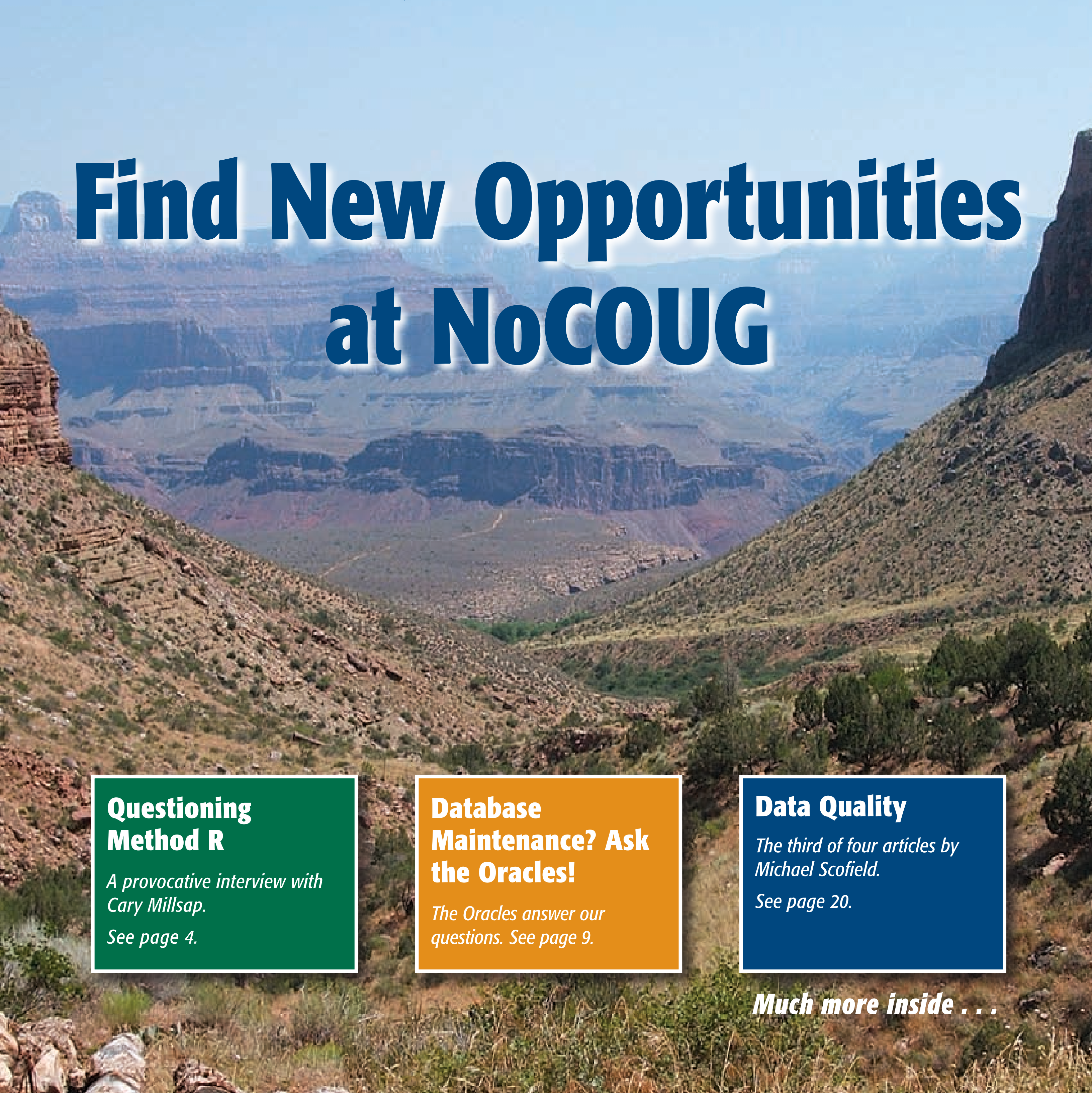
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 22, No. 3 · AUGUST 2008 $15



Find New Opportunities at NoCOUG

Questioning Method R

A provocative interview with Cary Millsap.

See page 4.

Database Maintenance? Ask the Oracles!

The Oracles answer our questions. See page 9.

Data Quality

The third of four articles by Michael Scofield.

See page 20.

Much more inside . . .

Great Training Value for Only $80

When I interviewed Jeremiah Wilton for the NoCOUG Journal, I asked him about the value of NoCOUG membership, and this is how he answered:

"Local user group membership and participation is one of the ways that junior DBAs can become senior. Compared to high-priced national conferences, local user groups like NoCOUG provide DBAs with affordable access not only to their peers in the local community, but also to well-respected authors and speakers from the global Oracle community. Local user groups are also a great way to keep up with which companies are hiring, how other people are solving problems, and what technologies are emerging. It is hard to remain good in a field when you are in a bubble. Local user group membership and attendance gets you out of your company bubble and exposes you to new ideas and trends."

This year, NoCOUG is bringing you speakers of the caliber of Rich Niemiec, Cary Millsap, and Jonathan Lewis. Where in the world can you find such great training value for only $80? I do hope to see you at the summer conference on August 21. ▲

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

Publication Notices and Submission Format

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

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

The submission deadline for the upcoming November 2008 issue is August 31, 2008. Article submissions should be made in Microsoft Word format via email.

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

NoCOUG BOARD

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

Be Prepared!

by Roger Schrag

I first hiked into the Grand Canyon when I was 19 years old. I was on a road trip with three high school friends, and like many younger adults we were winging it. We set forth from Grand Canyon Village at the South Rim into the deepest gorge in the United States with every intention of hiking to the Colorado River at the bottom and back out all in one day.

Had we done any research beforehand, we would have discovered that the round-trip hike is about 20 miles long, involves nearly a mile of altitude change, and is strongly discouraged by the National Park Service. (The park rangers perform over 200 emergency rescues at the Grand Canyon each year, mostly because people attempt a hike they are not prepared for.)

We had no idea what was in store for us. We started too late in the day and we did not bring enough food or water. We hadn't arranged a place to stay overnight at the bottom, so we would have to make it back to the top of the canyon before dark. At a rest stop halfway down, we realized this was impossible. So we abandoned our goal and instead trudged back up to the South Rim.

The best of ideas—foiled by a lack of preparation. We hadn't a clue what we were doing. Some would say we were young and adventurous, or perhaps foolish or naive. But all the enthusiasm and good intentions the four of us had just couldn't make up for the fact that we hadn't adequately prepared for our hike into the Grand Canyon.

I came away from that trip wanting to return someday and try again. To celebrate my 40th birthday, my wife Lisa and I did just that. This time we made it to the bottom. The hike was easy and a great deal of fun. Friends and family think we are in such great shape because we've hiked to the bottom of the Grand Canyon. But actually it can be easier than you might think—if you do the proper planning and preparation.

This time, we booked a cabin at the bottom of the canyon two years in advance. This meant we didn't have to hike down and up all in one day, and we didn't have to carry camping equipment either. We did lots of research so we'd know what the trails would be like. We also bought the proper equipment—good boots, hiking sticks, and such. And of course we took many training hikes to get in shape.

Have you figured out yet where I am going with all of this? Are you still wondering what any of this has to do with being a good Oracle professional? I'll sum it up in one sentence: Planning a successful Oracle project is not much different from planning a successful hike to the bottom of the Grand Canyon.

When I was an Oracle DBA and I was scheduled to perform some sort of database project—such as upgrading a database or rolling out a bunch of schema changes—I planned and prepared just like I did with the second Grand Canyon trip.

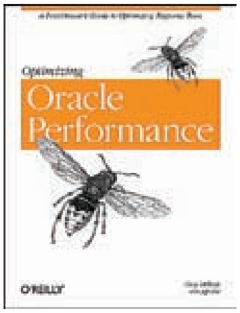
Instead of booking accommodations, I would coordinate schedules with all parties affected by the database work. Instead of researching the water supplies on the trail I'd research all of the established facts and best practices related to my database project. Instead of buying good hiking boots I'd make sure I had all of the right tools for the job and knew how to use them properly. And instead of training hikes I'd practice the database maneuver repeatedly in a test environment to make sure it went smoothly in production.

Adequately preparing for your Oracle project greatly increases the chance of success and reflects positively on you at the same time. Preparing and planning will help you move your Oracle career forward, and NoCOUG can help in this area as well. Use NoCOUG as a resource in your research and information-gathering stage. At NoCOUG conferences you can network with others who may have already completed a similar task. The NoCOUG Journal and website are also valuable sources of information.

This takes us up to the NoCOUG Summer Conference taking place August 21 in San Ramon. Cary Millsap, a recognized leader in the realm of Oracle performance management and methodology, will kick off the day with a keynote presentation in which he will discuss his Grand Unified Theory of Tuning, followed by a presentation of performance problem diagnosis case studies.

The day will be filled with technical presentations by staff from Oracle Corporation as well as real Oracle users like you and me. The day will be rounded out with the usual networking opportunities, book raffles, and plenty of food and drink. Get all of the conference details and submit your registration on the NoCOUG website at **www.nocoug.org**.

See you there! ▲



Nullius in Verba—Questioning Method R

Cary Millsap

Cary Millsap will be delivering the keynote address at our Summer Conference on August 21. He is the co-creator of Method R, an antidote for Compulsive Tuning Disorder—the disease that afflicts so many of us. His motto is "Nullius in verba" which means "On the words of no one." In other words, check the facts for yourself.

Questioning Method R

What is Method R and how does it cure Compulsive Tuning Disorder?

Method R is a commonsense approach to optimizing a system. The "R" stands for "response time." Many people have used Method R extensively at Oracle sites throughout the 2000s with tremendous success. Jeff Holt and I first documented the method in the book *Optimizing Oracle Performance* (O'Reilly 2003). You can read Chapter 1 online for free at **www.oreilly.com/catalog/optoraclep/chapter/ch01.pdf**.

Method R prescribes four simple steps:

1. Identify the task that's the most important to you.
2. Measure its *response time* (R). In detail.
3. Optimize that response time in the most economically efficient way.
4. Repeat until your system is economically optimal.

Compulsive Tuning Disorder, or *CTD*, is a term introduced by Gaja Vaidyanatha, Kirti Deshpande, and John Kostelac in their 2001 book, *Oracle Performance Tuning 101*. CTD is the compulsion to tinker forever with the performance of a system, inspired by the hope that there *might* be something in there that needs improving. It's a disorder instead of a virtue because CTD causes people to waste time and money.

Jeff and I developed Method R for Oracle in response to several big problems with traditional "tuning" approaches, the CTD-inspiring absence of a terminating condition among them.

CTD just doesn't make sense in a Method R context. With Method R, you know exactly whether or not a computer-executed business task has any time in it that's been wasted. If there is, you either fix it or prove that fixing it isn't worth the cost. If there isn't any wasted time in there, you move on. That's the key: either way you move on. You don't linger in a non-productive state, because you know exactly where there's opportunity for improvement and you know exactly where there's not.

We love stories. Tell us a performance tuning story. Tell us two.

Recently, Karen Morton has posted some good performance stories at her blog. Karen is my director of education and consulting at Method R Corporation. Two stories that you should get a kick out of can be found at:

karenmorton.blogspot.com/2008/05/dual-ing-for-resources.html

karenmorton.blogspot.com/2008/06/dont-do-work-you-dont-have-to-do.html

How about three? My favorite performance tuning story at the moment is a note I received yesterday from a customer who has recently bought our Method R Profiler product. He and his team had previously been using an open-source profiling tool, so I had expressed interest in how he was feeling so far about the purchase. Here is the response I received:

"Within one minute of installation the Profiler exceeded all expectations. We went live with a new warehouse management system this Friday because all of the tuning was done ahead of time. The Profiler pointed out all of the application inefficiencies which we turned over to the vendor. They were able to fix the ones with the largest impact on the online response times. That gave our management the confidence they needed to make the Go/No-Go decision. Until I got the Profiler we were looking at postponing it until next quarter. I pity the poor souls that have to optimize in mere mortal ways."

I love this story because it touches on so many of the reasons we created Method R in the first place, like:

- It hooks right in to the *business* in a way that makes performance easy for everyone in the company to understand.
- It gives you positive evidence of success—not just some uncertain feeling that maybe you're okay because you haven't yet found any problems. This is why the CTD problem never enters into a Method R project.

- It shows you unambiguously what's really taking your time, so you don't have to guess what effect your tuning will have on the business task you're trying to help. *Nullius in verba*!
- It gives you more power to help your application software vendors help you. It makes you faster and more effective, even when you buy your applications from somebody else.

Which is better, Method R or ASH/AWR?

That's like asking, "Which is better, Google Maps or fuel injection?" Method R is a process to accomplish a specific goal. ASH/AWR is a functional capability within the Oracle Database software that allows someone to partly implement Method R.[1]

A major confusion I've encountered in the field is the misunderstanding in which people equate Method R with Oracle trace data. The method called Method R has nothing specifically to do with Oracle or databases—or even computer software. It's simply a method for optimizing a process—any process, like driving to work or shopping for groceries.

*It does so happen that implementing Method R within the Oracle Database context requires Oracle extended SQL trace data in some (I'd say many) cases. There are two principal reasons for this. First, ASH/AWR is wired to the concept of database time. That's different from end-user response time. In some cases, it's very different. In those cases, we need to see the amount of "SQL*Net message from client" time that a process consumes, and even the time that's completely unaccounted for by database measurement probes.*

Second, there are issues of data skew that you can detect in detailed trace data that are impossible to detect with ASH/AWR. For example, 100 calls consumed 100 seconds. How much time will we save if we can eliminate 90 calls? The answer is not necessarily 90 seconds; it could be anything between nearly 0 seconds and nearly 100 seconds. It depends completely upon which specific calls you eliminate. (Hint: Imagine that the 100 call durations were as follows: 99.01s, .01s, .01s, .01s, …, .01s.)

ASH and AWR are good things. But an argument for ASH/AWR is not an argument against Method R. The closer Oracle Corporation comes to giving us the true end-user response-time measurements that Method R requires, the better for everyone.

We spend a lot of time tuning databases. Why does it have to be that way? Is Oracle difficult to tune? Are we building the databases incorrectly?

I began learning how to tune Oracle databases in 1989. At the time, it seemed nearly impossible to me that people could be productive by following these instructions that the "experts" at the time were telling me I had to follow. I would have felt better had I known that it wasn't just my own ineptitude working against me. In recent years, I've been able to prove why the instructions we were taught were fundamentally ineffective against a lot of the everyday performance problems we all have to face.

So is Oracle difficult to tune? If you do it the way we've all been taught, then absolutely: it's nearly impossible. In fact, the better you are as a student, the harder it gets. It's because the method is flawed. Is it difficult to peel an apple? If the only tool you have is a steel ball bearing, then absolutely: it's nearly impossible.

I think a lot of databases are built incorrectly because people don't measure them properly as they're building them. It's a result of the same method-is-flawed problem I just mentioned. The things that database architects, application developers, operational administrators—everyone in the stack—have been taught to measure regarding performance are just ludicrous. The measurements you're taught to use—the counters and the ratios—are not connected to the qualities that real people using the real application in real life will really measure later.

It's difficult to do anything new—especially when it's complicated—without immediate feedback. Imagine trying to learn how to putt in golf in an environment where you couldn't learn where your ball went until three months after you hit it.

That's what people do with their databases. Everyone in the project has lots and lots of tasks to do, but often it's nowhere in the plan to measure *end-user response time* during the construction process. Then, when the system goes live three months later, the only performance metric that most people are going to care about hasn't even been looked at. Worse yet, many systems (*most* systems) don't include instrumentation that allows an analyst in a real-life production environment to even *measure* what the real response time is as it's happening.

There's no way you can expect to succeed in an environment like that.

Won't upgrading the hardware save time and money in the long run?

It depends. Before you upgrade hardware, you really need to consider Amdahl's Law:

> *Your upgrade will improve performance of only those tasks that use a lot of the component you improved.*

A lot of people upgrade CPUs to 2× faster devices, or maybe, as I recently wrote at **carymillsap.blogspot.com**, they upgrade their disk drives to 100× faster flash drives and then expect their tasks to be 2× or 100× faster, and they're just not. It's Amdahl's Law: When you make your device *n* times faster, only the component of response time where that device was actually *used* will be *n* times faster (and that's the optimistic case). So if your total response time is only 5% disk I/O to begin with, upgrading to flash drives will improve performance by *no more than 5%*.

Hardware upgrades don't always end well. In *Optimizing Oracle Performance*, Jeff and I describe a now-classic case in which a hardware upgrade actually slowed down the perfor-

[1] Editor's Note: AWR requires licenses for Enterprise Edition and Diagnostics Pack.

mance of an important business task. It's like a UFO sighting. A lot of people have claimed that they've experienced it, but having the actual data to prove what really happened . . . that's another thing entirely. In our book, we have all the data it takes to show exactly how this kind of catastrophe happens. You can read the whole story at **www.oreillynet.com/pub/a/network/excerpt/optimizing_oracle_chap12/index.html**.

Here's another story for you. A couple of years ago, I was teaching a private course at a company. In the part of the course where I was supposed to have convinced everyone that the Method R way of doing things was worth learning about, I still had a lot of students with crossed arms and body language that was generally saying, "I ain't buying it."

So I suggested that we should work on an actual problem that the company had recently experienced. That way, I could show off the method in a context that was already familiar to them. They agreed.

They told me that they had recently executed a half-million-dollar hardware upgrade that had doubled a system's number of CPUs from 20 to 40. But the system throughput was just as lousy on the 40-CPU box as it had been on the 20-CPU box. They wanted to know what could possibly cause this application not to scale up at all in spite of the much greater capacity.

They had actually *printed* the 7MB trace file for an important task that had taken about 100 seconds to run. The document was about two inches think. They motioned in frustration at this stack of paper and explained that they had already run *tkprof* on the file, and that hadn't helped their understanding at all.

The task behind the trace file was a batch job that comprised nearly the total workload on this upgraded machine. They had a 40-CPU machine with 300 concurrent copies of this batch job that they had traced running on it.

I ran our Profiler product on the trace data. The total run time for the Profiler itself was about 7 seconds, which was good given that all eyes were on me while it was running. The output was most excellent.

The #1 response time contributor: unaccounted-for time, almost 60%. I expected that before I even looked at the profile. That's what you'll get when you run 300 concurrent batch jobs on a 40-CPU machine. (I wrote about this phenomenon many years ago in a paper called "Batch queue management and the magic of '2'.") CPU service for database calls was #2, at about 13%. I expected that too. Sleeps for "library cache" latch acquisitions was #3, at about 10%. There was more; this was enough.

So I looked at the task's top time-consuming SQL, and it looked pretty good to me. Only five LIO calls per row manipulated per joined table, that kind of thing.

Their performance problem wasn't bad SQL or a lack of horsepower, it was an inevitability of how the application was designed. Every one of these 300 identical processes was parsing like crazy inside of Pro*C loops. It looked like tens of thousands of prepare calls from these 300 processes were occurring every minute, because I saw hundreds of them per few-second interval in each of the 300 concurrent jobs.

The prepare calls were all blocking all over each other because they were all trying to prepare the same exact SQL query, all in different sessions, all at the same time. Of course, the one hash chain in the library cache was getting scanned over and over again, and those scans are protected by the chain's "library cache" latch. On top of all the latch waiting was all the CPU service time consumed by the spinning for latches. And on top of that was the waiting in a CPU-preempted state for CPU service that the majority of the 300 concurrent jobs were spending the majority of their time doing.

This is an example of how a hardware upgrade *cannot* improve performance of an application whose bottleneck is software serialization. The remedy: stop making so many OCIStmtPrepare calls from the application. At least keep cursors open for reuse within a single session. Better yet, use a three-tier transaction monitor architecture that will allow several processes to share a single cursor. This application should have parsed each of its SQL statements once per instance lifetime, not thousands of times per session lifetime.

What's next for Method R?

Well, Method R is not just a method anymore, it's a whole new company now. I formed Method R Corporation earlier this year, and I have some fantastic people with me. We can help you if you have needs pertaining in any way to performance of Oracle-based applications. Visit **www.prweb.com/releases/2008/05/prweb839554.htm** for details.

A major opportunity for us right now is that Method R has enormous untapped potential in preventive contexts. Today

I think that people view Method R dominantly as a performance problem diagnostic method. That's fair, because that's what the book is about, and I think that's how most people who know about Method R have come to know about it.

But the method works for preventing performance problems from ever occurring in the first place. Application developers drastically reduce their performance problem vulnerability when they regularly profile their code at each stage of promotion, from development through unit testing, performance readiness testing, and into production. Profiles—expressed plainly in seconds of response time—should be the standard language that architects, developers, database and system administrators, end users, and managers use to discuss application performance.

As the founder of a new business, I understand the natural desire that a business leader has to outsource any service that's not part of his core business. It's why my company outsources our HR and payroll functions to a PEO. It's why we buy Gmail services from Google and accounting software services from QuickBooks Online. Oracle application performance is the same way. Why on earth would an airline or an entertainment company or a bank or a telco want to learn everything that Method R Corporation knows about Oracle performance?

What I think people want is the luxury of forgetting about software performance. At Method R Corporation, we can provide that luxury. We can track the performance of your applications, the way your users perceive them, through a service that you don't have to think about all the time. We can let you know in advance when your performance is trending toward danger. We have written the software that makes it really easy for us to do that. Of course, when we help you find (or predict) a problem, we can help you fix it, or if you prefer, we can teach you how to fix it yourself. The goal is the same in either case: genuinely satisfying software performance for the users of your system. That's what's next for Method R.

Ask the Oracle

Cars need regular maintenance. Ships need regular maintenance. Planes need regular maintenance. Don't databases need regular maintenance?[2]

They do, but the analogy is a little more twisty than you might think. Physical machines have parts that wear out. Software doesn't wear out, but from time to time it can need a human hand to help it adapt to the changes that periodically intrude upon it.

Your database changes all the time. Every insert, update, and delete changes the world in which your database server must function. With Oracle software, you do need to perform some types of regular maintenance in response to those changes.

One type of regular maintenance that databases need is software patching. No software that is as complex as a database manager is perfect; there are defects inside. As your software manufacturer finds and fixes those defects, you're going to benefit from some of the patches that result.

Another type of regular maintenance that some databases need is gathering of statistics that the optimizer uses to make its decisions about query plan assignments. That gathering process is not yet completely foolproof, and so it requires some human decision-making and periodic attention.

One topic of regular maintenance that I should mention specifically because there is so much discussion about it out there is index rebuilding. That's an area where a lot of people think their database needs regular maintenance, but it really doesn't. Richard Foote, Jonathan Lewis, and Tom Kyte have some excellent articles explaining why.

Over time, as Oracle adds more features to the database software, the requirement for more and more kinds of regular maintenance will diminish, and probably eventually disappear. An easy example from history is Oracle's approach to undo storage. In version 6, I spent a *lot* of my time as a DBA rebuilding people's rollback segments. That kind of thing is no longer necessary, because with rollback segments, Oracle Corporation has evolved its software to eliminate the need for so much human intervention.

Help! My database is slow![3]

"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.

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

[2] Originally printed in the *Ask the Oracles* column in the August 2008 issue of the *NoCOUG Journal*.

[3] Originally printed in the *Ask the Oracles* column in the November 2006 issue of the *NoCOUG Journal*.

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.

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

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.

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 sys-tem. It was one of those, "Ah! Don't do it that way—do it this way" problems. Of course, more often, there's some-thing legitimately wrong with how the system responds to Nancy's click, so you'll need more data to figure out what it is.

The data that you need is profile data: the kind of stuff C programmers get when they use "gcc -pg," and 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 Profiler product), which filters and aggregates profile data in useful ways to help you make quick sense of what took so long.

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.

Does the optimizer need a hint?[4]

I like Tom Kyte's idea that there are good Hints and there are bad Hints (**asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:7038986332061**). Good Hints, like FIRST_ROWS and ALL_ROWS, give the Optimizer additional information about your intentions that can help it to make better decisions. Bad Hints, like USE_NL and HASH, constrain the Optimizer from choosing an execution plan that might actually be ideal for you.

Bad Hints are like morphine for your database. They're just right for a very specific purpose, but if your application needs them to run, it's a problem.

Bad Hints are just right for experimenting with your SQL to see how different plans perform. Hints let you tell the Optimizer exactly what to do, so you can measure how it acts when it does it.

But habitual bad-Hint use is a bad thing. Especially because since version 9.2, the Oracle Optimizer generally makes excellent decisions based upon the information you (or your DBA) give it. Having the Optimizer is a lot like having a really smart SQL performance expert in your office.

And here's where I think a lot of people mess up. Imagine that a really smart SQL performance expert is saying that your query's most efficient execution plan is something you find utterly ridiculous. What would you do? If it really were a smart person in your office, you might at least listen respectfully. But with Oracle, a lot of people just roll their eyes and slam a Hint onto their SQL to constrain the Optimizer from choosing the apparently dumb plan.

The problem is that the Optimizer probably made a smart decision based on the data you gave it. Maybe it chose a stupid plan because you accidentally told it that your 10,000,000-row table has only 20 rows in it. If you just tape your Optimizer's mouth shut with an INDEX Hint, you may never find the 26 other queries that also use bad plans because of this bad assumption.

So when your Optimizer does something crazy, don't reach for the Hints; find out why a good decision-maker has made a bad decision. The less you can rely on bad Hints, the less time and effort you'll have to spend hand-tuning individual SQL statements, and the better your odds will be of having stable performance after your next database upgrade.

You can cure your addiction to bad Hints. Start by visiting the Asktom URL that I listed earlier. The full prescription is to read Jonathan Lewis's outstanding book *Cost-Based Oracle—Fundamentals*. It covers a tremendous range of information that will help make the Oracle Optimizer your friend. ▲

*Cary Millsap is the founder and president of Method R Corporation (***www.method-r.com***), a company devoted to genuinely satisfying software performance. He is widely known in the Oracle community as a speaker, educator, consultant, and writer. He is the author (with Jeff Holt) of* Optimizing Oracle Performance, *for which he and Jeff were named* Oracle Magazine*'s 2004 Authors of the Year. He is also a contributor to* Oracle Insights: Tales of the Oak Table. *Cary is the former vice president of Oracle's System Performance Group, and a co-founder of Hotsos. Cary is also an Oracle ACE Director and a founding partner of the Oak Table Network, an informal association of Oracle scientists that are well known throughout the Oracle community.*

[4] Originally printed in the *Ask the Oracles* column in the August 2006 issue of the *NoCOUG Journal.*

Database Maintenance?

Ask the Oracles!

Now there were some terrible seeds on the planet that was the home of the little prince; and these were the seeds of the baobab. The soil of that planet was infested with them. A baobab is something you will never, never be able to get rid of if you attend to it too late. It spreads over the entire planet. It bores clear through it with its roots. And if the planet is too small, and the baobabs are too many, they split it in pieces . . .

"It is a question of discipline," the little prince said to me later on. "When you've finished your own toilet in the morning, then it is time to attend to the toilet of your planet, just so, with the greatest care. You must see to it that you pull up regularly all the baobabs, at the very first moment when they can be distinguished from the rosebushes which they resemble so closely in their earliest youth. It is very tedious work," the little prince added, "but very easy."

And one day he said to me: "You ought to make a beautiful drawing, so that the children where you live can see exactly how all this is. That would be very useful to them if they were to travel some day. Sometimes," he added, "there is no harm in putting off a piece of work until another day. But when it is a matter of baobabs, that always means a catastrophe. I knew a planet that was inhabited by a lazy man. He neglected three little bushes . . ."

So, as the little prince described it to me, I have made a drawing of that planet. I do not much like to take the tone of a moralist. But the danger of the baobabs is so little understood, and such considerable risks would be run by anyone who might get lost on an asteroid, that for once I am breaking through my reserve. "Children," I say plainly, "watch out for the baobabs!"

—from *The Little Prince* by Saint-Exupéry

Arup Nanda: Like anything else, maintenance in databases has to have a purpose. Before deciding on any activity, however innocuous it may seem, ask yourself these questions: Why are you doing it?, What harm will be done if it's not done?, What are the potential side effects?, and Is there a fallback plan? It may sound like a rocket launch, but often this simple discipline of addressing these questions beforehand saves you many hours of frustration and helps you understand other factors that you might have overlooked. Beware, especially, of the so-called "best practices," many of which can be so far from best. So, there is no one-size-fits-all policy for maintenance.

Consider, for instance, statistics collection. Best practices advocated by many pundits and even from Oracle repeat the theme: your regular maintenance should include stats gathering, but time and again this has been proved wrong. Stats are about patterns in data, not data themselves. For instance it's immaterial that the company sold 2000 more widgets this month; if it's about the same percentage of the overall product sales compared to last month, then your stats collection does nothing to help the optimizer and may have hurt some other query plans. So, you spent time and resources to perform a so-called maintenance activity only to introduce issues elsewhere and with no benefit to your original cause. The same goes for index rebuilding. Research by many has proven that in many cases rebuilding index may do nothing to improve the performance, and in some cases it may hurt since the clustering factor might change.

What should you do as a "regular" maintenance? First, whatever the activity may be, it should not need immediate attention and can wait. It should have no side effects, should have a rationale (not just best practices), and should be relatively simple. I consider these activities alone:

1. Cleaning up trace files, log files, and core dumps a few days old. Why? Because if you don't, then Oracle Home might get filled up, forcing the database to halt. Oh yes, don't forget the audit files under rdbms/audit, and other log files such as RMAN logs and the sqlnet.log files all over the place. A simple script such as the following will do the trick:

```
find /u02/app/oracle/admin -name “*.log” -ctime ${DAYS}
-exec rm {} \;
```

2. Moving the alert log to a new name and compressing it.
3. Trimming the listener log by using dd from /dev/null. Remember, you shouldn't remove the listener.log; unlike alert.log, it just continues being written unless listener is shut down.
4. Trim the AUD$ table (if you have enabled auditing). Remember, you can't just truncate the table. Take advantage of a downtime to perform this.
5. Test if backups are valid. A simple RMAN command "restore database preview" will check the existence of the proper backupsets without actually restoring.
6. Checking for space consumption and updating projections for future growth. The same goes for bandwidth of all other resources such as CPU, I/O, network, and so on.
7. Checking for objects with a lot of wasted space and a very large high water mark, and shrinking them.

Some things should always be scripted and never part of a time-driven maintenance. Examples include adding range partitions (dropping may be maintenance), checking for ORA-600/7445 errors in alert log, tablespaces approaching the limit, and so on. Checking corruption in archivelogs is done in RMAN backup. Other maintenance activities that you may not have control over include applying quarterly security patches. Applying any other patches is mostly an event drive activity; you apply patches only when you have a need for it—encountering bugs or taking preventive action for a future issue, etc. So, in summary, regular scheduled maintenance should include items that you deem important, have the lowest risks, can wait, and can be done with relative ease. Other unscheduled maintenance should be based on need or not time driven. But above all, understand what you are doing, entirely and accurately; there is no way around that one. ▲

Arup Nanda has been an Oracle DBA since 1993, covering all aspects of Oracle Database Administration from modeling to performance tuning to disaster recovery and security. He often speaks at conferences and writes technical articles, and he has co-authored four books on Oracle. He was the recipient of the DBA of the Year award in 2003.

Jeremiah Wilton: Sometime in the late '90s, *Oracle Magazine* ran an article entitled "A Day in the Life of an Enterprise DBA." The article presented a chronology that began with the DBA's arrival at work at 8 a.m. It followed him as he proceeded through a series of manual checks, procedures, and adjustments to his databases, all using Enterprise Manager. This was meant to represent a "typical day" for that DBA.

I found this article troubling. The DBA spent his time doing many tasks that should have been automated or avoided altogether through sound practices. Worse yet, the DBA in this Oracle-sanctioned article made many live off-the-cuff production changes to object storage, DDL, and data using GUI tools and a mouse. In other words, the DBA shot from the hip, making production changes in a way that was neither scripted, logged, nor peer reviewed. To me, this represented the antithesis of how a DBA should maintain a database.

In an ideally configured Oracle environment, there should be no routine manual maintenance tasks other than the patching and upgrading that the DBA must perform to maintain the optimal service level. Any manual operation that a DBA performs periodically should instead be scripted, tested, and automated. Furthermore, many of the most frequently seen automated maintenance tasks are unnecessary and needlessly rob the host of resources. For instance:

Gathering segment statistics should not be performed periodically. Many DBAs gather stats for all segments in their databases on a daily or weekly basis, consuming many resources and ending up with statistics that are 99% unchanged. Instead, statistics should be gathered on a segment only when it sustains significant data change. Since Oracle 8*i*, Oracle has been able to monitor the amount of DML performed on a segment. DBAs should implement automated jobs to detect significant change, and gather statistics accordingly. In 10gR1 and above, the default gather_stats_job does just this. Among the benefits to this on-demand approach are reduced resource consumption from stats gathering and improved stability from less frequent changes to statistics.

> ***"Exactly what is understood to be maintenance differs from site to site and DBA to DBA, but rationale for action should be the same: Perform only those activities for which there is a valid technical or business rationale, and only do them at such times as it is necessary or advantageous to the organization."***

Periodic index rebuilds as commonly performed are generally unnecessary and often add no lasting benefit. With few exceptions, periodic index rebuilds fail to achieve the goals intended by the DBA, such as space reduction and performance improvement. A b-tree tends to approach a steady state with respect to level and leaf density depending on the particular dataset and DML profile. Commonly, rebuilding an index that has achieved this state temporarily compacts the index until it once again reaches a steady state. Therefore any space and performance improvements are fleeting. An exception is b-tree indexes on monotonically increasing keys. In such cases, applications may delete some but not all of the values in lower- (older-) order leaf blocks, causing those blocks to never have their space reused. In such cases, recording and monitoring of index entries per leaf (See Jonathan Lewis's method at **www.tinyurl.com/5q4bkw**) over time should allow an automated job to choose to coalesce (not rebuild) the index only when needed, not on an arbitrary schedule.

Upgrades and patchsets are clearly necessary for a variety of reasons, and are among the few legitimate activities that can be called "maintenance." However, a DBA should not undertake any upgrade or patch without a compelling technical or business reason. For instance, a company may choose to remain on an old version of Oracle for as long as it is supported if they are not encountering version-specific bugs or are not beset by limitations that newer versions overcome. Those companies that do choose to upgrade or patch should do so not merely because the new version or patch exists, but because they can show real technical reasons that justify the disruption that arises from these activities.

Even those manual tasks that are occasionally necessary in the most automated environment are not really regular maintenance. They are activities undertaken at irregular intervals for specific technical reasons at a specific time in reaction to specific emerging or anticipated conditions. In this respect they are no different from any other work a DBA or other IT professional performs. Exactly what is understood to be "maintenance" differs from site to site and DBA to DBA, but rationale for action should be the same: Perform only those activities for which there is a valid technical or business rationale, and only do them at such times as it is necessary or advantageous to the organization. ▲

Jeremiah Wilton founded ORA-600 Consulting in 2004. Before becoming independent, he spent seven years at Amazon.com, initially as their first database administrator. Using expertise he gained during Amazon.com's period of exponential growth, he now specializes in scalability, high availability, stability, and complex recoveries. Jeremiah is a member of the Oak Table Network, has presented at numerous conferences and user group meetings, and is the author of a variety of technical whitepapers and articles. He also co-teaches the certificate program in Oracle at the University of Washington. His publications are available at **www.ora-600.net**.

Cary Millsap: Physical machines have parts that wear out. Software doesn't wear out, but from time to time it can need a human hand to help it adapt to the changes that periodically intrude upon it.

Your database changes all the time. Every insert, update, and delete changes the world in which your database server must function. With Oracle software, you do need to perform some types of regular maintenance in response to those changes.

One type of regular maintenance that databases need is software patching. No software that is as complex as a database manager is perfect; there are defects inside. As your software manufacturer finds and fixes those defects, you're going to benefit from some of the patches that result.

Another type of regular maintenance that some data-bases need is gathering of statistics that the optimizer uses to make its decisions about query plan assignments. That gathering process is not yet completely foolproof, and so it requires some human decision-making and periodic attention.

One topic of regular maintenance that I should mention specifically because there is so much discussion about it out there is index rebuilding. That's an area where a lot of people think their database needs regular maintenance, but it really doesn't. Richard Foote, Jonathan Lewis, and Tom Kyte have some excellent articles explaining why.

Over time, as Oracle adds more features to the database software, the requirement for more and more kinds of regular maintenance will diminish, and probably eventually disappear. An easy example from history is Oracle's approach to undo storage. In version 6, I spent a lot of my time as a DBA rebuilding people's rollback segments. That kind of thing is no longer necessary, because with rollback segments, Oracle Corporation has evolved its software to eliminate the need for so much human intervention. ▲

*Cary Millsap is the founder and president of Method R Corporation (***www.method-r.com***), a company devoted to genuinely satisfying software performance. He is widely known in the Oracle community as a speaker, educator, consultant, and writer. He is the author (with Jeff Holt) of* Optimizing Oracle Performance, *for which he and Jeff were named Oracle Magazine's 2004 Authors of the Year. He is also a contributor to* Oracle Insights: Tales of the Oak Table. *Cary is the former vice president of Oracle's System Performance Group, and a co-founder of Hotsos. Cary is also an Oracle ACE Director and a founding partner of the Oak Table Net-work, an informal association of Oracle scientists that are well known throughout the Oracle community.*

Mogens Nørgaard: Yes, vessels, marriages, and many other things need regular caretaking. But things made of 0s and 1s should (in my view) not need that much regular maintenance or massage. Bits don't get tired and become 0.2s.

However, that is of course not the whole truth. There is maintenance and there is maintenance.

When it comes to checking for space, for instance, it's not as important as it used to be, since various automatic mechanisms now exist. Checking for backups and messages in error logs can be automated. And if you know what you're doing, you can pretty much automate performance checks these days.

The goal is to change maintenance from being something you check (to see if something is wrong) as often as you can (or care to), and into some trigger-based procedure that sends out a strong signal whenever something unpleasant happens or is about to happen.

So much for my theories and dreams. Now back to the real world...

We (Miracle) have a service where we check databases for customers weekly. The guys log in, check stuff, and send the customer an email with observations and suggestions.

That should of course not make any difference according to my rant above, but it seems to, nevertheless. Time and again we have status meetings with the customers where they say something like, "Well, another three months have gone by without us having any trouble with the databases. . . . "

I don't know why. Perhaps databases need a little nursing now and then after all.

In the big hosting centers, small customers' databases are not checked or maintained, and they've assigned 100 databases to each DBA, which means no manual maintenance is possible. The procedure is this:

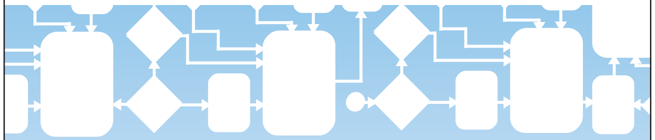
1. Automate everything and leave the database alone, preferably forever or until an upgrade is essential.
2. The customer will detect if there is a problem.
3. Train the call-center or on-duty guys to reply "Yep, we know, we're on the case." (I wish I were making this one up. I'm not. They're being trained to lie.)

So there. Many more databases than before. Fewer people to maintain them. Oracle's automation initiatives take care of some stuff, many databases get by just fine by themselves, and some databases still need that gentle "Are you OK?" greeting on a regular basis. ▲

*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**.

Andrew Kerber: Even though Oracle 11*g* claims to have automated many of the well-structured DBA tasks, the professional Oracle DBA will always be needed, especially for OS-level chores. Many senior DBAs develop procedures to automate many of the more complex tasks such as managing space on the filesystem and automating Oracle performance tuning. To date, Oracle has announced the automation of many of the mundane and well-structured DBA tasks, and Oracle Enterprise Manager has allowed the DBA to monitor dozens of important metrics and send automated alerts. But it's not just the monitoring that can be automated, and Oracle has many automated features:

- ➤ Automated Memory Management (AMM). Oracle changes the sizes of SGA regions based on current processing needs.
- ➤ Automated Workload Repository (AWR). The AWR allows the DBA to automate performance benchmark testing, using real-world workloads.
- ➤ Automated Storage Management (ASM). With ASM, the DBA is relieved of the tedious tasks of disk load balancing and I/O monitoring.
- ➤ The more that DBAs can automate, the more time they have to pursue those complex DBA tasks that cannot be automated. The wise DBA will automate many of the well-structured and repetitive daily maintenance tasks as possible:
- ➤ Daily backup verification
- ➤ Space usage delta reports (datafiles reaching maxsize, tables reaching maxextents, filesystems becoming full)
- ➤ Auditing exception reports
- ➤ Identifying candidate indexes for rebuilding (by parsing the Oracle segment advisor output)
- ➤ Trace file and alert log monitoring to escalate for specific messages



The above is a very short list of tasks that could be automated, but DBAs with shell scripting skills can take automation to the "George Jetson" level, where they simply press a button and let the server perform all of the Oracle monitoring and space management tasks. It's also important to note that many of Oracle's "fully automated" features do not do as good a job as a human DBA. For example, human DBAs will often outperform Oracle's Automated Memory Management because they know the circadian rhythms of the database (trends by hour of the day and day of the week). And there are other high-level DBA tasks that can never be fully automated:

- ➤ Examining trace and dump files
- ➤ Determining the root cause of a performance slowdown
- ➤ Searching for bugs on MetaLink
- ➤ Managing database security privileges
- ➤ New to Oracle 11g, we see some new DBA automation features:
- ➤ SQL Performance Analyzer (SPA). The SPA is marketed as "fully automated SQL tuning," whereby the DBA uses proven statistical methods to test global database changes and implement improvements to only those SQL statements that see a 3x improvement in response time.
- ➤ Automatic Diagnostic Repository (ADR). ADR can be used when critical errors are detected to create what Oracle terms an "incident," a bundle of information that is sent to Oracle Support Services. In addition, primitive health checks can be configured to run automatically and can be created to send to Oracle support. This final option has its own name: the Incident Packaging Service (IPS).

While each release of Oracle brings on new automated features, the human DBA will often do a better job, especially for semistructured tasks such as SQL tuning and performance troubleshooting. As each release of Oracle brings new automation, the DBA is free to spend time pursuing those DBA tasks that require skill and intuition. ▲

Andrew Kerber is an experienced Oracle RAC DBA consultant and an Oracle ACE. In addition to his bachelor of science degree in computer science from the prestigious United States Military Academy at West Point, Andy is an Oracle Certified Professional with more than 10 years experience working with mission-critical Oracle databases.

NoCOUG presents

Jonathan Lewis Training Days

Register at www.speak-tech.com

TROUBLESHOOTING AND TUNING

November 11, 2008

Troubleshooting or Tuning: What's the difference? What are the strategies? Why tuning is hard but troubleshooting is easy. Key targets, indicators, and mechanisms for producing a well-tuned system on day one. Strategies for dealing with badly performing systems after go-live.

Frequently Occurring Problems: Some of the most commonly occurring issues that affect performance after a system has gone into production. Methods for spotting them, measuring the impact, and dealing with the cost/risk/benefit triangle involved in fixing them. Getting into the habit of pre-emptive analysis and proactive fixing.

Quick Fixes: Methods, workarounds, dirty tricks, and parameters for dealing with classic performance problems when the system is in production. There really aren't many quick fixes that can be applied across the board—each one needs careful examination of costs, risk, and benefits. In this session we consider some of the options that are most likely to be worthwhile.

V$ and X$: It's a good idea to be familiar with just a few of the dynamic performance views—and there are a couple of items still hidden away in the X$ objects that can add a little value. This session will describe the views that are most commonly of use, and explain the meaning of some of the more useful items.

WRITING OPTIMAL SQL

November 12, 2008

Background: We will examine the need for a global viewpoint and a general strategy for minimizing work while recognizing that there will always be a conflict between local and global optimization. A quick review of the use of indexes, including some of the less-common uses and common errors in index design. Closing with the need to be able to investigate and understand the data and business requirements.

Basic Practices: How to approach the task of translating business requirements into SQL. The need for clarity in code, with a suggested set of standards for presentation to improve ease of understanding. An introduction to the essential strategy for writing SQL that gets its result by doing the minimum work, and a model for investigating and re-engineering poorly performing statements.

Sundry Methods: A few examples demonstrating classic problems and their solutions that use some of the less-well-known features of SQL. A brief discussion of the mechanics (and costing) of nested loop and hash joins. Warnings about the abuse of views and the problems of predicates. And a few closing comments about tuning distributed queries.

More Methods: Fixing the code without touching the code—the latest option. Problems and options with knowing internal mechanisms and taking advantage of them. A little time with OEM, and a look at the threats of hints and how to use them properly. ▲

EARLY-BIRD PRICING (UNTIL SEPTEMBER 19)

One day: $350 for non-members,
$300 for NoCOUG members
Both days: $650 for non-members,
$550 for NoCOUG members

REGULAR PRICING

One day: $375 for non-members,
$325 for NoCOUG members
Both days: $700 for non-members,
$600 for NoCOUG members

GRC and You

by John Weathington

John Weathington

It's no secret that Governance, Risk, and Compliance (GRC) is a big concern in the world today; however, the best-kept secret of top companies is how they employ their database professionals to not only respond to compliance but also to leverage their data systems in a way that produces a more compliant, ethical, and profitable enterprise.

As a compliance consultant, I see companies make all kinds of mistakes. However, the biggest mistake I see is the underestimation of how valuable their data professionals are to the organization's compliance efforts. This is in part due to ignorance and in part due to arrogance. God forbid, they confuse IT with all their vainglorious methods and syntax. When I tell people I'm a compliance consultant, I'm often answered with, "Oh, are you a lawyer?" or, "Oh, are you an accountant?" I just smile while internally shaking my head and respond, "No, most people think it's a legal [or financial] problem. That's what keeps me in demand."

It's Not about the Lawyers, and It's Not about the Accountants

The reality is, it's not a legal or accounting (in the case of Sarbanes-Oxley) issue. I'm not saying the lawyers and accountants aren't necessary in the equation, I'm just saying their part is overemphasized in the industry. Consider this: We've had financial regulation since the early 1930s with the Securities Act of 1933 and the Securities Exchange Act of 1934. So why was there such a meltdown with companies like Enron and WorldCom bringing the nation to a full boil over the egregious lies and accounting scandals? This blatant abuse of corporate power sent people to jail, and the accounting collusion took the "Big Five" to the "Big Four" when Arthur Andersen was publicly forced to surrender its license to practice. What happened to all the legal counsel? What about the financial guidance?

If proper GRC had been in place at Enron, there's no way this could have happened. This of course was the seminal impetus for the Sarbanes-Oxley Act of 2002 (more "affectionately" known as SOX). By the way, the SOX act zipped through Congress with a bipartisan vote. For some reason, our entire government body thought the solution to this "outrage" would be more regulation and oversight!

Accountants and lawyers came out in droves to "help" Corporate America with the new legislation. Companies then proceeded to hemorrhage money on SOX compliance efforts. I was working with a high-tech company at the time that ended up spending hundreds of millions of U.S. dollars on SOX-related activities. If you work for a big company, I'll bet your story is the same. The amount of money spent on SOX is unbelievable.

You would think that with all this renewed vigor and diligence around financial controls, guidance provided by the government and re-issued and interpreted by the accountants and lawyers with their fancy suits and solid oak conference tables that fit 50 people, there's no way a meltdown like this could happen again, right?

Well, welcome to the 2008 mortgage meltdown and credit crisis. I hope you weren't holding any Bear Stearns stock in your portfolio! I personally have Citigroup, and I'm not a happy camper right now. How in the world can this happen? It's only been six years since SOX!

Now, Henry Paulson, our treasury secretary, is on a warpath to . . . wait for it . . . wait for it . . . overhaul the financial regulatory system! You have got to be kidding me! When do you think they'll get it right? I wouldn't hold my breath.

Recently, Harvey Pitt, the former chair of the SEC, addressed a group of compliance officers on the topic. He said that the regulatory landscape in the United States today is confusing and needs to be simplified. I couldn't agree more.

The Best Kept Secret to Effective Compliance

Here's the moral of the story: Over-reliance on the lawyers and accountants to solve your compliance problems is a mistake—as I said, the biggest one I see. I'm qualified to make that statement, because I've seen both sides of the issue. I've seen companies undervalue IT in their compliance efforts and fail miserably while simultaneously spewing loads of cash. I've also seen companies engage IT properly—especially their data professionals—and succeed within a handsome margin, while keeping a lot of their cash in their pockets.

Here's why the data professional is so important to the mix. You, and only you, have the knowledge to design systems that empower and enable proper GRC. Your data systems cannot do it by themselves, but this is the weak link in Corporate America, as I see it.

A company needs three things working in tandem to achieve proper GRC:

1. People that are intelligent, ethical, and responsible. A company needs good people that can build good policy, administer good policy, and most importantly execute good policy.
2. Processes that are built to support the policies. Good process control is essential. Processes need to be documented, monitored, and improved on a regular basis.
3. Data systems that support the people and processes that make your compliance work. Without these data systems, it's not going to work. In the old days, we relied on efficient file systems. Today, we rely on databases. Since databases don't build themselves, it's up to you, the data professional, to ensure that your company has the data architecture necessary to support its compliance needs.

You need all three components to be working properly in order to achieve proper GRC; however, here's my feeling: In general, people are good. We go to school to learn the things we need to know, and we work hard at our jobs under the right conditions (good pay, good leadership, nicely defined set of responsibilities, etc.). There is a segment of the population that will be unethical; however, with proper GRC in place, these people can be flushed out.

Also, process control is achievable by most, because it's largely intuitive. Even if you have only a high school education, I'm sure you can figure out how to map out a process. And even though process improvement methodologies such as Six Sigma take an advanced set of knowledge, you can probably figure out how to improve a process on your own just by taking the time to understand it.

The challenge comes with architecting a proper data system. There's nothing intuitive about this at all, and there's a significantly large knowledge barrier to entry. I'm always amused to come across a finance person who has created a Microsoft Access database—or worse yet a Microsoft Excel spreadsheet—to manage their compliance. And then they don't understand when things start falling apart, as if business intelligence is something they can just figure out on the fly. I've spent the last 15 years learning and implementing business intelligence solutions, and I'm still learning. So you can't tell me a finance person is just going to wing it. This is part of the ignorance and arrogance in the industry that I was talking about.

GRC—A Gentle Introduction for the Data Professional

As a data professional, you can make the difference in your company's compliance efforts. You can do this by first understanding the GRC landscape and then advocating and constructing what I call a Compliance Data System (CDS). I'll introduce the CDS concept shortly, but first let's get you acclimated to the GRC waters.

As noted above, GRC stands for Governance, Risk, and Compliance. Obviously, each word has its own meaning; however, the compliance industry has recognized a certain synergy of efforts between the three, so you'll often see them addressed as one combined effort. You'll also see the word "compliance" sometimes used in the context of GRC. For instance, I tell people, "I help companies improve their contractual and regulatory compliance." However in reality I deal with all aspects of GRC.

Governance is about administration and control. It's the processes and policies that a corporation develops to make sure it's achieving its mission and goals. Governance is also tied to performance and return on investment (ROI). For instance, there is a big trend in business intelligence governance now. This simply means understanding and directing the ROI for your business intelligence investment and systematically migrating it through maturity.

Risk is uncertainty. Managing risk means managing uncertainty. There's a common misconception that risk only involves unfavorable events. This is not true. There is a concept of positive risk, which means that something unexpected and fortunate happened. Mitigating risk means taking deliberate action to lessen the probability, increase the visibility, and/or reduce the impact of a risk-related event.

Compliance is making sure that stated policy is adhered to. This is more specific and tactical than governance. Compliance is usually either regulatory (complying with laws like Sarbanes-Oxley, HIPPA, or PCI) or contractual (complying with contract terms like GSA or royalties). GSA compliance is a universal contractual concern for most companies that want to do business with the U.S. government.

Compliance Data System (CDS) is the term I use for the key data architecture in your company that supports its compliance efforts. Over the course of this series, we will discuss some architectural considerations for realizing this; however, basically, the CDS is a data system for the internal auditors. It makes their life easy, keeps external auditors in their place, and is the platform for solid corporate performance.

Summary

I hope I've been able to build the case for the data professional in the compliance arena and give you a gentle introduction to the GRC world. Understanding that data systems are the weak link in Corporate America is the key, not only for the data professional but also for all the decision-makers at your company. In the articles to come, we'll explore the Compliance Data System and hopefully give you a number of ideas that you can bring to the compliance table at your company. In the meantime, start evangelizing your contribution and value to the company's compliance effort. It's in their best interest to listen. ▲

John Weathington is a management consultant who helps companies improve their regulatory and contractual compliance. His San Francisco Bay Area-based company, Excellent Management Systems, Inc., has helped companies such as Sun Microsystems, Silicon Graphics, Hitachi Data Systems, and Hogan and Hartson, LLP, all over the world. He is a Project Management Professional (PMP) and a Six Sigma Black Belt, as well as an Oracle DBA and Business Intelligence Architect. He runs an expert blog for Quest Software called, "John Weathington's Quest for Compliance," a monthly newsletter called Flawless Compliance™ *and a public blog called "Hard-Boiled Compliance." For more information, please access his website at* **www.excellentmanagementsystems.com**.

JPA Is Your Friend

by Joel Thompson

Joel Thompson

What is JPA? And why should I care? These are questions that you might have asked yourself in the last two years since JPA hit the scene. First, JPA stands for Java Persistence API. JPA is one of the hottest parts of Java these days and should not be overlooked. JPA is a standard interface defined by Sun in June 2006 that is used to map your RDBMS objects with plain old Java objects (POJOs) using annotations or XML, and provides a rich SQL-like query language. Another benefit is that you can swap out pluggable persistence providers from different vendors, and thus you are not locked into a particular vendor. Oracle has Toplink technology as its persistence provider. JPA also can run inside a Java EE container or in standalone applications. There, JPA takes the place of Entity Beans in EJB 2. In this article we'll examine how to get started quickly with JPA, we'll discuss some of the basics, and then we'll tackle a more complex scenario (One-to-Many and Lazy vs. Eager loading) that may be useful for the intermediate or advanced users of JPA.

A Standalone Application

First, I'll explain how to set up a standalone application (non JEE/EJB), since that is often not covered on the Net. The basic steps go like this:

1. Create a project directory.
2. Create a persistence.xml file.
3. Create a single POJO class, calling it Person.java, adding a few annotations to map to the RDBMS table representation of Person and a main routine (in package com.rhinosystems.pojos, to follow with the examples provided in this article).
4. Log in to your RDBMS and create the Person table.
5. Create a Jar file deployment.
6. Run your program in a DOS prompt from <project> directory by calling "java com.rhinosystems.pojo.MyMain".

Step 1:

Create a project directory c:\projects\sample (aka <project>) with these subdirectories (just so we can have consistency throughout this article's examples): <project>\deploy\META-INF and <project>\src\com\rhinosystems\pojo.

Step 2:

Here is a sample persistence.xml for Toplink. Create this file and move into your <project>\deploy\META-INF directory. Notice that you must change out your database-specific login information and jdbc url. Also notice the persistence-unit name is "default"; that will match what we use in our MyMain.java class.

```
<?xml version=”1.0” encoding=”windows-1252”?>
<persistence xmlns:xsi=”http://www.w3.org/2001/
XMLSchema-instance”

xsi:schemaLocation=”http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd”
             version=”1.0”
xmlns=”http://java.sun.com/xml/ns/persistence”>
<persistence-unit name=”default” transaction-
type=”RESOURCE_LOCAL”>
          <provider>
             oracle.toplink.essentials.PersistenceProvider
          </provider>
          <class>com.rhinosystems.pojos.Person</class>
          <properties>
                <property name=”toplink.logging.level”
value=”FINE”/>
                <property name=”toplink.jdbc.driver”
value=”oracle.jdbc.OracleDriver”/>

                <!-- update to match database-->
                <property name=”toplink.jdbc.url”

value=”jdbc:oracle:thin:@localhost:1521:XE”/>
                <property name=”toplink.jdbc.password”
                         value=”tiger”/>
                <property name=”toplink.jdbc.user”
value=”scott”/>
       </properties>
</persistence-unit>
</persistence>
```

Step 3:

In JDeveloper (10.1.3), create a new Application named "projects" (give it a directory of C:\projects) and select "Choose No Technologies." Then it will ask you for a Project name: call it "sample". Right-click your "sample" project select properties and set up your Input default package name as "com.rhinosystems.pojos" (or whatever you want, but make it consistent throughout this exercise). Choose c:\projects\example_project\deploy as your Output Directory. Also be sure to add "Top Link Essentials" to your Libraries.

Right-click the "com.rhinosystems.pojos" package and create a new Java class file for Person by selecting "New . . ." then "Simple Files" and Java Class. Name it "Person".

Next add some attributes and annotations to make this Person POJO a JPA-enabled class.

```
package com.rhinosystems.pojos;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Id;
```

```
@Entity
@Table(name=”PERSON”)
public class Person {

/* I like to declare my primary keys with manufactured
surrogate
 * keys, based on Oracle’s sequences.  This method will
work with Toplink and
 * Hibernate.
 * Make sure to create the sequence PERSON_SEQ.
 */
    @Id
    @SequenceGenerator(name = “PERSONidentifier”,
                     sequenceName = “PERSON_SEQ”,
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
                     generator = “PERSONidentifier”)
    @Column(name = “PERSON_ID”, nullable = false)
    private Long personId;

    @Column(name = “F_NAME”)
    String fname=null;

    @Column(name = “AGE”)
    Integer age=null;

    public Person() {
    }

    public void setPersonId(Long personId) {
        this.personId = personId;
    }

    public Long getPersonId() {
        return personId;
    }

    public void setFname(String fname) {
        this.fname = fname;
    }

    public String getFname() {
        return fname;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Integer getAge() {
        return age;
    }
}
```

Create MyMain class to run the main(...) routine. (Refer to how you created the Person class for JDeveloper.)

```
package com.rhinosystems.pojos;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class MyMain {
    public MyMain() {
    }

    public static void main(String[] args) {

        // Create EntityManagerFactory for persistent
unit named “default”
        // to be used in this sample - notice
“default” in the persistence.xml
        // file.
        EntityManagerFactory emf =

Persistence.createEntityManagerFactory(“default”);

        // Create new EntityManager
        EntityManager em = emf.createEntityManager();

        //        If a transaction required, it is
started:

        // Begin transaction
        em.getTransaction().begin();

        // business logic here:
        Person p = new Person();
        p.setFname(“John”);
        p.setAge(new Integer(16));
        em.persist(p);

        // Commit the transaction
        em.getTransaction().commit();

        //        Close EntityManager if not used again:
        em.close();
        emf.close();
    }
}
```

Step 4:

Create the database schema.

Log in with the credentials you referenced in your persistence.xml and create the following objects:

Person Sequence:

```
create sequence person_seq;
```

Person Table:

```
create table person (
        Person_id number primary key,
        F_name varchar2(32),
        age  number);
```

Step 5:

Deploy and run the application.

Right-click your “sample” project and select “New...”, and then select “Deployment Profiles -> JAR File”. Name it “myjar” in the default project directory. Change the output Jar file property to c:\projects\sample\myjar.jar, and all the rest of the defaults for the JAR Deployment Profile Properties are fine (you should notice that under the “filters” you’ll see your persistence.xml listed).

Deploy your Jar file by right-clicking it and selecting “Deploy to Jar file”.

Step 6:

You need to have a few files in your classpath in order to run this program:

<oracle>\ojdbc14.jar
<jdev>\j2ee\home\lib\persistence.jar
<jdev>\toplink\jlib\toplink.jar;
<jdev>\\toplink\jlib\toplink-essentials.jar
c:\projects\sample\myjar.jar
Example (all on one line):

```
set
CLASSPATH=C:\oracle\10g_client\jdbc\lib\ojdbc14.jar;C:\
jdevstudio10133\j2ee\home\lib\persistence.jar;C:\
jdevstudio10133\toplink\jlib\toplink.jar;C:\jdevstudio10133\
toplink\jlib\toplink-essentials.jar;.\myjar.jar
```

Then run your program while in a DOS window with the following command:

```
C:\> java com.rhinosystems.pojos.MyMain
```

A Many-to-One Example

Now that you have the basics working, we can add to the project that you’ve created and demonstrate a One-to-Many relationship with loading and querying.

In this example we’ll do the following:

1. Add a new class called Car.
2. Add a new Car table and sequence.
3. Add annotations to Person to make them linked together.

4. Modify MyMain to create a new Person with multiple cars and query all the Person objects with Cars preloaded.
5. Modify the persistence.xml to add the Car class.
6. Run the program.

Step 1:

Create the Car class.

```
package com.rhinosystems.pojos;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
@Table(name=”CAR”)
public class Car {
    public Car() {
    }
    @Id
    @SequenceGenerator(name = “CARidentifier”,
                       sequenceName = “CAR_SEQ”,
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
                    generator = “CARidentifier”)
    @Column(name = “CAR_ID”, nullable = false)
    private Long carId;

    @Column(name = “MAKE”, nullable = false)
    String make=null;
    @Column(name = “MODEL”, nullable = false)
    String model=null;

    @ManyToOne
    @JoinColumn(name = “CAR_PERSON_ID”,
            referencedColumnName = “PERSON_
ID”,nullable=false)
    private Person person;

    public void setCarId(Long carId) {
        this.carId = carId;
    }

    public Long getCarId() {
        return carId;
    }

    public void setMake(String make) {
        this.make = make;
    }

    public String getMake() {
        return make;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getModel() {
        return model;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    public Person getPerson() {
        return person;
    }
}
```

Step 2:

Add new Car table and sequence.

```
Create table car (
       Car_id number primary key,
       Car_person_id number,
       Make varchar2(32),
       Model varchar2(32));
```

Create sequence car_seq.

```
alter table car add constraint fk_person foreign key
(car_person_id)
references person(person_id);
```

Step 3:

Add the following code to the Person class:

```
    @OneToMany(mappedBy = “person”,
               fetch=FetchType.EAGER,
               cascade = { CascadeType.ALL })
     private List<Car> carList;

    public void setCarList(List<Car> carList) {
         this.carList = carList;
     }

     public List<Car> getCarList() {
         return carList;
     }
```

Step 4:

Modify MyMain to create a new Person with multiple cars, and query all Person objects with multiple cars.

```
package com.rhinosystems.pojos;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class MyMain {
    public MyMain() {
    }

    public static void main(String[] args) {

        // Create EntityManagerFactory for persistent
unit named “default”
        // to be used in this sample - notice
“default” in the persistence.xml
        // file.
        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory(“d
efault”);


        // Create new EntityManager
        EntityManager em = emf.createEntityManager();

        //        If a transaction required, it is
started:

        // Begin transaction
        em.getTransaction().begin();

        // business logic here:

        Person bob=null;

        //create query to see if Bob already exisst,
if not then add.
        Query qBob=em.createQuery(“select p from
Person p” +
        “ where p.fname=?1”);

 //bind parameter
        qBob.setParameter(1,”Bob”);

        try
        {
           bob = (Person)qBob.getSingleResult();
        }catch(javax.persistence.NoResultException e){
            //ignore since we’ll create ‘em if not
there.
        }

        //Bob doesn’t exist, so add him.
        if (bob == null) {
            bob = new Person();
            bob.setFname(“Bob”);
            bob.setAge(new Integer(35));
            Car bmw = new Car();
            bmw.setMake(“BMW”);
            bmw.setModel(“320”);
            bmw.setPerson(bob);

            Car volvo = new Car();
            volvo.setMake(“VOLVO”);
            volvo.setModel(“x140”);
            volvo.setPerson(bob);
            ArrayList cars = new ArrayList();
            cars.add(bmw);
            cars.add(volvo);

            bob.setCarList(cars);

            em.persist(bob);
            // Commit the transaction
            em.getTransaction().commit();
        }


 //now query all Person records in database, and
       // notice that the Cars are preloaded! SQL-like
syntax.

        Query q=em.createQuery(“select p from Person
p”);
        List<Person> list=q.getResultList();

        for(Person p:list) {
            System.out.println(“First Name “+p.
getFname() + “ ownes:”);
            for(Car c:p.getCarList()) {
                System.out.println(“\tMake:”+c.
getMake());
                System.out.println(“\tModel:”+c.
getModel());
            }
        }
        //       Close EntityManager if not used
again:
        em.close();
        emf.close();

    }

}
```

Step 5:

Modify the persistence.xml to add the following:

```
        <class>com.rhinosystems.pojos.Car</class>
```

Step 6:

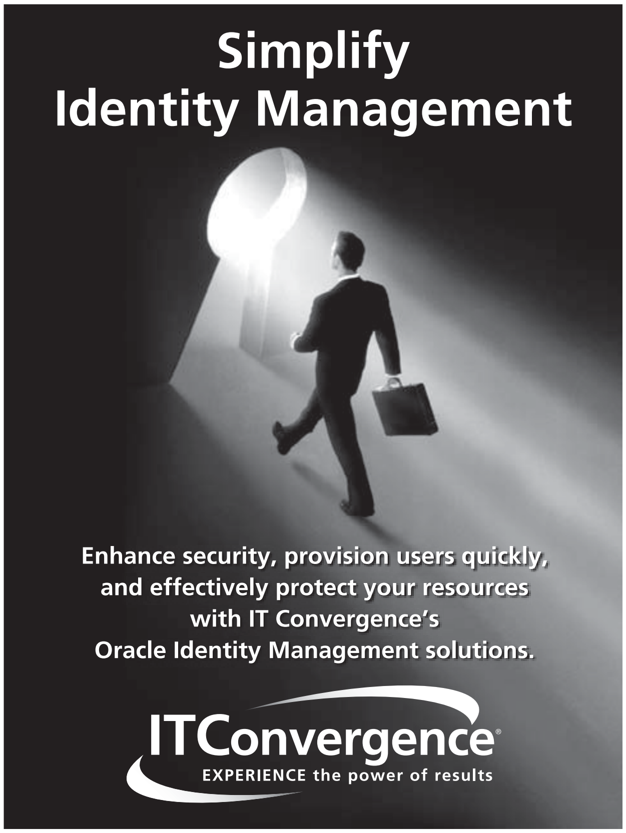
Follow Steps 5 and 6 of the previous sample to run this program.

The beauty of what we just accomplished is that we can create Java objects and associate them to other objects all through Java POJO code, and we can run a simple query and pull up all related objects automatically. If you declare the list of cars to lazy load, then you won't get any cars when you query the Person until you access the list, and then JPA will automatically load the cars at that time. However, a word of warning about this: if you serialize the Person object so that it is no longer managed by the EntityManager, then you will get a LazyInitializationException. What if you want to declare Lazy, yet load the cars sometimes and make them available to a remote jvm? You can run an outer join query to query up all the cars for this person. An example of this query would be:

```
 “SELECT p from Person p LEFT” +
              “ OUTER JOIN FETCH p.carList WHERE
p.fname=?1”
```

I hope you enjoyed this hands-on article about JPA and that it has piqued your interest in the topic. If you have any problems going through the example code above, please feel free to email me and I'll do my best to help. ▲

Joel Thompson is currently working as a senior software engineer at EBay; however, please use Joel's personal email at **joel@rhinosystems.com**. *He has programmed in Java/J2EE since 1997, and prior to that he worked as senior programmer and development manager at Oracle Corporation since 1989. He resides in Auburn, CA, with his wife and three children, and enjoys snowboarding, jogging, tennis, and many more outdoor activities.*


Simplify
Identity Management
Enhance security, provision users quickly,
and effectively protect your resources
with IT Convergence's
Oracle Identity Management solutions.
ITConvergence®
EXPERIENCE the power of results

Fundamentals of Data Quality—Part III

by Michael Scofield

Michael Scofield

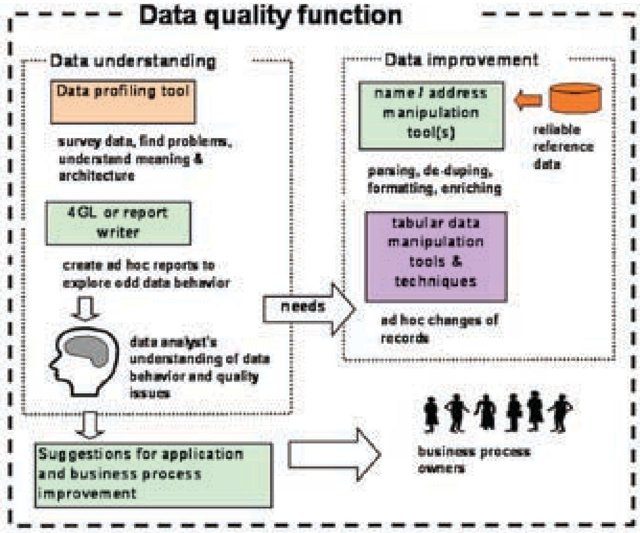
In the first installment of this series, we talked about how data quality as a business function fits within a wider collection of roles and responsibilities such as data architecture and data governance. We also established some useful definitions of various aspects of data quality such as accuracy, validity, and reasonableness.

In the second installment, we looked at simple techniques for data quality assessment and understanding data behavior. We saw that watching and testing for many errors in DQ can be accomplished with query tools or report writers that you may already have.

Now we need to understand what we mean by data quality tools, what functions they might perform, and how to match those functions with our expectations.

Understanding vs. Improvement

First, we need to distinguish between looking at the data and understanding it, and actually making changes to the data that improve it. The former ("data understanding," shown on the left side of the sketch below) is centered on the role of a human. That may be a data steward, data analyst, or DQ analyst. The output of these processes is knowledge and understanding. That person ultimately must understand the essential characteristics of data behavior in production databases and files.


Data quality function
Data understanding
Data profiling tool
survey data, find problems, understand meaning & architecture
4GL or report writer
create ad hoc reports to explore odd data behavior
data analyst's understanding of data behavior and quality issues
needs
Data improvement
name / address manipulation tool(s)
reliable reference data
parsing, de-duping, formatting, enriching
tabular data manipulation tools & techniques
ad hoc changes of records
Suggestions for application and business process improvement
business process owners

The human mind cannot possibly comprehend patterns out of millions of individual facts. Here is where DQ tools (or query tools) can help by giving visibility to patterns and anomalies. But decisions about the cost or severity of a data problem and whether to fix it or not remain in the realm of a human decision-maker; no tool can do that.

The right half of the sketch addresses "data improvement" —that deliberate modification of data values to improve them. There are two types of data that can be improved. Name-and-address improvement (based upon comparison with authoritative reference files for addresses and names) is what most vendors mean when they say "data quality."

But the vast majority (by volume) of corporate data is not names and addresses. Included are a host of other facts about business events, products, customers, vendors, inventory, processes, and market conditions that usually have no other reference data to validate them against. Indeed, most corporate data has no backup of the observations. It is the only version of "truth" we have.

It's important to understand the limitations of what DQ tools can do, and we must carefully establish and document our expectations of DQ tool functions in very specific ways. There can be no magic in data quality improvement. We must be able to understand and measure what goes on in every DQ function and sub-function.

Functional Requirements Document

Don't go looking for any DQ software nor commit to buying such software without first creating a detailed functional requirements statement. Such a statement should answer the following questions:

- ➤ Are you going to use this tool for name-and-address or non N&A data?
- ➤ Are you merely surveying data or do you intend to change (and hopefully improve) data?
- ➤ Who is going to be using these tools?
- ➤ What skills will they require?
- ➤ In what technological environment will they be used?
- ➤ What kind of data (structural and content) will they be used on?

Batch or online, or both?

A functional requirements document can be supplied to potential vendors to let them know what you are looking for and how you use critical terms and expressions. Such a document may be informed by reading vendor literature or seeing demonstrations. You may learn about a function you had not considered, but no vendor can write the functional requirements for you—and don't let any vendor constrain or define your thinking and lexicon.

Functions vs. Tools

Don't confuse DQ tools (as defined and named by the vendors) with the specific DQ functions that you want to perform. Indeed, some of the names created by vendors for marketing purposes can be confusing, and a particular word may mean one thing to one vendor and something else to another.

Some vendors pack functions too tightly. Each distinct function (as discussed below, for example, in name-address improvement) should have a measurable input and output so you can see what is happening inside the box.

So you must establish (in your functional requirements) your own precise lexicon of what a tool will do. It helps by starting at the general functional categories of "data understanding" and "data improvement."

Functionality in Data Understanding

In the previous installment we talked about how to find data anomalies through examination of the data through simple queries. When you have hundreds or thousands of fields, found on hundreds of files and/or tables, it can get tedious to write code (even in the most advanced query language). Here is where data examination or data profiling tools can help.

Similarly, grasping the inter-relationship (or correlation) between two fields can be exponentially tedious as the number of fields grows. Here again, a profiling tool that can read your data dictionary and set out a course of investigation, examination, and discovery of anomalies on its own can save much human labor.

Here are some questions to consider about "data understanding" tools:

- ➤ Will the tool be able to read all the corporate data, no matter what kind of DBMS or file structure it is stored in?
- ➤ What reports will the tool produce?
- ➤ Can you create custom reports in the tool and apply those reports to a variety of data?
- ➤ Will the tool perform well for the scale or magnitude of data that you have?
- ➤ Can the tool distinguish between dates, codes, and amount fields, and choose its tests appropriately (a median or mean is meaningless for a text or code field)?
- ➤ Does the tool work with your existing metadata repository? In what way? Or are the metadata it creates isolated from the rest of your metadata?

Another major question is whether the tool can reposit its findings in some kind of versioned repository, for what data survey and profiling create is a kind of "dynamic metadata"—knowledge about how production data behaves. The ability to compare periodic snapshots of the data can enable change analysis. For example, this can show how and why a table is growing so fast. That can be useful to application designers, data warehouse analysts, and so on, if they can find that information quickly.

Additionally, the tool should be able to detect anomalies (or outliers) and notify the data steward only when a problem is found. This would apply both to latent data in production databases ("data at rest") and data flows into the enterprise, exiting the enterprise, and between applications in the enterprise ("data movement").

To have a tool produce pages and pages of reports, most of which do not show anything meaningful, is a waste of time for the data steward. Rather, reports and reporting functions should be structured to show only those behaviors (particularly anomalies) that are important to consider. Some tools will do this.

Will the tool support a data surveillance program that monitors latent data periodically and data flows more frequently? Can the tool support the concept of "agent" to go out and watch critical data behavior (particularly data flows) when you are sleeping?

Functionality in Name-Address Improvement

Most of the DQ tools proposed by vendors are really name-address improvement. In addition to buying the software itself, you'll probably need to establish some kind of subscription to reference data that your names and addresses are checked against. The scope of the reference data is significant—if the scope of your data is international (multiple countries), you pay more and your processes will be more complex.

Name-address improvement involves a finite set of subfunctions, the most important of which are as follows:

Text parsing finds multiple logical elements of address and name in one or more text fields, dividing them, and placing each of them into its own distinct output field. This is most often done for name and address, but it's occasionally done on free-form comments text. The output should contain more columns, each of which contains one element of data, with consistent behavior over the entire column.

Address standardization involves replacing variants of known words and terms and expressions with standardized version (e.g., changing "Ave" to "Avenue"). This requires a list of the best expression of any concept (e.g., the best spelling).

Derivation of new data. Additional values or attributes are obtained based upon observed values (e.g., when you are missing a city name, using the zip code to deduce the city name). Such replacement decisions may or may not be driven by one or more authoritative reference files.

Matching and cleansing (name and address). Names and/or addresses are compared to an external authoritative reference file of addresses (e.g., the USPS's NCOA) with low-risk replacement of values in certain fields that are misspelled or improperly expressed, yet (with high confidence) preserving the correct meaning of those cells. The rules for element replacement may be fixed by some vendors or subject to your own tuning to achieve the highest output quality.

Transformation can be converting a fact from one format to another. Or, it may involve changing data from one language or character set to another, or converting from one domain of codes to another. An example of the latter is converting from a FIPS code of "06" (meaning California) to a postal code of "CA."

De-duplication involves detecting multiple records for the same instance in reality, with the option of deleting (now or later) all but one (perhaps the "best"). Determining just what the "best" version of data is can be a challenge. Rules of precedence are necessary, either by virtue of the reputation (for accuracy and/or currency) of each source or based on some date stamp of competing records (which was created more recently).

These functions are listed in the approximate order in which they would work on data, but this sequence of functions is not absolute. One question to ask is if you can resequence these functions if needed.

Augmenting and enhancement. The ability to take data records about a single instance (usually a person) from diverse sources and match them, either on a common identifier (relatively easy) or clues (often quite difficult) such as date of birth, social security number, and other facts (i.e., "matching"). If the matching you do is based on a key, you may not need to buy this functionality from a vendor.

Survivorship and merge. Taking known duplicate records (representing a single instance in reality) and creating one surviving record with the best (or most current, according to rules of confidence) data from those candidate duplicates, thus yielding a more completely populated record, which survives. This may involve picking and choosing individual facts from different source records. An example is taking the name from one record and the credit rating from another.

This is also known as building best of breed in the survivor. This is not to be confused with tracing the pedigree of records about businesses that merge in reality (with one surviving business entity in the real world). The merging and splitting of data about businesses is much more complex than data about humans, because humans are basically singular. Businesses can have subsidiaries, divisions, joint ventures, and many facets at a particular time, and can merge and split over time. Keeping up such a complex reality can be challenging.

So, name and address improvement involves both software and a set of reference data, not necessarily from the same source. Questions to consider in your functional requirements should include:

- ➤ What is the scope of your addresses, just the U.S. or international?
- ➤ Is this "scrubbing" a one-time event or an ongoing process (being constantly applied to new records, and periodically applied to legacy records)?
- ➤ What will be your relationship to the provider of the reference data?
- ➤ Will one provider cover all the countries you are interested in, or will you need several data providers?
- ➤ Will there be multiple countries on a single file, or will the address records be separated by country before the improvement processes?
- ➤ What character sets will you be working with? Does the tool support them?

And, of course, you must manage this process, preserving the original version of the name and/or address somewhere in case the appropriateness of the changes is challenged.

You need to be able to reverse or back out such processes, e.g., when you (or your DQ tool) mistakenly merge data about a father and son who have the same name and live at the same address. If a customer complains, you may have to restore the original data.

Duplicate Data

We need to discuss duplicate data a bit more. There are many kinds of data duplication, some deliberate and some inadvertent. Whether data duplication is good or bad depends upon business rules and expectations.

For example, in general we would say that having two records on the customer master file describing "Mary Smith" is probably a bad thing. We don't want to send her duplicate mailings.

Evaluating data duplication can be complex and involves data architecture and database design. If Mary Smith has two accounts with us, separating the "party data" from the "account data" is a good data design technique. But perhaps her relationship with us for each account is unique. For example, she may want one statement mailed to her PO box and the other mailed to her residence—thus, a legitimate reason for two records on the customer master file.

For several years, a major software company had me on its mailing list three times and I got three copies of its glossy color magazine every quarter. Expensive for them; annoying for me.

Actually, within a given table, there are several kinds of duplication:

- ➤ Totally identical records (two or more)
- ➤ Identical content, different keys
- ➤ Identical keys, different content describing different instances in reality
- ➤ Subtly different content describing what is actually the same instance

What is more problematic is when keys are not used or have no business meaning—that the estimation of duplication (or not) is based solely upon non-key data such as name. One horror story here is when persons with identical or similar names to terror suspects are placed on a "do not fly" list. Even if they are granted access to the aircraft, they may have to undergo additional security.

Conclusion

There are many vendors eager to sell you their data quality tools. They vary widely in scope of functions and ease of use, and their sales literature is often lacking in specifics of how each function works (input and output) and weak in giving simple examples. It really is a dismal situation for the buyer.

The most important thing the buyer can do is to have a clear understanding of the functional requirements of the tool and your expectations of what you want to accomplish. These need to be documented with use cases and examples of both the common situations and the anomalous situations (such as an international address).

Again, the tools exist to support a data quality function—a business function, which involves mature and deliberate thought on the part of a human (the data analyst) about what is to be done in each circumstance. Don't expect magic or miracles from any DQ tool.

In the next installment, we will look at integration of data from multiple sources, the many potential problems that can occur, and how to mitigate them through thorough analysis, modeling, and thoughtful design. ▲

Michael Scofield is manager of data asset development at ESRI, Inc. in Redlands, CA. He is also an adjunct faculty member at Loma Linda University in the Department of Health Information Management. He is the 2008 recipient of the DAMA International (Data Management Assn.) Community Award for his contributions to the education efforts of that organization.

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, Roger Schrag, at **www.nocoug.org/contact_us.html?recipient=roger**. ▲

Long-term event sponsorship:

LOCKHEED MARTIN

CHEVRON

ORACLE CORP.

PG&E

Thank you! Year 2008 Gold Vendors:

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

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

TREASURER'S REPORT

Jen Hong, *Treasurer*

Beginning Balance		
April 1, 2008		**$ 44,676.26**
Revenue		
Membership Dues	10,820.00	
Meeting Fees	440.00	
Vendor Receipts	5,900.00	
Advertising Fee	–	
Training Day	–	
Sponsorship	–	
Interest	53.17	
Paypal balance	–	
Total Revenue		**$ 17,213.17**
Expenses		
Regional Meeting	9,846.59	
Journal	6,076.21	
Membership	290.40	
Administration	1,432.13	
Website	–	
Board Meeting	235.25	
Marketing	100.00	
Insurance	–	
Vendors	44.40	
P.O. Box	–	
Training Day	500.20	
FTB	800.00	
Miscellaneous	–	
Total Expenses		**$ 19,325.18**
Ending Balance		
June 30, 2008		**$ 42,564.25**

NoCOUG Summer Conference

Session Descriptions

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

—Keynote—

Millsap's Grand Unified Theory of Tuning

Cary Millsap, *Method-R.com* 9:30–10:30

This is a break from the technical presentations, but not too much of a break. Sit back and listen to the history of the relational database. Find out the crucial moves that Oracle made at critical junctures of its history. See what drove the product from inception, over the rocky road, and eventually to the top of the mountain. Learn what made Oracle, the product, a success, but also find out the attributes that made Oracle, the company, a font of technological wizardry. This talk will reveal several seldom-heard facts and some unknown secrets of Oracle's success.

*Cary Millsap is the founder and president of Method R Corporation (***www.method-r.com***), a company devoted to genuinely satisfying software performance. He is widely known in the Oracle community as a speaker, educator, consultant, and writer. He is the author (with Jeff Holt) of* Optimizing Oracle Performance, *for which he and Jeff were named* Oracle Magazine*'s 2004 Authors of the Year. He is also a contributor to* Oracle Insights: Tales of the Oak Table. *Cary is the former vice president of Oracle's System Performance Group, and a co-founder of Hotsos. Cary is also an Oracle ACE Director and a founding partner of the Oak Table Network, an informal association of Oracle scientists that are well known throughout the Oracle community.*

—Room 1220—

Case Studies in Performance Problem Diagnosis and Repair

Cary Millsap, *Method-R.com* 11:00–12:00

While we certainly learn from our own experiences, we can save ourselves plenty of time and frustration if we can learn from the experiences of others. This presentation reviews the real-life experiences of professionals like you who faced some nasty performance problems and lived to tell the tale. Come experience their journeys as a spectator so that the next time you're faced with a similar problem, you may have just the information you need to handle it with greater ease and efficiency.

Getting Coherence: Introduction to Data Grids

Raanan Dagan, *Oracle*. 1:00–2:00

Grid-based infrastructures are being developed, deployed, and used to achieve unlimited application scalability and continuous availability across multiple datacenters. Understanding the additional capabilities of these infrastructures—and how they can be improved with the use of data grid technology to solve increasingly difficult and complex problems—ensures that your organization is getting the maximum utility from grid computing.

This session focuses on how Oracle Coherence Data Grid can easily help you achieve all of these goals and more!

Power at Your Fingertips: Overlooked Gems in Oracle Enterprise Manager

John Sheaffer, *Oracle Corporation* 2:30–3:30

Today's DBA is increasingly tasked with providing greater support across the enterprise—in some cases, being stretched so thin that leveraging automated tools is your only survival mechanism. In this session, you will learn about how often-overlooked features of Oracle Enterprise Manager will help you improve efficiency, lower your blood pressure, and get you home on time!

A Tour of the AWR Tables

Dave Abercrombie, *Convio*. 4:00–5:00

Introduced in version 10*g*, Oracle's Automatic Workload Repository (AWR) provides diagnostic information for performance and scalability studies, automatically recording a rich variety of database performance statistics.

What's the best way to leverage this wealth of data? While you can run Oracle-supplied AWR reports or use Oracle features such as the Automatic Database Diagnostic Monitor (ADDM), each Oracle database presents its own unique tuning challenges. In this session you'll learn how to work directly with AWR tables, using customized queries to improve insight into your own particular scalability issues.

Topics include:

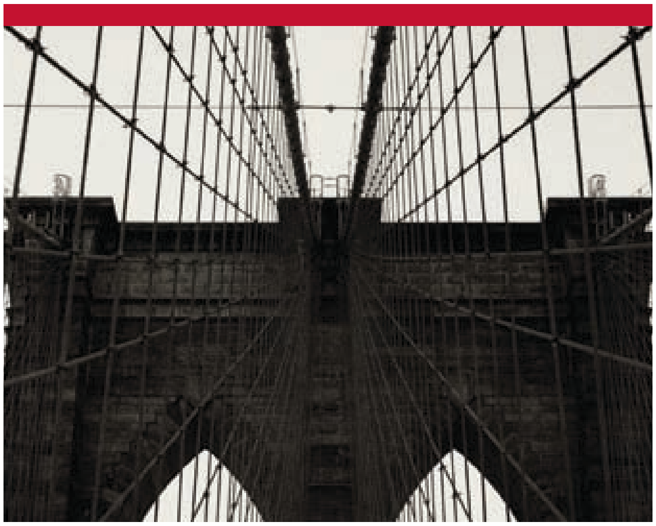
- Important AWR tables, their contents, how to join them, and their quirks and limitations.
- Sample queries that can be easily adapted to focus on your own unique set of problems.
- Estimating the "Average Active Session" metric.
- Simple statistical techniques to find spikes and other types of anomalous behavior.
- A comparison of techniques used for historical scalability studies with those used for real-time performance crisis resolution.
- Use of DBMS_APPLICATION_INFO and JDBC end-to-end metrics.
- Useful tips on configuring AWR.

—Room 1240—

Aces in the Hole: Learning Advanced SQL Techniques from the OTN Forum Pros

Greg Pike, *Piocon*. 11:00–12:00

Although seasoned professionals understand the benefits of solving business problems with efficient queries or PL/SQL, the volunteer experts and Oracle Ace contributors on OTN's SQL and PL/SQL discussion forums raise the query-



We can't change the past, but...
we could have predicted it!

writing bar to an entirely new level. Oracle professionals at any skill level will find this forum packed with a treasure chest of tips, tricks, and techniques. With over 60,000 topics and 325,000 posts, it's the mother lode of SQL and PL/SQL education from recognized experts worldwide.

In this session, the powerful techniques of advanced query authoring are explored by reviewing real-world forum threads and the unique solutions posted by the gurus. The resident experts from the OTN forums solve problems using a combination of analytic functions, hierarchical queries (CONNECT BY), collections (COLLECT), XML functions/operators (SYS_XMLGEN, and XMLSEQUENCE), Pipelined Functions, the MODEL clause, and more.

Data Warehousing with Oracle 11*g*

George Lumpkin, *Oracle Corporation* 1:00–2:00

Satisfying business intelligence requirements for all users throughout the enterprise requires a fast, reliable, and scalable data warehouse to protect and maintain quality business information. Come learn why Oracle is the #1 database for data warehousing, why Oracle Warehouse Builder is the best tool for building data warehouses, how ground-breaking new features in Oracle Database 11*g* will dramatically speed query performance, and how you can accelerate data warehousing deployments.

Introduction to Java: PL/SQL Developers Take Heart

Editor's Pick

Peter Koletzke, *Quovera* 2:30–3:30

Oracle's current focus on implementing database and development features based on the Java language may have you thinking that you need to learn Java. However, if you are familiar with PL/SQL, your first view of Java may be a bit discouraging because its object-oriented core makes it look very different. Also, you may be wondering about Java's strengths and weaknesses and where it fits in the industry.

This presentation explains the basic concepts of, and terms used in, Java to PL/SQL developers who have had little or no exposure to Java. The presentation provides an overview of the language and reviews the concepts of object orientation that Java is based on. It also discusses the fundamental Java code structures—classes and methods—as well as control statements, exception handling, data types, and variables. This explanation will act as a springboard for further study.

Level: *Beginner*

What the Oracle Really Meant: The Quest for PL/SQL Testing Using Code Tester

Arnie Weinstein, *Lawrence Livermore National Laboratory* 4:00–5:00

Software developers face constant pressure to produce highly complex PL/SQL code under tight deadlines. Without an efficient and reliable way to perform thorough code testing, software is released with defects that would otherwise be eliminated. Using an automated test tool carries risks that may be mitigated by certain practices. These practices greatly improve our ability to develop high-quality and efficient testing software. In this paper, we will share some of these practices.

—Room 1130—

Architecting Data Systems for Compliance

John Weathington,
Excellent Management Systems. 11:00–12:00

A gentle introduction to compliance for database system architects, and introductory concepts for the optimal architecture to support the growing need for your company's compliance.

Building a Web-Based Application Using Application Express

Willie Albino, *Lockheed Martin* 1:00–2:00

Application Express is a free web application development environment that comes with the Oracle database (Oracle 9.0.3+). Using a simple web browser, you can create and deploy web-based, database-centric applications very quickly. In addition you can upload and create applications from MS Excel spreadsheets. In this session, an introduction to the Application Express environment will be provided. In addition, an application will be built from tables existing in the database, as well as data uploaded from some Excel spreadsheets.

Oracle Fusion Middleware Roadmap & Strategy

Margaret Lee, *Oracle Corporation* 2:30–3:30

With the recent acquisition of BEA, a number of best-in-class products have been added to Oracle's robust portfolio of middleware solutions. Customers are interested in knowing what Oracle's strategy and roadmap will be for protecting existing BEA customers' investments and integrating BEA products into existing Fusion Middleware solutions. This session will detail how Oracle will leverage best products from both companies to provide solutions in Java & Transaction Processing, SOA and Business Process Management, and User Interaction and Enterprise 2.0.

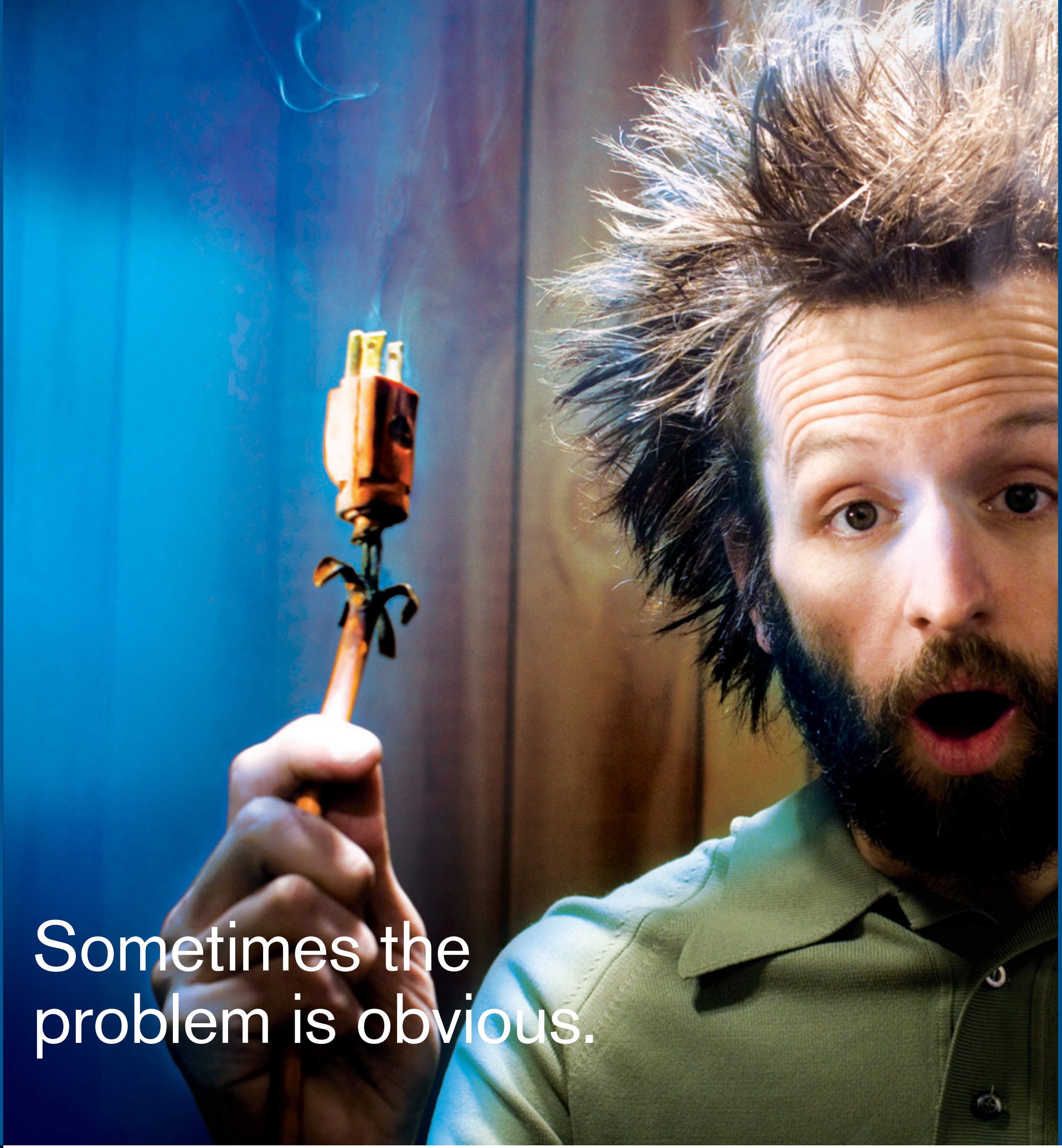
Oracle Analytical Functions: The Hidden Treasure

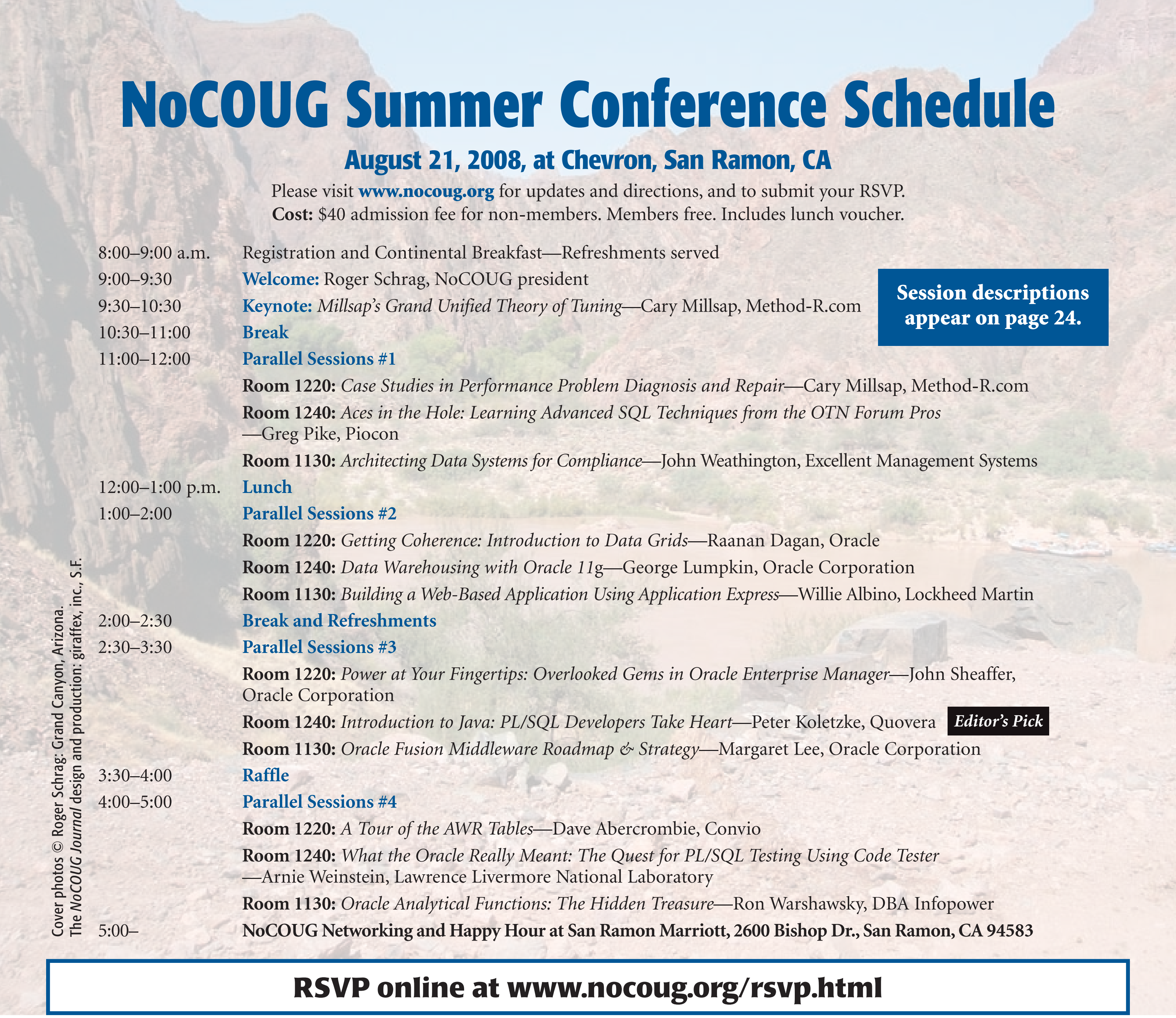
Ron Warshawsky, *DBA Infopower* 4:00–5:00

Oracle introduced them in 8*i*, but do we really use them to their full potential? Oracle analytical functions can save a great deal of coding and provide amazing results very fast.

- How about doing this—and this is only a beginning:
- Ranking data within subsets of a data set.
- Aggregating data within subsets of a data set.
- Performing aggregations over moving windows.
- Displaying and comparing aggregates to individual entries within a single query.
- Comparing two or more rows within a given data set.

Jonathan Lewis Training Days—details on page 13


Sometimes the
problem is obvious.



NoCOUG Summer Conference Schedule

August 21, 2008, at Chevron, San Ramon, CA

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

Session descriptions appear on page 24.

8:00–9:00 a.m.	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Roger Schrag, NoCOUG president
9:30–10:30	**Keynote:** *Millsap's Grand Unified Theory of Tuning*—Cary Millsap, Method-R.com
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Room 1220: *Case Studies in Performance Problem Diagnosis and Repair*—Cary Millsap, Method-R.com
	Room 1240: *Aces in the Hole: Learning Advanced SQL Techniques from the OTN Forum Pros* —Greg Pike, Piocon
	Room 1130: *Architecting Data Systems for Compliance*—John Weathington, Excellent Management Systems
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Room 1220: *Getting Coherence: Introduction to Data Grids*—Raanan Dagan, Oracle
	Room 1240: *Data Warehousing with Oracle 11g*—George Lumpkin, Oracle Corporation
	Room 1130: *Building a Web-Based Application Using Application Express*—Willie Albino, Lockheed Martin
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Room 1220: *Power at Your Fingertips: Overlooked Gems in Oracle Enterprise Manager*—John Sheaffer, Oracle Corporation
	Room 1240: *Introduction to Java: PL/SQL Developers Take Heart*—Peter Koletzke, Quovera ***Editor's Pick***
	Room 1130: *Oracle Fusion Middleware Roadmap & Strategy*—Margaret Lee, Oracle Corporation
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Room 1220: *A Tour of the AWR Tables*—Dave Abercrombie, Convio
	Room 1240: *What the Oracle Really Meant: The Quest for PL/SQL Testing Using Code Tester* —Arnie Weinstein, Lawrence Livermore National Laboratory
	Room 1130: *Oracle Analytical Functions: The Hidden Treasure*—Ron Warshawsky, DBA Infopower
5:00–	**NoCOUG Networking and Happy Hour at San Ramon Marriott, 2600 Bishop Dr., San Ramon, CA 94583**

Cover photos © Roger Schrag: Grand Canyon, Arizona.
The *NoCOUG Journal* design and production: giraffex, inc., S.F.

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

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED

Jonathan Lewis Training Days—details on page 13