (which implies that they continue to perform well as your tables grow) and their cost is proportional to the number of rows retrieved. They may not be the most efficient plans but they are not very likely to be far off the mark. Such plans are conducive to stability and predictability and are obviously very desirable. Refer to Tow's book for more details.

Hinting to the Max

Having perhaps persuaded the gentle reader that Hints are perhaps not such a bad idea after all, we present a technique that might be called "Hinting to the Max." We assume that the reader is conversant with the subject of Hints and their use and with access methods such as "nested loop join," "merge join," "hash join," "semi join," and "anti join." We use the relatively new technique of "subquery factoring" to achieve the intended results via a series of simple steps. At each step, we use Hints to define the access path we wish Oracle to take. We use the same Supplier-Parts example that we dissected in Part II of this essay. The list of suppliers who supply all parts can be retrieved using the following SQL statement which we provide without further explanation. We don't attempt to justify the access path chosen at each step because this example is only intended to illustrate how a query can be systematically broken up into smaller parts and systematically hinted.

```
with
 AllCombinations as (
    select /*+ NO_MERGE ORDERED FULL(Suppliers)
FULL(Parts) USE_NL(Parts) */
       SupplierName,
       PartName
    from
       Suppliers,
       Parts
 ),

 InvalidCombinations as (
    select /*+ NO_MERGE */
       SupplierName,
       PartName
    from
       AllCombinations
    where (SupplierName, PartName) not in (
       select /*+ INDEX(SuppliedParts
(SupplierName, PartName)) NL_AJ */
          SupplierName,
          PartName
       from
          SuppliedParts
    )
 ),

 UnwantedSuppliers as (
    select /*+ NO_MERGE */
       SupplierName
    from
       InvalidCombinations
 ),

 WantedSuppliers as (
    select /*+ NO_MERGE */
       SupplierName
    from
       Suppliers
    where SupplierName not in (
       select /*+ HASH_AJ */
          SupplierName
       from
          UnwantedSuppliers
    )
 )
select * from WantedSuppliers;
```

[5] Prior to 9*i*, Oracle *ignored* the values of the bind variables when choosing a query plan. In effect, this meant that Oracle ignored any histograms that the database administrator might have created for the tables in question.

Take Home Message

The *promise* of relational technology was that application programmers would be relieved of the responsibility for the efficiency of the SQL statements they write. We have argued that this promise is a *failed promise* and that there are tremendous barriers preventing it from ever being fulfilled. And so, therefore, our "take home message" is that *SQL performance requires conscious effort on the part of the developer*!

Concluding Remarks

I am aware that the positions adopted in this essay are not generally held in favor. I have only recently begun to favor these positions and, therefore, it is very possible that I have overlooked some good arguments both for and against them. I am also aware that the documentation of Oracle Hints is quite sparse, which makes it difficult to learn how to use them. Be as it may, I enjoyed writing this essay and welcome your comments.

Further Reading

Dan Tow's book provides guidance in manually designing a query execution plan. Another excellent book on SQL tuning is *Oracle SQL High Performance Tuning* by Guy Harrison. ▲

The author can be reached at **iggy_fernandez@hotmail.com**.

A Flowchart for Success?

By Chris Lawson

Chris Lawson

Years ago, a large corporation issued a notebook to all the engineers. The purpose of the notebook was to illustrate sound approaches to problem solving. The first diagram was a flow chart illustrating the steps that an engineer should follow to solve problems. The first box said "TEST SYSTEM." The second box said "DOES THE SYSTEM MEET REQUIREMENTS?" A "Yes" response led to the box labeled "DONE." A "No" response led to the third box, labeled "REDESIGN SYSTEM."

As you might expect, we laughed at the absurdity of using such a flowchart. This example is absolutely true. I mention it because it illustrates the absurdity of trying to reduce complex processes to a single "flow chart of success."

The Performance Analyst's Job

Making SQL run faster is indeed a big part of performance analysis, but it's sometimes (mis)represented as the only part. This notion is wrong because the process of tuning SQL presupposes the form of the solution. That is, the act of tuning SQL is based on the idea that slow SQL is the root cause of the problem.

Slow running SQL is not always the root cause of performance problems.

Of course, slow running SQL is not always the root cause of performance problems. For instance, we might need to ask why an application is running certain SQL. Did the designer make a poor design decision that requires the program to run millions of unnecessary transactions?

A Silliness Detector?

Several years ago, I assisted a group of report designers with a poorly running report. I noticed that the exact same long-running query was run multiple times. I pointed this out to the designer, suggesting that he remove the redundant queries. Ignoring my advice, the designer instead installed a performance tool to tune the SQL. He slightly improved the efficiency of each redundant execution, but missed the whole point.

In the above example, the report designer did exactly what the tool told him, instead of stepping back and looking at the big picture. He presupposed the form of the solution, which led him to do something silly.

Here's another example–one that's much more subtle: The DBA team at a large insurance company isolated a problem query, which checked the last 12 months of payments. The key to resolving this issue was not how to make the query run faster; rather, it was why the program needed to check an entire year of payments. In this case, the issue was not even a technical issue–it was the business requirement driving the query. The business need was overly broad, causing the database work to be much more extensive than necessary.

We might need to ask why an application is running certain SQL.

Complex Problems Require Flexible Approaches

Clearly, "one size fits all" algorithms or tools rarely work for complex tasks in huge organizations. Just like the silly flow chart that purported to solve all problems, there isn't a "flowchart to success" for the performance analyst.

That's not to say that using tools is always a bad idea. Indeed, tools can be helpful in checking issues such as missing indexes, bad statistics, etc. These types of tools can certainly solve some types of problems–but their method is not universally applicable; i.e., their scope is limited.

I have found that few problems are efficiently resolved by relying solely on one approach. The best DBAs and performance specialists I know use a tool kit of different approaches. For any given assignment, these experts adapt their heuristics to fit the circumstances. If necessary, they develop new approaches or create different scripts. ▲

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

(continued from page 13)

If an isolated set of activities or transactions is experiencing performance problems, you should try to acquire a SQL trace if at all possible. In recent releases of Oracle, this can be achieved using the DBMS_MONITOR package. Trace files can be analyzed using "tkprof." I've written a short "quick start" article for tracing at **guyharrison.typepad.com/oracleguy/2006/09/10g_tracing_qui.html**. Your goal in tracing is to identify individual SQL statements that are dominating elapsed time, and either tune them as appropriate (add an index, for instance) or examine their wait events to determine any contention that they might be experiencing (locks, for instance).

If the problem appears to be affecting all or most database operations, there may be a more global performance problem. In ancient times when I was a young DBA, we would examine certain ratios such as the "buffer cache hit ratio" or the "latch sleep ratio" to try to find inefficiencies within the Oracle server. In modern times, most of us will instead interrogate the wait interface as exposed in views such as V$SYSTEM_EVENT and the 10*g* "time model" as exposed in V$SYS_TIME_MODEL. You can find my favorite query for extracting this high-level information at **guyharrison.typepad.com/oracleguy/2006/09/10g_time_model_.html**.

The time model and wait interface focuses on the areas where the database spends the most time.

The time model and wait interface focuses on the areas where the database spends the most time.

The beauty of the time model and wait interface is that it allows us to focus on the areas where the database is spending the most time. However, an overly simplistic analysis of this data can result in misdirected tuning efforts. For instance, a single "missing" index might magnify I/O requirements by an order of magnitude, overloading the disk subsystem and causing the wait interface to report a massive disk bottleneck. However, the disk bottleneck is the symptom, not the cause.

To avoid treating symptoms rather than causes, you need to address symptoms systematically.

To avoid treating symptoms rather than causes, you need to address symptoms systematically.

To avoid treating symptoms rather than causes, you need to address symptoms systematically. In short, stage your tuning efforts so that you address the following in order:

- ➤ First, reduce application demand on the database by tuning SQL and PL/SQL.
- ➤ Then, eliminate contention within the database that might be preventing the full scope of the application demand from being realized. This means reducing time spent waiting for latches, locks, buffers, and so on.
- ➤ After that, reduce the amount of logical IO that turns into physical IO. Primarily, this is done by allocating memory to the SGA and PGA. Allocating memory to the PGA prevents hash joins and sorts from doing physical IO, while allocating memory to the buffer cache reduces the amount of logical reads that turn into physical database reads.
- ➤ Finally, configure your disk subsystem to meet the now optimized physical IO demand by providing enough IO bandwidth (i.e., individual disk devices) and spreading the IO demand evenly across these devices—usually by striping (though virtually never by using RAID5). ▲

Guy Harrison is chief architect for database solutions at Quest Software. He is the author of Oracle SQL High Performance Tuning, Oracle Desk Reference, *and* MySQL Stored Procedure Programming, *as well as numerous shorter articles and presentations. Guy can be reached at* **guy.harrison@quest.com**.

Chris Lawson: Vague comments about database slowness are common and understandable complaints. It's simply not fair to expect database users to analyze and correct database performance issues on their own; rather, they just know their application is slow and want some help! When faced with complaints like these, don't despair—it's easy to help customers who are willing to admit they don't know the cause of the slowdown. In fact, the really tough challenge is helping those who think they have it all figured out.

So where do we begin? How do we cope with such vague generalities as "My database is slow"?

The performance specialist is a "quick change" artist, using different tactics at each stage of the process.

The performance specialist is a "quick change" artist, using different tactics at each stage of the process.

I like to think of the performance specialist as a "quick change" artist, using different tactics at different stages of the process. Like an actor, the performance expert dons different hats and plays different roles at different times. I call this procedure the "Physician to Magician" approach. Here are the steps:

1.	*Physician:*	Listen to the "patient"
2.	*Detective:*	Gather clues
3.	*Pathologist:*	Isolate the root cause
4.	*Artist:*	Create a solution
5.	*Magician:*	Implement the fix

Let's take a look at each step and see how it works:

Physician: When we visit a doctor, the doctor soon asks, "What seems to be the trouble?" In medical clinics, this is called the "chief complaint." The DBA doctor must do the same thing. We must ask the users, "What is the complaint?" Then we must follow up to get the details. For instance, we may ask the question, "How long is the query delay?" or "When does the problem occur?"

Detective: After defining the problem, the next step is investigation. Here, the DBA changes hats from physician to detective. The main objective in this step is to re-create and quantify the problem. Some questions that should be asked are, "What is the elapsed time of the query?" or "How many disk/logical reads are performed?"

Watch the end user as he executes the program. Get to know a little about how the application works.

In some cases, it is best to watch the end user as he executes the program. Get to know a little about how the application works. A side benefit of this is that the user will understand that you are serious about solving the problem and will appreciate your interest.

Pathologist: Now that we have identified what the problem is and have quantified the problem, we are ready to find the root cause. In this role, the DBA wears the hat of a pathologist. We try to find the disease that is the primary cause of the performance problem. Note that there is no need to guess at the root cause. In fact, speculation not based on the facts should be discouraged. Instead, results from the previous steps will help focus attention on the key problem areas.

Artist: Instead of analyzing, we now synthesize. Now is the time to be creative and imagine the perfect solution that addresses the root cause in the simplest fashion possible. Is the solution just a matter of adding a missing index, or is a more complex solution required?

Watch the end user as he executes the program. Get to know a little about how the application works.

Magician: This is a simple and fun step. All the hard work has been done, and we have a solution that truly addresses the performance bottleneck. It's a good idea to test the solution in a sandbox and also to clearly document the solution with "before" and "after" statistics.

It's a good idea to test the solution in a sandbox and to clearly document it with before-and-after statistics.

A good DBA will wear many hats in the course of solving performance problems. Beginning as a kindly doctor treating a sick patient, the DBA will in turn transform into a detective, pathologist, artist, and then, finally, a magician.

So, the next time you hear, "Help! My database is slow," please don't fret. Just get out your black doctor's bag and ask, "Now what seems to be the trouble?" ▲

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

When you are ready to choose who to partner with for your Consulting and Support Services needs...

We have the solution for you!

ITConvergence

EXPERIENCE the power of results

Visit our web site at: www.itconvergence.com

Pioneering the Field of Capacity Optimized Storage

The Data Domain DD200 Restorer:

A disk-based recovery storage appliance designed for Oracle database backups:

- Works with leading backup software vendors and RMAN
- Patented Capacity Optimized Storage technology that enables long term data protection at low cost
- Data Invulnerability Architecture ensures recoverability of backup data
- Rapid disaster recovery via cost effective site-to-site data replication

Data Domain®

Next Generation Data Protection

www.datadomain.com
1.877.622.2587

(continued from page 5)

team, ensuring that the output is well documented for adept handling of that area by other DBAs in the future.

Segregate and delegate the mundane

Once the mundane areas are segregated and documented, highly effective DBAs can either automate those tasks, assign them to the less-experienced DBAs in the team, or outsource them to companies that specialize in this area and can bring economies of scale to bear. Highly effective DBAs know that their bandwidth is scarce and the value-added projects they need to really focus on tend to be time critical. The faster they can get these mundane tasks delegated, the sooner they can begin spending time on the higher-value areas.

Weed out the one-trick ponies

Many environments attract a plethora of point solutions (tools that do just one thing well; examples are database monitoring tools, SQL tuning tools, and many other DBA GUI tools). Such tools tend to make the environment complex by making too many options available to do a finite set of tasks, and reduce any chances of universal adoption of one tools platform by all DBAs in the team. Moreover, since environment-specific mundane tasks are not covered via these generic tools, DBAs often are left to their own devices to tend to the workload. As a result, DBAs build their own scripts to do things their way, leading to an unmanageable array of practices across the organization—some good, some not—unnecessary duplication of effort, and a complex work environment.

One of the foremost goals of the highly effective DBA is to simplify and standardize the environment. One way to achieve that is to identify, evaluate, and consolidate functionality provided by many of the point solutions into a unified tools platform, and negotiate a maintenance and support contract with the vendor to include integration with those enterprise tools and processes that cannot or should not be weeded out.

In Conclusion . . .

I have watched DBAs implement each and every one of the recommendations here, and they have proven to be effective. I would like to get your feedback about how you operate and whether you feel that these suggestions apply to your environment. And if you try these steps, I would love to know what kind of results you get. Email me at **dbafeedback@StrataVia.com**.

Venkat S. Devraj is the co-founder and chief architect of StrataVia Corporation (formerly ExtraQuest Corporation) in Denver. StrataVia is the developer of the Data Palette autonomics technology and also offers managed database administration services. Venkat is the author of Oracle 24x7 Tips & Techniques *and co-author of* Oracle8*i* Web Development.

There's No Substitute for Experience

Our team represents some of the most knowledgeable and experienced in the industry. We are authors and speakers with long careers as Oracle experts, averaging 12 years. Our specialty is providing **remote DBA services** and **onsite Oracle database consulting**.

We offer a free consultation to discuss:

- ➤ Increasing uptime and reliability
- ➤ Minimizing downtime and data loss
- ➤ Optimizing performance
- ➤ Reducing cost of database operations

Call Us Today!

(415) 344-0500 • (888) 648-0500

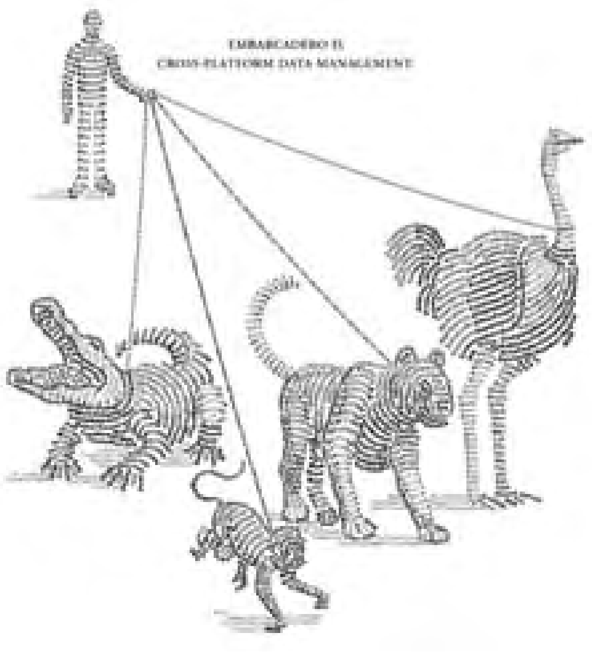
www.dbspecialists.com

ORACLE | CERTIFIED SOLUTION PARTNER

DatabaseSpecialists

● In a perfect world, all databases would be created equal. But in reality, it's a jungle out there. Different database platforms. Diverse operating systems. And increasing pressure to keep them all under control. Fortunately, there's an easy way to manage all your data. With Embarcadero's DBArtisan, Rapid SQL, and the Analyst Series, you'll handle more data with fewer resources. Reduce costs. And instantly extend your expertise across multiple database platforms. To learn more, please visit www.embarcadero.com/cross-platform

EMBARCADERO TECHNOLOGIES
we make data work



Bag 'o Tricks

by Danny Chow

Danny Chow

You may find that a query that fails with an ORA-01652 error does not itself use a lot of temp space. There could be other active statements using a lot of temp space and elbowing out queries that need just a little temp space. It is not difficult to determine how the database's temp space is being used; a few queries against the v$ views will give you all the information you need to troubleshoot the problem on your system.

- The v$sort_segment view describes general information about each sort segment, such as size and number of blocks currently in use. (This view only provides information about temporary segments located in temporary tablespaces.)

- The v$sort_usage view describes how the space within one temporary segment is being used on a session and statement level. Typically, an instance will have only one sort segment per temporary tablespace, but the space within the segment can be used by multiple statements running in different sessions. Note that v$tempseg_usage is a synonym for v$sort_usage.

The following queries can be used to determine temp space usage on the database.

```
--
-- Listing of temp segments.
--

SELECT  A.tablespace_name tablespace,
        D.mb_total,
        SUM (A.used_blocks * D.block_size) / 1024 / 1024 mb_used,
        D.mb_total - SUM (A.used_blocks * D.block_size) / 1024 / 1024
mb_free
FROM    v$sort_segment A,
        (
        SELECT  B.name,
                C.block_size,
                SUM (C.bytes) / 1024 / 1024 mb_total
        FROM    v$tablespace B,
                v$tempfile C
        WHERE   B.ts#= C.ts#
        GROUP BY
                B.name,
                C.block_size
        ) D
WHERE   A.tablespace_name = D.name
GROUP BY
        A.tablespace_name,
        D.mb_total;
```

```
--
-- Temp segment usage per session.
--

SELECT  S.sid || ',' || S.serial# sid_serial,
        S.username,
        S.osuser,
        P.spid,
        S.module,
        P.program,
        SUM (T.blocks) * TBS.block_size / 1024 / 1024 mb_used,
        T.tablespace,
        COUNT(*) statements
FROM    v$sort_usage T,
        v$session S,
        dba_tablespaces TBS,
        v$process P
WHERE   T.session_addr = S.saddr
AND     S.paddr = P.addr
AND     T.tablespace = TBS.tablespace_name
GROUP BY
        S.sid,
        S.serial#,
        S.username,
        S.osuser,
        P.spid,
        S.module,
        P.program,
        TBS.block_size,
        T.tablespace
ORDER BY
        sid_serial;
```

```
--
-- Temp segment usage per statement.
--

SELECT  S.sid || ',' || S.serial# sid_serial,
        S.username,
        Q.hash_value,
        Q.sql_text,
        T.blocks * TBS.block_size / 1024 / 1024 mb_used,
        T.tablespace
FROM    v$sort_usage T,
        v$session S,
        v$sqlarea Q,
        dba_tablespaces TBS
WHERE   T.session_addr = S.saddr
AND     T.sqladdr = Q.address
AND     T.tablespace = TBS.tablespace_name
ORDER BY
        S.sid;
```

Note that a minor adaptation will be required for Oracle 8*i* databases because the dba_tablespaces view does not have a block_size column in Oracle 8*i*. ▲

Danny Chow is a senior DBA with Database Specialists. He can be contacted via **dchow@dbspecialists.com**.

Fall Conference Abstracts

	Room 308	Room 304	Room 301B
11:00 a.m. to 12:00	**Questioning Method R—Cary Millsap, Hotsos Enterprises** In 2003, Cary Millsap and Jeff Holt prescribed a new Oracle performance problem diagnosis method called Method R in the book *Optimizing Oracle Performance*. Supporters of Method R cite extraordinary successes, and its critics describe various inadequacies. This session presents a refined understanding of Method R that incorporates experience with the method since its invention.	**RAC Performance Tuning Best Practices—Sri Subramaniam, Oracle Corporation** The session will cover RAC architecture, RAC tuning, RAC-specific wait events, Automatic Workload Repository (AWR), and Automatic Database Diagnostic Monitor (ADDM).	
1:00 p.m. to 2:00 p.m.	**The New Data Pump—Caleb Small, CALEB.COM Technology Consulting** Data Pump is the next generation of Oracle's Import/Export utility. Its architecture is radically different, offering many new benefits, as well as some serious "gotchas" for the uninitiated. This session explores the architecture and use of Data Pump and includes numerous live demos of the new tool.	**Web Application Security with JAZN—Implementing the Superstition in JDeveloper—Peter Koletzke, Quovera** Oracle Containers for J2EE (OC4J) offers runtime a security feature, JAZN (Java AuthoriZatioN), which allows developers to use standard Java security libraries to implement user access and restriction features. This presentation explains this feature and discusses how it provides solid security services for J2EE web applications.	**Developing High-Class UML Class Models—Jeffrey Jacobs, Covad Communications** Class models form the heart and soul of UML modeling, just as ER models form the core of Oracle method and information engineering. Poorly developed models result in brittle, low-quality systems. This presentation will present techniques and guidelines for avoiding common mistakes in class models, resulting in high-quality, robust models and systems.
2:30 p.m. to 3:30 p.m.	**Planning and Installing a RAC Database—Caleb Small, CALEB.COM Technology Consulting** Real Application Cluster technology is gaining acceptance in the industry. It brings many benefits in terms of scalability, reliability, and high availability. It also represents some significant challenges to successful installation and operation. This session explores the critical success factors required to plan and install a RAC database.	**Unraveling the Mysteries of Web Application Communications—Peter Koletzke, Quovera** This presentation explains the mechanics of HTTP. It describes the communication between the client browser and application server; HTTP request and response messages; Get and Post methods; the components of a URL; and J2EE deployment files such as web.xml, server.xml, Enterprise Archive (EAR), and Web Archive (WAR).	**Oracle Workload Characterization—Andy Rivenes, Lawrence Livermore National Laboratory** Workload characterization is the process of identifying classes of workload, measuring those classes and then identifying their impact to the business. The goal is to understand the impact of those classes on database workload and to enable the organization to better schedule its business processes. We will explore how this can be done in an Oracle environment.
4:00 p.m. to 5:00 p.m.	**Root Cause and Other Urban DBA Legends—Brian Hitchcock, Sun Microsystems** Over the last several years, I've worked on a number of interesting cases that involved poor performance, downed servers, upgrades gone bad, etc. We take up each case and examine what worked, what didn't, whether we ever found the root cause, and whether 10046 traces, wait states, and Statspack were useful.	**SQL Design Patterns—Vadim Tropashko, Oracle Corporation** A design pattern is a general, repeatable solution to a commonly occurring problem in software design. Each pattern has its own name, so that developers can instantly refer to it and leverage it as a standard solution. Beyond a certain point, the skill of piling up subqueries doesn't pay off, and one has to study some rudimentary theory that classifies known SQL solutions into common design patterns.	**JSP Web Pages with a Database Backend—Joel Thompson, Rhino Systems** This presentation walks you through how to create a basic Java/JSP web page that uses Oracle as a data source to fill a table. The discussion will focus primarily on web page basics like JSP Lifecycle, HTML/JavaScript, JSP Scriptlets, JSTL, and JDBC. We will also touch upon N-tier architectures, and frameworks like Struts, JavaServer Faces, and EJB.

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, Darrin Swan, at darrin.swan@quest.com. ▲

Long-term event sponsorship:

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

Thank you! Year 2006 Gold Vendors:

- BEZ Systems
- Confio Software
- Data Domain
- Database Specialists, Inc.
- Embarcadero Technologies
- GoldenGate Software, Inc.
- IT Convergence
- Quest Software
- Quovera, Inc.

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

TREASURER'S REPORT

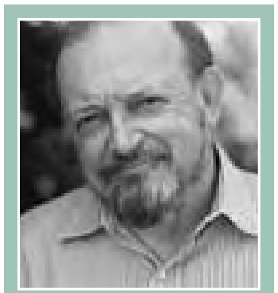
Diane Lee, *Treasurer*

Beginning Balance		
July 1, 2006		**$ 43,280.20**
Revenue		
Membership Dues	2,180.00	
Meeting Fees	1,080.00	
Vendor Receipts	7,125.00	
Miscellaneous	27.19	
Interest	57.94	
Total Revenue		**$ 10,470.13**
Expenses		
Regional Meeting	5,520.43	
Journal	7,211.90	
Membership	29.66	
Administration	28.20	
Website	–	
Board Meeting	616.27	
Marketing	–	
Vendors	29.60	
Total Expenses		**$ 13,436.06**
Ending Balance		
September 30, 2006		**$ 40,314.27**

Career Health Check-up!

by Clay Parsons

The I.T. community is very concerned about the offshoring of I.T. jobs. Veteran career management coach Clay Parsons has some advice for us.


Clay Parsons

Be Proactive

Are you concerned about the offshoring of I.T. jobs? I'm here to tell you that it's not so much about quickly finding another job when you lose one, but about proactive career management and development. Take this quiz to find out if you're managing your career in the best possible way. Give yourself a score of 1 if your answer is an unequivocal "Absolutely," 2 if your answer is "I've made some progress in this regard," 3 if your answer is "I've been trying to find the time to do this," and 4 if your answer is a plain "No."

1. I know my personal career assets including skills, abilities, accomplishments, and personal resources.
2. I know my career/life goals and objectives. I try to think about them regularly and keep them updated.
3. I know how to prospect for job leads where none are advertised.
4. I can identify at least five potential employers with whom I want to obtain interviews in the future.
5. I have taken career interests and personality assessments in the past three years and had them interpreted by a career development professional.
6. I am actively expanding my network. I am a member of all of the professional or trade associations related to my job/career. I also do volunteer work.
7. I subscribe to all of my profession's publications. I watch for articles about new trends. I also monitor job advertisements and announcements.
8. I am keeping my skills up to date and maintaining a "knowledge edge" over my competition.
9. I enjoy going to work in the morning. I like my job.
10. I meet regularly with a career development professional to discuss my job and career path.
11. I have adequate transportation, child care, and family support.
12. I read the business and professional section of papers such as the *San Jose Mercury News* and the *Wall Street Journal*.
13. I have accessed the Internet for my career planning and job search.
14. I have identified the personal barriers/issues which could sabotage my future career, and I am actively working on them with a counseling professional.
15. I have conducted a mock interview with at least two people during the past year.
16. I know where to receive the proper education and training to help me advance and obtain employment. I also have a long-term education plan and I am currently implementing it by taking classes.
17. I have a detailed "career search" journal, in which I keep a record of all of my job or career search- and advancement-related activities.
18. I have several carefully prepared responses to the following questions: "What are you interested in? What do you want to do? What could you do for us?" In fact, I have used them on several recent occasions.
19. I have a personal journal in which I record my thoughts, feelings, and ideas about my job and career.
20. I have prepared a list of those transferable skills I can use in my next job.

How Did You Score?

1 to 20—Great job! Keep right on truckin'!
21 to 40—A little tune-up wouldn't hurt!
41 to 60—It's time for an overhaul!
61 to 80—Quick! You need a complete revitalization. Do it now! ▲

Clayton Reed Parsons, MA, MLA, is principal, career counselor, and coach at ALTERNATIVE FUTURES, a full-service Career Development Consulting Services firm in Berkeley, California. Clay is an expert on helping people change and manage their careers and find meaningful work. He can be reached at (510) 287-5664, **clay@alternativefutures.com,** *or through his award-winning website,* **www.alternativefutures.com**.

AMOREACTIONBETTERACCESSM

RE **LARRY ELLISON** RESOURCEST

DEXPERTS **CHARLES PHILLIPS** M

DEMOS **HECTOR DE J. RUIZ** ENE

EOPENWORLD **MICHAEL DELL** M

TIGROWTH **MARK HURD** PROFIT

IFUN **JOHN CHAMBERS** SALES

ORE **JONATHAN SCHWARTZ** MO

NMOREVALUEBETTERRESULTSM

ORACLE OPEN WORLD

Join leading industry innovators at Oracle OpenWorld for their take on the latest trends shaping the world of business and technology. Get more inspiration and more insight for better results.

OCT 22–26, 2006 | MOSCONE CENTER | SAN FRANCISCO
Register today: **oracle.com/openworld/sanfrancisco**

ORACLE®

[illegible] DIAMOND PREMIER ELITE

AMD Cisco Dell hp Sun NetApp IBM

[illegible] PLATINUM INNOVATOR

Capgemini Deloitte. EDS EMC² accenture

NoCOUG Fall Conference Schedule

November 2, 2006, at PG&E, San Francisco, CA

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

8:00A.M.–9:00	**Registration and Continental Breakfast**—Refreshments served
9:00–9:30	**General Session and Welcome**—Darrin Swan, NoCOUG President
9:30–10:30	**Keynote:** *Why You Can't See Your Real Performance Problems*—Cary Millsap, Hotsos Enterprises
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Room 308: *Questioning Method R*—Cary Millsap, Hotsos Enterprises
	Room 304: *RAC Performance Tuning Best Practices*—Sri Subramaniam, Oracle Corporation
12:00–1:00 P.M.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Room 308: *Planning and Installing a RAC Database*—Caleb Small, CALEB.COM Technology Consulting
	Room 304: *Web Application Security with JAZN—Implementing the Superstition in JDeveloper* —Peter Koletzke, Quovera
	Room 301B: *Developing High-Class UMLO Class Models*—Jeffrey Jacobs, Covad Comminications
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Room 308: *Planning and Installing a RAC Database*—Caleb Small, CALEB.COM Technology Consulting
	Room 304: *Unraveling the Mysteries of Web Application Communications*—Peter Koletzke, Quovera
	Room 301B: *Oracle Workload Characterization*—Andy Rivenes, Lawrence Livermore National Laboratory
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Room 308: *Root Cause and Other Urban DBA Legends* —Brian Hitchcock, Sun Microsystems
	Room 304: *SQL Design Patterns*—Vadim Tropashko, Oracle Corporation
	Room 301B: *JSP Web Pages with a Database Backend*—Joel Thompson, Rhino Systems
5:00–??	**NoCOUG Networking and Happy Hour at Beale Street Bar and Grill, 133 Beale Street at Mission**

Session descriptions appear on page 24.

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

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED

FIRST-CLASS MAIL
U.S. POSTAGE
PAID
SAN FRANCISCO, CA
PERMIT NO. 11882