Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 24, No. 2 · MAY 2010 $15

Soar with Oracle at NoCOUG

Battle Against Any Guess

An interview with the founder of the BAAG party.

See page 4.

Expert Oracle Practices

"Required reading," concludes our book reviewer.

See page 8.

Battle Against Any Guess

The opening chapter from Expert Oracle Practices.

See page 13.

Much more inside . . .

Battle Against Any Guess

As the editor of the *NoCOUG Journal*, I have had the privilege of interviewing some of the very best minds of the Oracle world such as Craig Shallahamer, Cary Millsap, and Jonathan Lewis—to name just a few. In this issue, I've interviewed Alex Gorbachev, Chief Technology Officer of Pythian and the leader of the "Battle Against Any Guess" (BAAG). In addition to the interview with Alex, this issue of the *Journal* contains a review of *Expert Oracle Practices* co-authored by Alex and other members of the Oak Table Network. Our reviewer pronounces the book required reading for all database administrators and the *Journal* does its part by bringing you the entire text of the chapter by Alex.

But wait, there's more! Alex will also be delivering two talks on Real Application Clusters technology at the NoCOUG spring conference on Thursday, May 20, at the Oracle conference center in Redwood Shores. Join me there to hear Alex speak. ▲

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

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 August 2010 issue is May 31, 2010. Article submissions should be made in Microsoft Word format via email.

2010 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

"I'll tell you what, we're having an earth..."

by Hanan Hit

Hanan Hit

On October 17, 1989, sixty-three thousand fans filed into Candlestick Park to watch Game 3 of the World Series between the two local Bay Area rivals, the San Francisco Giants and the Oakland Athletics. At 17:04, thirty-one minutes before the scheduled start of the game, an earthquake measuring 6.9 on the Richter scale hit northern California, the epicenter about forty miles south of Candlestick Park. Al Michaels had just handed off to color man Tim McCarver, who was describing game highlights when the video signal began to flicker and then break up. Just before the picture and sound were lost, the TV audience heard Michaels announce the news: "We're having an earth…"

Thinking about the 1989 earthquake led me to ponder the technological earthquakes that were introduced by Oracle in the past two years. The first technological earthquake was the introduction of the database machine when Oracle incorporated hardware and software into a single package. The database machine—Exadata—is seeking to define the database architecture of the future. The second technological earthquake was the Sun acquisition which brings the addition of servers, storage, SPARC processors, Solaris operating system, Java, and the MySQL database to Oracle's portfolio. I believe that both will shake the enterprise industry in the next five to ten years.

NoCOUG is a wonderful asset to Oracle DBAs and developers. With *NoCOUG Journal* articles and conference presentations on such topics as the Exadata database machine, RAC workload management, best practices, and Oracle Database 11*g*R2 new features, it is easy to see that NoCOUG offers many ways for Oracle professionals to sharpen their technical skills.

The NoCOUG Spring Conference will be held on May 20 at the Oracle conference center in Redwood Shores. Oracle VP Sushil Kumar will kick off the day with a keynote presentation titled *Business-Driven IT Management With Oracle Enterprise Manager* followed by technical presentations by Oracle employees and other great speakers such as Alex Gorbachev, CTO of Pythian who is speaking at NoCOUG for the first time. 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**. I would also like to invite you to NoCOUG's annual training day which will take place on August 18th at the CarrAmerica conference center in Pleasanton and features performance guru Craig Shallahamer. ▲

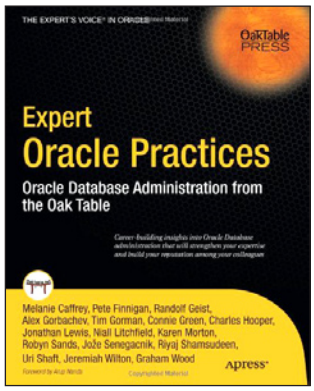
One-Day Performance Seminar with Craig Shallahamer

Wednesday, August 18, 2010—CarrAmerica Conference Center, Pleasanton, CA

OraPub's One-Day 2010 Performance Seminar was created specifically for experienced DBAs and is squarely focused on advanced Oracle performance analysis. To develop the seminar, OraPub surveyed the most experienced students in the firefighting and advanced analysis courses and specifically asked what were the most memorable, valuable, and relevant aspects of the courses.

Craig Shallahamer is planning a day of intensive learning, which includes multiple aspects of Oracle performance tuning. We will learn how to analyze the resources consumed by competing processes and how to avoid hitting resource capacity limits. We will map the different performance problems and the specific techniques to approach each problem type. We will explore the relations between different components that impact performance and how they influence one another. Then we will take all this knowledge and learn how it applies specifically to the problem of free buffer waits, taking a deep dive into Oracle architecture in the process.

If you want to learn how to quickly solve Oracle performance problems, come take advantage of a full day of training by Craig Shallahamer on August 18, at the CarrAmerica Conference Center in Pleasanton for $400. Register by July 10 to receive an early bird price of $350. Contact **training_day@nocoug.org** *or register at* **www.nocoug.org**. *For more details, refer to page 28.*



Battle Against Any Guess

With Alex Gorbachev

Alex Gorbachev

Pythian CTO Alex Gorbachev is a respected figure in the Oracle world and a sought-after leader and speaker at Oracle conferences around the globe. He also regularly publishes articles on the Pythian blog. Alex is a member of the Oak Table Network and holds an Oracle ACE director title from Oracle Corporation. He is the founder of the Battle Against Any Guess (BAAG) movement promoting scientific troubleshooting techniques.

Battle Against Any Guess

Tell us a story. Tell us two. We love stories!

It's June 2007, and I still have enough time left in my day to be active on the Oracle-L list. I'm reading the threads and once again there is one thread full of guesswork-based solutions to solve a particular performance problem. Not the first one and not the last. After entering into the discussion, I felt the conversation was the same I'd had time and time again (like a broken record), and this prompted me to create a place on the internet that I can refer to whenever I and others need to point out the fallacies of guesswork solutions. And so, the BAAG Party was born—**www.BattleAgainstAnyGuess.com**. The name idea came from the BAARF Party (Battle Against Any Raid Five) organized by fellow Oak Table Network members James Morle and Mogens Nørgaard.

What's wrong with making an educated guess? We have limited data, limited knowledge, limited experience, limited tools, and limited time. Can we ever really know?

"Yes we can!" At least, we should strive to know.

I'll never forget how enlightened I was the moment I saw the slide "Why Guess When You Can Know?" presented by Cary Millsap, another fellow member of the Oak Table Network. Most real life problems can be solved with the knowledge that is available in the public domain, using data that is possible to extract by applying the right experience and tools and taking enough time to do the job properly.

It is the purpose of the Battle to promote the importance of knowledge fighting ignorance, selecting the right tools for the job, popularizing the appropriate troubleshooting techniques, gaining experience, and learning to take time to diagnose the issue before applying the solution. One might think that the BAAG motto is a bit extreme but that's a political decision to emphasize the importance of the goal.

I have elaborated on the concept of the "educated guess" in the first chapter of the book *Expert Oracle Practices: Oracle Database Administration from the Oak Table*. The chapter is titled "Battle Against Any Guess." I would like to quote the following from page 11:

Oracle Database is not only a complex product, it's also proprietary software. Oracle Corporation introduced significant instrumentation and provided lots of new documentation in the last decade, but there are still many blanks about how the product works, especially when it comes to the implementation of new features and of some advanced deployments that hit the boundaries of software and hardware. Whether it's because Oracle wants to keep some of its software secrets or because documentation and instrumentation are simply lagging, we always face situations that are somewhat unique and require deeper research into the software internals.

When I established the Battle Against Any Guess Party, a number of people argued that guesswork is the cruel reality with Oracle databases because sometimes we do hit the wall of the unknown. The argument is that at such point, there is nothing else left but to employ guesswork. Several times people have thrown out the refined term "educated guess." However, I would argue that even in these cases, or especially in these cases, we should be applying scientific techniques. Two good techniques are deduction and induction.

When we have general knowledge and apply it to the particular situation, we use deductive reasoning or deductive logic. Deduction is often known as a "top-down" method. It's easy to use when we have no gaps in our understanding. Deduction is often the path we take when we know a lot about the problem domain and can formulate a hypothesis that we can confirm or deny by observation (problem symptoms).

Inductive reasoning is often considered the opposite of deductive reasoning and represents a bottom-up approach. We start with particular observations, then recognize a pattern, and based on that pattern we form a hypothesis and a new general theory.

While these techniques are quite different, we can find ourselves using both at different stages as verification that our conclusions are correct. The more unknowns we face, the more we favor inductive reasoning when we need to come up with the generic theory while explaining a particular problem. However, when we form the theory via inductive logic, we often want to

prove it with additional experiments, and that's when we enter into a deduction exercise.

When taking a deductive approach first, when applying known knowledge and principles, we often uncover some inconsistencies in the results that require us to review existing theories and formulate new hypotheses. This is when research reverts into inductive reasoning path.

Deduction and induction each have their place; they are both tools in your arsenal. The trick is to use the correct tool at the correct time.

How do we decide which competing methodology to use? Which tool is the best tool for the job? In matters of performance tuning, should we trace, sample, or summarize?

Good questions. Logic and common sense come to mind as the universal methodology for any troubleshooting. If we focus on performance then we should define what it means to improve performance. For me, performance tuning is all about reducing the response time of a business activity. When I think performance, I think response time. This is what Cary Millsap taught me through his book *Optimizing Oracle Performance*—he shifted my paradigm of performance tuning back then (by the way, you can read more about the paradigm shift concept in my chapter referenced above).

Since we identified that response time is what matters, the next step is to analyze where the time goes—build the response time profile. Adopting a top-down approach we might find that 2% of the time is spent on the application tier and 98% of the time spent in the database. Drilling down to the next level of granularity, we could identify two SQL statements that consume a 42% response time each. Focusing on those two, we drill down further into, say, wait events. We could pinpoint the reason for excessive response time at this stage or we might need to dig even deeper—somewhere where timed information isn't available. This is where the current battle lies—we could win it by introducing the right instrumentation and tools.

More than a decade ago, Oracle database performance analysts didn't have the luxury of wait interface and had to rely on various aggregations and ratios as time proxies. The same happens now on another level—when wait interface granularity is not enough, we have to rely on counters and methods such as call-stack sampling. Again, the same goes when execution exits the database, for example, to do storage I/O. Current I/O systems are not instrumented to provide a clear response time profile.

However, I want to emphasize that the vast majority of mistakes during performance diagnostic happen much earlier when we have enough knowledge and tools to avoid applying guesswork solutions, but we often don't.

I digressed in my response from the original question on what the best tools are, but, unfortunately, I will have to disappoint—there is no magic-bullet performance tool that will diagnose all problems. The most sound advice I can give is to study the performance methods and tools available, understand how they work, when they should be used, and what their limitations are and why. There are a number of books published and if you ask me to distinguish one of the recent books, I would mention *Troubleshooting Oracle Performance* by Christian Antognini.

Should we extend the scientific method to Oracle recommendations or should we adhere to the party line: use the cost-based optimizer, don't use hints, collect statistics every night, upgrade to Oracle 11g, apply the latest patch set, CPU, and PSU, etc.? After all, nobody gets fired for following vendor recommendations. Many years ago, I lost a major political battle about Optimal Flexible Architecture (OFA) and never recovered my credibility there. Once Bitten, Twice Shy is now my motto.

I've touched on the issue of best practices in the BAAG chapter:

"Best practices" has become an extremely popular concept in recent years, and the way IT best practices are treated these days is very dangerous. Initially, the concept of best practices came around to save time and effort on a project. Best practices represent a way to reuse results from previous, similar engagements. Current application of best practices has changed radically as the term has come into vogue.

What are now called best practices used to be called "rules of thumb," or "industry standards," or "guidelines." They were valuable in the initial phase of a project to provide a reasonable starting point for analysis and design. Unfortunately, modern best practices are often treated as IT law—if your system doesn't comply, you are clearly violating that commonly accepted law.

Vendor recommendations are very valuable in the early stages of a project and even later on, as progress is made. In order to apply vendor recommendations correctly, one should understand the reasoning behind such advice, what problems it solves specifically and what else could possibly be affected. If you take an example of collecting statistics every night, then it makes sense for the majority of Oracle databases. There are plenty of exceptions, however, and at Pythian, we often modify the default collection schedule for our customers. Having a sound understanding of what a vendor recommends and why is the key to a successful implementation.

In some cases, it might be difficult to act contrary to generic vendor recommendations, and convincing management otherwise is usually very difficult. Some basic principles to keeping in mind when deciding your course of action are below:

- Vendor recommendations are generic. Consider them as the default configuration of init.ora parameters. Nobody runs with all default parameters.
- Instead of going against vendor recommendations, call it modifying or adapting to a particular environment.
- Find a precedent where a recommendation has failed and why. It's like being in court—nothing beats a precedent.
- Playing politics is a whole different game. Either you are a player or you stay away.

Pythian

Tell us something about Pythian. Where does the name come from?

Centuries before the Roman Empire, the Pythian Priestess,

also known as the Oracle of Delphi, was widely recognized and respected as the world's most accurate, most prolific, most trusted dispenser of wisdom and prophecy. Specially chosen, carefully trained, deeply insightful, and profoundly wise, Pythian Priestesses were thought to speak with the voice, the vision, and the very soul of Apollo, the Greek God of the sun, medicine, prophecy, and music.

> ***"If I had to make a decision on whether to invest my money in certification versus conferences or user group fees, I would choose the latter."***

During a dynasty that would last some twelve hundred years, the temple at Delphi was the intellectual center of the world. Pythian Priestesses were credited by the world leaders of the era with guiding and inspiring many great triumphs of art, science, justice, commerce, and civilization. They were also credited with the creation of the Pythian Games, occurring every four years during much of those twelve centuries, alternating with the ancient Olympics, but emphasizing music and poetry as well as athletic contests.

This is well aligned to our vision that database administration is not only a science but an art.

The Pythian website says: "We have developed unparalleled skills, mature methodologies, best practices and tools that ensure Pythian clients receive a level of service that can't be found anywhere else." Please do tell us some more.

Pythian has been living and breathing databases for over thirteen years. Having a dedicated global team working for our customers 24/7/365, we serve over 400 clients, and perform countless implementations, migrations, etc. To be able to provide such high quality of services, we go out of our way to hire and retain the world's best DBAs in their respective field. DBA talent is our most important asset—four of Pythian DBAs are Oracle ACEs, and many Pythian DBAs are conference presenters and authors on the Pythian blog read by tens of thousands every month seeking help for their technical challenges.

Does Pythian have any plans to support other database technologies such as MySQL and SQL Server? If so, why? If not, why not?

Pythian has been supporting MySQL and SQL Server for years in addition to Oracle databases as well as Oracle E-Business Suite and Fusion Middleware (what used to be Oracle Applications Server all those years). Many of our customers are running heterogeneous environments and our flexible, utility-based business model lets them easily take advantage of all the expertise that we have accumulated from over 13 years in business. We have separate support teams focused on MySQL and SQL Server and several top-tier cross-platform experts.

Today, I'm really excited about MySQL coming under the Oracle umbrella. I think that MySQL and Oracle technologies fit very well together. Oracle has shown strong support for the MySQL community and it's showing already through the expansion of their Oracle ACE Program to include a MySQL domain speciality. Pythian's Sheeri Cabral has recently been named the very first MySQL Oracle ACE director, for her contributions to the community and MySQL expertise. Very exciting news for all of us as Sheeri is definitely the #1 community leader—she lives and breathes MySQL. I can't stop smiling knowing that Oracle is recognizing MySQL community contributors just like all their other technologies. Hopefully this should lay to reset whines like, "Oracle will kill MySQL."

The Job Scene

Where have all the DBA jobs gone? I tested the waters by posting my resume on the HotJobs job board and received seven spam replies over the course of a week, none of which had anything to do with database administration. I counted the Oracle DBA positions listed nationwide in the past 30 days on HotJobs and found only 56—far fewer than I used to see.

Identifying and hiring the best candidates has always been one of our top priorities at Pythian. In the past year or two, we haven't seen much difference in the number of elite DBAs available for hire so we've just had to work a little harder to grow three times in the last three years.

Our main source of candidates is the community—through word of mouth, conferences, our blog, and social media like Twitter and LinkedIn. Perhaps job boards are not mainstream for DBA recruiting anymore? Based on what we hear and read from industry research published, database administrators continue to be one of the most in-demand IT positions.

What would you say companies are looking for in an entry-level DBA? What kind of knowledge would you say an entry-level DBA should possess before applying for a job? (Question sent in by a student from Atlanta, GA)

When we hire a DBA, we don't look at years of experience in the traditional way and when it comes to the opening of junior DBA, many candidates with years and years of experience don't qualify. We are looking for individuals with the ability to take responsibility, ability to learn quickly, who possess a broad area of interests in IT, who are logical thinkers and analytical personalities. Because you're front line in working with our clients, it's also very important to have excellent communication skills, and that's not very common among "techies," to say the least. A number of DBAs started at Pythian as complete juniors within the strong database teams and evolved into our top-tier consultants doing impressive projects now such as Exadata implementation or architecting a distributed system to handle tens of thousands of transactions per second. On the other hand, there were quite a few that started well into their careers, and who were very experienced, but didn't cut it at Pythian.

Will certification help our careers? Which certifications do you recommend, if any? Certification is very expensive because of the requirement to attend training courses at Oracle University. Is it a good return on investment? Should we just read a few good books instead?

I do have OCP certificates for a number of database releases and even as a developer. I can't recall that being a certified DBA

has helped me in my career directly. However, in preparing for the exams I had to fill some gaps in my knowledge and that definitely helped later on. Oh, and my certificates also look good on my wall in nice black-and-gold frames.

On a serious note, some recruiters and HR departments are still looking for certifications as criteria and so I do realize that it might be a vital requirement for junior-mid-level and sometimes even senior positions. In many cases, not always known to outsiders, some vendors require their partners to employ a number of certified specialists to qualify for a certain partnership level. For example, Pythian has to have a certain number of certified DBAs to achieve the Platinum Level Oracle Partner status that we have today. Such a vendor requirement is a great way to motivate your employer to pay your certification pre-requisite courses.

In hindsight, if I had to make a decision on whether to invest my money in certification versus conferences or user group fees, I would choose the latter. It definitely gives a better career boost.

Few of us are well-rounded. We know a lot about Oracle 9i, Oracle 10g, and Oracle 11g, but not much else. Perhaps that's why our jobs are so replaceable. Do you have any recommendations in this area?

When we hire a DBA, we hire permanently. We are engaged in many short-term projects and consulting engagements but never really hire consultants for specific projects. Besides technology knowledge, it's important to consider lots of other criteria such as the ones I mentioned above. Did I emphasize enough that effective communications skills are crucial to a DBA's role?

In addition, for a DBA to be successful, it's not enough to have a good knowledge of the just the database technology. DBAs are usually required to know and understand the intricacies of a very broad set of technologies including networking, storage, operating systems, clustering, virtualization, data modelling, understanding of development frameworks, etc. Investing in learning all those areas will make you more valuable, boost your employment opportunities, and make you harder to replace.

On the other hand, Oracle database technology has become so broad that it's impossible for a mere mortal to know it all in depth these days. Specialization is another way to differentiate yourself, like digging into Security or Streams replication and other data movement features.

Is Oracle a legacy platform? Should we be worried about MySQL and PostgreSQL?

I think that any technology that has reached very high adoption levels becomes stalled and further evolution is much more difficult compared to market newcomers who can enter and revolutionize an industry. Wide adoption doesn't allow a vendor to perform radical changes to the product that will make it unusable to existing customers. However, I think Oracle has been quite successful so far in evolving its product, unlike some of its competitors, who shall stay nameless.

MySQL is definitely on the rise and is one of our fastest growing service areas. With Oracle taking over MySQL, we expect MySQL only to accelerate. This is why we developed a special MySQL Accelerator program which is specifically designed to kick start your DBA team with MySQL technology, based on your own environments over the course of just few days. It's been ever popular since its introduction at Pythian. You can find out more about the program on our website or by sending an email to **mysql@pythian.com**.

Bonus Round

It seems that every move I make and every step I take is being watched and recorded in a database. Are databases more evil than good?

Information is power. If it's in good hands—it's good power. Information that falls into bad hands is dangerous. That's why it's so important to safeguard your data these days and why information security and privacy is so high on the radar of IT managers.

Thanks for coming out to speak at our next conference; we're looking forward to your presentations on RAC workload management. It costs us about $14,000 per year to produce and distribute the* NoCOUG Journal *and considerably more than that to organize four conferences. We have about 500 members and about 200 attendees at each conference; we've stagnated at those levels for many years. Is Google making us obsolete? I begged my previous company to buy a corporate membership but, except for my manager, not a single person from that company ever showed up at our conferences. Should we close shop? What could we do better to attract more members and conference attendees?

The Internet has provided us with a new communication medium that brings completely new opportunities and communication efficiencies that shouldn't be ignored. In order to adapt to the age of the internet, we all must re-evaluate our investments and optimize their effectiveness.

Because a user group exists solely for the benefit of its members, I believe that members should decide whether the *NoCOUG Journal* is worth the investment. Perhaps it makes sense to ask them to vote whether they would rather see the printed *Journal* (which costs them $28 of their annual fees) or a more informal online journal/newsletter edition that is five times cheaper, and spend the rest of the funds to organize an additional seminar or two for the members.

Having said that, face-to-face communication is very important and provides another dimension in networking possibilities. Online media still doesn't provide all nuances of human interaction, so gathering a big crowd of like-minded people triggers completely new discussions and ideas.

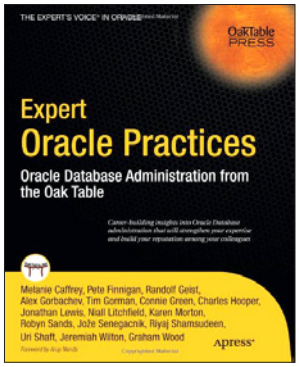
While building Pythian Australia in Sydney, Australia, I founded Sydney Oracle Meetup—an informal club sponsored by Pythian and gathering twice a month in the evening in a semiformal environment where we had a presentation followed by follow up discussions and networking—all that mixed with pizza and drinks. In just a year, we've reached almost 200 members and 20–40 people are showing up for every meeting. Perhaps you could try such informal regular gatherings? ▲

Contact Alex at **gorbachev@pythian.com**, follow him on Twitter **@AlexGorbachev** or read his blog at **www.pythian.com/news/author/alex**.

Expert Oracle Practices

A Book Review by Dave Abercrombie

Details



Author: Sixteen members of the Oak Table Network

ISBN-13: 978-1-4302-2668-0

ISBN-10: 1-4302-2668-4

Pages: 592

Publication Date: January 2010

Edition: 1

Price: $49.99 (list)

Publisher: Apress

Summary

Overall review: Each chapter author is a recognized expert in their field, and each chapter is easy to read, while also infused with practical, real-world experience.

Target audience: DBAs

Would you recommend this to others? Yes.

Who will get the most from this book? Performance experts will benefit the most, since most of the chapters focus on performance, scalability, troubleshooting, and tuning. However DBAs who work with large databases or security, or who are curious about cloud computing, will also benefit.

Is this book platform specific? Mostly 10*g* and 11*g*, and the authors are very careful to highlight any version dependencies.

Why did I obtain this book? I found the first Oak Table book, "Oracle Insights, Tales of the Oak Table" to be inspirational, practical, and fun to read. This is a very rare combination, so I eagerly read this book as soon as it became available. Also, the authors are all very active contributors to the Oracle community, and I was very interested in what they would write in this collaborative forum.

Overall Review

This book covers a wide variety of critically important topics. Each chapter author is a recognized expert in their field, and each chapter is infused with practical, real-world experience. Each chapter includes illustrative and informative examples. The topics cover a very wide range: design, troubleshooting, monitoring, tuning, firefighting, very large databases, running on Windows, cloud computing, and security. Most of the chapters are technical, but plenty of organizational and collaborative advice is also included. Although it is unlikely that your work spans all of these topics, you will certainly find direct and immediate benefit from one or more of the chapters. Reading the other chapters is an excellent and easy way to broaden your outlook and improve your skills.

Chapter 1: Battle Against Any Guess

Alex Gorbachev's non-technical chapter motivates us to avoid guesswork by helping us to recognize its causes, symptoms, and dangers. Of course, we all do some guessing, since we rarely have time for full investigations, and we often lack adequate training, tools, and reliable advice. Alex shows that the first step to avoiding guesswork is to really understand the problem, which requires clear communication. Alex points out that the "root cause" of a problem is not always obvious; we so often face a "chicken-or-egg" situation. In response, Alex reminds us to focus on business requirements, costs, and common sense. Alex points out that "Best Practices" are similar to guesswork. They should only be used with thorough understanding, and without losing sight of actual user problems.

Alex enlivens his chapter with examples, including a cross-dressing husband, bind variable peeking on rainy Monday mornings, misbehaving children, RAC node evictions, and a worn bicycle drive train. For more details on these examples and Alex's perspectives, visit his website **www.BattleAgainstAnyGuess.com**.

Chapter 2: A Partly Cloudy Future

Jeremiah Wilton shows us how to run Oracle on Amazon's *Elastic Compute Cloud* (EC2). Jeremiah clarifies the bewildering variety of "Cloud" computing by briefly comparing its three main categories: *Software* as a Service (e.g., Google's Gmail), *Platform* as a Service (e.g., Salesforce.com's Force.com), and *Infrastructure* as a Service (Amazon's EC2). Jeremiah points out that "the term cloud is a poor term for flexible managed services that can be provisioned on demand. Clouds in the sky come and go as they please . . . Maybe this emerging computing model should be named for something IT professionals more commonly provision incrementally, like beer or pizza."

Jeremiah very clearly illustrates how to set up a Linux OS on Amazon's EC2, and how to attach Elastic Block Storage (EBS) disk volumes. He shows how to use an Amazon Machine Image (AMI) containing a pre-built Oracle instance. To avoid loss of necessary data after an EC2 server shutdown, Jeremiah contrasts three different persistence methods.

Chapter 3: Developing a Performance Methodology

Connie Green, Graham Wood, and Uri Shaft encourage us to develop a robust, systematic, and methodical performance strategy. This strategy should be adopted early during application design and development. Some developers might push back at performance review: one even complained *"you're picking holes in it before we've even built it!"* Connie *et al.* advocate for building in application instrumentation, briefly mentioning DBMS_APPLICATION_INFO.

Connie *et al.* rightly view performance as an important aspect of Quality Assurance, arguing for early performance testing. However, they do not emphasize the need for representative test data. All too often we find queries that work fine on the

development platform but cause problems in production, often due to larger data volume or different distributions. Assuming that this limitation has been addressed, Connie *et al.* advise us to save SQL performance statistics from the development environment for later comparison to production values. Such baselines will expedite production problem solving efforts. They are also fans of Oracle's Automatic Workload Repository (AWR), and suggest saving its history in another database to assist capacity planning and pattern identification. Connie *et al.* provide many tips regarding upgrades, such as backing up optimizer statistics along with SQL performance data. This will help to diagnose whether performance changes were due to changes in workload, the execution plan, or the application.

Connie *et al.* define "reactive tuning" as responding to production performance problems, with its increased pressure for solutions and an increased difficulty in making changes. They suggest a simple, scientific, but easily forgotten strategy: 1) define the problem, 2) formulate a theory, 3) implement and verify the solution, and 4) repeat from Step 2 if your goals are not met. They point out that it can be surprisingly difficult to determine the "root cause" of a problem, giving an example where *"cache buffers chains latch contention increased CPU utilization to 100 percent, which resulted in library cache latch wait times increasing to the point that they were the top wait event. Although the symptom pointed to a library cache problem, it was actually the result of cache buffers chains latch contention, which itself was induced by inefficient SQL."* They also provide several examples where staff communication was the biggest problem!

Connie *et al.* provide a useful introduction to well-known diagnostic tools such as the Automatic Database Diagnostic Monitor (ADDM), Active Session History (ASH), AWR, Statspack, and SQL tracing. They also introduce less commonly known tool such as Oracle's "time model," ASH-estimated "DB time," and V$OSSTAT.

Chapter 4: The DBA as Designer

Melanie Caffrey continues the theme of DBAs getting more involved in early application design and development. Running through her specific and practical advice is the basic idea to "*be approachable.*" It can help to simply sit near each other! Melanie advises to be open to new ideas: if you show *"respectful openness to the developers during a design session, you are more likely to garner respect for your ideas in return."* She reminds us of something that is easy to forget: be on the same side. Both DBAs and developers are responsible for what goes into production and for making the customer happy, so an adversarial relationship is counterproductive. A DBA should *"learn what the developers have to go through, and encourage them to learn what you go through, all in the interest of, and with the end goal of, being on the same side."* Her other tips include periodic sign-offs, attending code reviews, postmortems, and brown bag sessions. She does not mince words when she insists that *"becoming your company's Oracle database evangelist should be one of your main goals."*

Melanie provides a wonderful comparison of *"design-first"* methodologies (*"Big Design Up Front - BDUF"*) and *"Agile"* techniques. She recognizes that neither method is usually followed completely, and prefers to discuss a continuum between *"adaptive"* to *"predictive."* Adaptive methods *"focus on adapting quickly to changing realities,"* while predictive methods *"focus on planning the future in as much detail as possible."* Melanie points out that the choice has more to do with how the team is *"best able to work, how exactly you have been indoctrinated in either methodology, and last, but not least, your own personal experience in designing applications."*

Melanie's experience is that schema design is better under BDUF methodologies, where there is more time and ability to take a more global view. She provides many useful and technical schema design examples, including datatype choice, varchar2 length (fetching 100 rows of 20 columns, each of 4000 characters requires 8 megabytes!), SQL comparisons, clusters, fat indexes, global temporary tables, index organized tables, etc. Melanie devotes several pages to the overwhelming advantage of implementing integrity constraints in the database rather than application.

Chapter 5: Running Oracle on Windows

Niall Litchfield immediately addresses the question "Why Windows?" when *"surely all serious databases run on some proprietary variety of Unix or . . . Linux."* His answer refers to its history: Oracle 7 was *"the first relational database available on the Windows platform . . . predating the Microsoft-developed SQL Server 6.0 offering by two years."* Oracle also released *"the first 64-bit database on Windows, the first with support for .Net stored procedures, and so on."*

We are all accustomed to thinking of a session's "server process" as a separate process in Unix, but Windows *uses "operating system threads running in the context of the* `oracle` *executable."* The `V$PROCESS` dynamic performance view still exists in Windows, but its mechanism is entirely different. Oracle supplies an *"Oracle Administration Assistant"* for viewing and killing threads in Windows, but its features are very limited. Niall suggests that instead we use the Process Explorer tool from **www.sysinternals.com**, and provides a detailed example of its use.

Niall walks us through two approaches to overcome 32-bit Windows' default limit of 1.7 gigabytes of addressable memory for any user process. The first uses a boot.ini parameter to allow Oracle to address 2.7 gigabytes. The second approach, known as *"Physical Address Extension (PAE), . . . is a similar approach to the extended memory architecture introduced with the Intel 286 processors."* This sounds complicated, and it is; in fact, regarding running 32-bit Oracle on 32-bit Windows, Niall simply says, *"Don't do it."*

Niall explains in detail how to deal with the many other nuances of running Oracle on Windows, including asynchronous I/O, direct I/O, the Registry, Services, Database Control, Alert Log monitoring, and scripting.

Chapter 6: Managing SQL Performance

Karen Morton advocates for adopting a *"performance mindset"* and focuses on response time. She explains that *"as far as users are concerned, performance is response time."* She points out that *"regardless of whether or not all your monitoring gadgets flash green lights of perfection, if your users are complaining, you've got a problem."* As an aside, in my experience, although

focusing on response time sounds straightforward, in some environments, such as a public website, user response time data may not be readily available, and may have a great deal of variation, all of which complicates the picture. Also, response time can be hard to define for bulk data processing applications where the traditional performance focus is on *throughput* rather than response time: e.g., "how fast can I send one million emails," rather than "how long does it take to display the bulk emailing status report page?"

Karen emphasizes "*managing*" performance and points out "*that if you want something to be manageable, you must give it skilled attention and effort.*" However, it takes more than this, she writes that your mindset, attitude, and your view of your role within the organization can make all the difference.

Karen describes ways of defining and measuring performance. She gives an example where the results of the `EXPLAIN PLAN` statement do not match the actual execution plan. She suggests using the `DBMS_XPLAN.DISPLAY_CURSOR` function instead, along with the `GATHER_PLAN_STATISTICS` hint for gathering rowsource-level details. She also walks us through an Extended SQL tracing session. Karen also advocates taking before and after snapshots of `V$SESSTAT` to include details not shown by the previous methods, such as the `buffer is pinned count` statistic that can shed light on latching overhead.

Karen provides four simple case studies: a suboptimal index, data skew that throws off optimizer estimates, using an analytic function to avoid a correlated subquery, and using tracing to measure costs for an outer query column definition that uses a PL/SQL function that does another query, fixed through use of analytic functions.

Chapter 7: PL/SQL and the CBO

Jože Senegacnik shows us how to help Oracle's Cost-Based Optimizer (CBO) prepare optimal execution plans when using custom PL/SQL function in the `WHERE` clause of a SQL statement. Jože begins with a wonderful walk through the optimizer's process of execution plan preparation, with detail regarding selectivity, cardinality, and cost.

These factors are vital to the CBO, but by default, the CBO cannot estimate selectivity for user-defined function. Jože points out that the *extensible optimizer* feature was added "*to allow users to create statistics collections, selectivity, and cost functions for their user-defined functions and indexes. The CBO can then apply that data to the problem of estimating query cost in order to generate execution plans that perform well.*"

Jože walks us through detailed examples of defining selectivity and cost through implementation of a statistical type. He explains results of event 10053 (optimizer) traces, and goes into detail regarding optimizer calculations, including formulas determined by Wolfgang Breitling, another member of Oak Table.net. His three examples are clear and informative, and additional details are available from the publisher's website (below).

Chapter 8: Understanding Performance Optimization Methods

Charles Hooper and Randolf Geist manage to pack a near-encyclopedic overview of the incredible wealth of diagnostic data provided by Oracle into only 117 pages. They point out that "*the vast list of performance-related information—including system-level and session-level statistics, system-level and session-level wait events, various performance views including `V$LOCK` and `V$SYS_TIME_MODEL`, Statspack, and Automatic Workload Repository (AWR) reports, `EXPLAIN PLAN` and `DBMS_XPLAN`, 10046 traces with and without tkprof analysis, 10053 cost-based optimizer traces, block-level dumps and systemstate dumps, and much more—potentially creates situations in which the database administrator is overwhelmed by all of the options when trying to select a suitable starting point for performance analysis.*" They point out that you first must understand the available options, and then you must carefully choose the correct option for any given problem. This chapter, the largest in this book, covers the *understanding* part, while the following chapter helps with the *choosing* part.

Charles and Randolf begin by echoing themes of Alex Gorbachev's "Battle Against Any Guess" (BAAG, Chapter 1 above). Their first example is of a DBA who used the "Blindly Changing Parameters" approach to increase `PGA_AGGREGATE_TARGET` in response to a problem with "`enq: TX - row lock contention`." The results were, not surprisingly, disastrous. They go through a detailed example showing the uselessness of the Buffer Cache Hit Ratio (BCHR).

After dispensing with the above tragicomedy, Charles and Randolf describe the useful technique of monitoring delta values of system and session statistics, derived from views like `V$SYSSTAT`, `V$SESS_IO`, `V$SESSTAT`, `V$SYS_TIME_MODEL`, and `V$SESS_TIME_MODEL`. As useful as these data are, they point out that "*statistics alone likely do not provide enough detail for a root cause analysis.*" Charles and Randolf discuss the very important topic of sampling these performance views, with low measurement overhead. They write, "*Capturing delta values for a short duration, and not values since the database was last opened, is a requirement when attempting to understand current problems. To capture this data with a minimal impact on system performance, temporary holding tables must be created or the performance data must be exported to another application, such as Microsoft Excel.*"

They also provide a wonderful overview of views like `V$SQLAREA`, `V$SQL`, `V$SQLSTATS`, `V$SESSION_LONGOPS`, and `V$SQL_MONITOR`, that can be used for monitoring the delta values for SQL statements. This is supplemented with advice on how to use views like `V$SQL_PLAN`, `V$SQL_SHARED_CURSOR`, `V$SQL_PLAN_STATISTICS` for examining execution plans and plan statistics. They also summarize the use and benefits of `DBMS_XPLAN.DISPLAY_CURSOR` to simplify the process of obtaining execution plans.

Charles and Randolf give an excellent and detailed overview of the use of event 10053 Cost-Based Optimizer (CBO) traces. Besides detailed instruction on activating and deactivating the optimizer trace, they examine each section of the trace: query blocks, peeked bind variables, optimizer parameters, transformations, system statistics, dynamic sampling, single table access path, general plans, plan table, query block registry, hints, and of course, the query text itself.

Charles and Randolf also explain 10046 extended SQL traces.

They start with a wonderful listing of additional information and resources, must reading even if you are familiar with these traces. Their brief summary of a raw 10046 extended trace file's contents is a very useful reference. Of course, they clearly describe how to enable and disable these traces, and walk us through an example sample trace file analysis. They also describe how to generating a trace file on error, and how to use `ORADEBUG`, `HANGANALYZE`, `HEAPDUMP`, and `SYSTEMSTATE`. They advise extreme caution when thinking about use of operating-system-generated stack traces such as DTrace.

Given their ambitious and encyclopedic scope, you will not be surprised to learn that Charles and Randolf also discuss ADDM, `tcpdump`, Wireshark, and Ethereal, as well as client-side traces such as SQL*Net tracing, Windows "Process Monitor," and Windows Spy++. They close this chapter with tips on investigating enqueue waits, and summarize the over forty scripts they make available.

Chapter 9: Choosing a Performance Optimization Method

Randolf Geist and Charles Hooper follow up their previous chapter that describes monitoring methods with this chapter that shows which methods are most appropriate for specific situations. To find performance problems that have not yet reported, they suggest reviewing the deltas gathered by the general "Sampling Performance with Low Overhead" technique of the previous chapter, supplemented as necessary by Statspack, AWR, and ADDM, or the Enterprise Manager's performance metrics. For problems reported by end users, Randolf and Charles distinguish between problems clearly tied to a specific single user or job function, and those that are not. For the former, they recommend starting with event 10046 Extended SQL trace file analysis. Otherwise, you must take a more global view, and look at Statspack, AWR, and ADDM, etc. For problems reported by IT Staff, Randolf and Charles suggest starting at the server level with operating system utilities, such as `sar`, `top`, `vmstat`, `iostat`, `netstat`, etc.

Chapter 10: Managing the Very Large Database

Tim Gorman starts by pointing out that the technical constraints for a Very Large Database (VLDB) are not immediately obvious. CPU and memory can be scaled in Windows and Linux by moving to Oracle Real Application Clusters (RAC). On proprietary Unix platforms, individual servers *"can scale to hundreds of CPU and/or cores, as well as hundreds or thousands of gigabytes of RAM."* Tim also says that *"neither is storage a technical constraint. There are storage-area network (SAN) products that offer petabytes of redundant and easily managed storage."* Instead, Tim says that *"the technical constraints consist of . . . limits hard-coded within Oracle, backup media throughput, affecting the speed of backups and restores, and incremental financial costs due to growth."*

Tim focuses first on partitioning, because *"scalability toward infinity starts with partitioning, plain and simple."* He walks us through an interesting example of using a parallel unrecoverable/nologging `CREATE TABLE . . . AS SELECT` operation, using parallel direct-path sessions, combined with an the `ALTER TABLE . . . EXCHANGE` command to update several billion rows in one afternoon.

Tim advocates for information lifecycle management (ILM), where older data is stored on cheaper disks, in read-only tablespaces. He shows that it could take a year to back up a five-petabyte database, at backup throughput rates of 500 to 1000 gigabytes per hour. However, he points out that *"it is only the read-write portion of the database, along with the archived redo log files, that need to be backed up frequently,"* which makes backups feasible.

Tim concludes with a summary of relevant Oracle limits, many of which relate to database block size. Also, the limit of 65,533 data files and tablespaces seems huge, but can be approached by VLDBs. Tim provides much practical advice on dealing with these limits.

Chapter 11: Statistics

Jonathan Lewis points out that even if you have designed good data structures, with perfect indexing, and appropriate SQL, you might still find some SQL statements that fail to run efficiently. He writes that there are two main reasons for this: *"1) the optimizer may literally be unable to produce a certain execution plan that seems to be an obvious choice to the human eye, and 2) the plan you want is technically available, but the statistics used by the optimizer make it look inappropriate."*

Regarding the first issue, Jonathan discusses a subquery anomaly example, where he stripped the problem down to the bare minimum, identified the basic cause, prepared a test case, and found a workaround. The problem was a bug in the way the optimizer applied a default 1% selectivity for a function prematurely in the plan. His next example involves rewriting the SQL to get Oracle to visit partitions in the order that seems obvious to humans.

However, Jonathan primarily focuses on his second issue in this chapter. He begins with a simple question: *"How many people in the world have more than the average number of legs?"* The non-obvious answer is *"almost all of them,"* but you probably guessed a different value rather than calculated. Jonathan then shows where Oracle makes similar guesses: "*So what does Oracle do when presented with a complex query that starts by selecting all the employees earning more than the average salary and then goes on to join seven other tables? Ideally, Oracle should run the first part of the query before trying to work how to run the second part of the query, but the optimizer doesn't work like that at present (even though dynamic sampling is a step in that direction). So what does it do? It "guesses" that the answer is five percent of the employees.*"

Jonathan gives several more interesting examples, including ASSM and clustering factor, partitioning, timing of statistics gathering, histograms, and setting statistics like high values and clustering factors.

Chapter 12: Troubleshooting Latch Contention

Riyaj Shamsudeen introduces his chapter by writing that *"latches are used to serialize changes on critical structures. While latches are crucial to maintain consistency, they also introduce contention points. Resolving these contention points will improve application scalability. Latch contention is a complex topic. This chapter explores the concepts behind latches and latch contention issues, and it offers a few methods to debug and resolve latch conten-*

tion. Some of the most commonly occurring hot-spots of contention for latches are discussed in detail, including cache buffer chains, the shared pool, the library cache, and enqueue hash chains."

Riyaj delves deeply into latch details, describing solitaire, parent, and child latches. He explains how they are managed by the OS using test-and-set instructions or compare-and-swap (CAS) instructions. He compares immediate mode, willing-to-wait mode, and latch-wait posting mode.

With this foundation, Riyaj describes how to identify and analyze latch contention using a well-defined three-step process. He devotes twelve pages to explaining, diagnosis, and fixing cache buffers chains latch contention. This is followed by ten pages on shared pool latch contention, and five pages on library cache latch contention (e.g., "*If your application is collecting histograms on all columns, then rethink that practice, and narrow the scope of collection to only those columns that matter*"). This is followed by five pages on enqueue hash chains latch contention. For each of these latch types, Riyaj goes clearly explains their common causes, their analysis and diagnosis, and problem resolution.

He concludes with advanced help for latch contention problems, including the `V$LATCH_PARENT` view, the parameter, and parameters such as `spin_count`, `_latch_classes`,`_latch_class_n`,`_latch_wait_posting`, and `_enable_reliable_latch_waits`. He rightly insists that you should contact Oracle Support before working with any of these parameters.

Chapter 13: Measuring for Robust Performance

Robyn Anderson Sands begins by discussing response time, throughput, and predictability of Oracle processes. She emphasizes that "*each of these factors can present a system optimization challenge, but the more inconsistency there is within the processes themselves, the more complex it becomes to find and fix the suboptimal components.*" Robyn uses real-world data warehouse tuning examples to illustrate how looking for variability can point to processes that most need attention.

A common measure of variability is the variance of a set of measurements, and it is often divided by the mean to give the variance-to-mean ratio. Robyn writes, "*The variance-to-mean ratio (VMR) is a measure of the randomness of the individual data points within the set. When this calculation was applied to the job execution data, it resulted in a sort of predictability factor: the less dispersion within a process's execution times, the more consistently a job performed. The results were logically consistent with job performance history; jobs with the highest VMR were some of the biggest troublemakers in the job schedules.*"

Robyn goes into detail explaining the mathematics behind this practical observation, referring to original work by Genichi Taguchi on quality engineering methods. However, most readers of this book do not need to worry about these mathematical details. For such readers, Robyn simply shows how to instrument your code to gather timing data, and she shows how to use VMR to find inconsistent, problematic parts of your code.

Chapter 14: User Security

Pete Finnigan starts by reminding us that "*protecting access to database user accounts is one of the most important aspects of securing data in an Oracle database*" and that "*there are still databases with all types of users having excessive privileges or having weak passwords, but there are also more subtle user security problems that must be understood and corrected. Security issues may arise, for example, when real people (users) share database accounts. Problems may also occur when database accounts are shared across multiple applications or administrative tasks or purposes. A final example is accounts in the database that are actually unused or unnecessary.*"

Pete walks us through the process of securing user accounts, which "*should be one of the first steps taken in securing a database.*" These steps include user enumeration, feature analysis, reduction of accounts, assessing account password strength, a roles and privilege assessment, password management, and audit settings. Pete's chapter is a real eye-opener!

Chapter 15: Securing Data

Pete Finnigan follows up on the previous chapter by telling us that "*attempting to secure an Oracle database by using just a checklist is a flawed idea. I say this even after creating some of the checklists that exist.*" In his view, while "*it is impossible to secure the Oracle software, it is possible to secure the data in your database. To create an effective security plan for your Oracle database, you have to start with the data. Unless you know exactly which data you want to secure, there is no way you can secure that chosen data. The idea of focusing on the data first is common sense, but the underlying story of security is common sense. The focus of this chapter is the data. To enable you to secure your database, you have to start with the data; to start with the data, you have to know which data to target.*"

Pete clearly describes a methodology to identify key data and then investigate how the data you have chosen is protected and controlled from a security perspective. As with his previous chapter, he provides a wealth of practical experience, with references to additional resources.

Conclusion

A book like this, written independently by sixteen authors, with each chapter addressing a different topic, has a real potential to be scattered and incoherent. This book does not fall prey to that danger. Instead, this book has a very coherent and cohesive flavor, probably due to the shared vision of the Oak Table Network. What seems to bind these authors together is a scientific spirit, grounded in empirical, verifiable, real-world experience. Also, these authors are all very active contributors to the Oracle community, so they know how to collaborate, explain, and educate. Moreover, they do this with a sense of fun and adventure! This spirit of joyful sharing, combined with practical expert advice, sets this book apart. It is required reading for all DBAs. ▲

Dave Abercrombie works for Convio (with a "v," not an "f") for about ten years. Each day, Convio sends tens of millions of emails to members of nonprofit membership organizations, and then tracks the ensuing member transactions and activities, such as donations, advocacy, and click-throughs. Dave has honed his troubleshooting and scalability skills by keeping these very busy databases happy. He can be reached at **dabercrombie@convio.com**.

Battle Against Any Guess

An excerpt from *Expert Oracle Practices* by Alex Gorbachev et al.

This chapter is an excerpt from the book Expert Oracle Practices: Oracle Database Administration from the Oak Table *published by Apress, Jan. 2010, ISBN 1430226684; Copyright 2010. For a complete table of contents, please visit the publisher site:* **www.apress.com/book/view/9781430226680**.

During my experience with Oracle, I have become very engaged in the user community. I've been a frequent visitor on the Oracle Technology Network forums and the Oracle-L list and have become a regular participant and contributor at user group conferences and other events. My experience started with seeking help and gradually shifted towards helping others with their issues. My growth in Oracle has correlated with the booming popularity of the Internet, over which it becomes very easy to both seek and give advice.

While the Internet increases community participation, it also causes some dysfunction that can lower the quality of the information. Many times I have seen online discussions branch into controversial arguments in which the "combatants" are going by guesswork. It is surprising how few people will stop to test what really happens, and instead will battle endlessly over what might happen or what they believe ought to happen.

While my contributions to the community have been usually rather technical, this chapter is more generic and rather motivational reading. My first attempt at a motivational contribution was creating **BattleAgainstAnyGuess.com**, or the BAAG Party for short, in June 2007. This is where the title of the chapter comes from. The trigger to establish the BAAG Party was coming across yet another quest for guess-based solutions on the Internet; and I wanted something generic to refer to every time I see such symptoms. Thus, I want to start this chapter by showing some examples of guess-provoking questions.

Guess Hunting

The way you ask a question is crucial. A badly formed inquiry is almost guaranteed to attract guess-based solutions. Here is one example of seeking a quick solution from the Oracle-L list:

"I'm also having performance issues with 10g. Why would my dictionary queries take a long time to return? . . . In 9i they used to take seconds, now they take minutes or they just never come back . . ."

When reading this question, it is difficult to divine precisely what the problem is that the writer is experiencing. Without a clear understanding of the problem, the proposed solutions were all over the map. Here are some of the suggestions that turned up quickly:

"You might need to analyze the system tables."

"There are a few known bugs with DD queries in 10g. Few of them involved the CDEF$ table so you might want to do a search for that and/or for the particular views you're having trouble with. The solution was to delete statistics from the involved tables and then lock the stats."

"Remove any initialization parameters set for Oracle 9i."

"Apply application vendor suggestions (like _optimizer_cost_based_transformation=false, NLS_LENGTH_SEMANTICS=CHAR, _gby_hash_aggregation_enabled=false)."

"Disable sub-query unnesting (_UNNEST_SUBQUERY = FALSE)."

"Don't use FIRST_ROWS optimizer goal."

All these might be absolutely valid solutions for different people's own problems. One could very well be the solution the original poster needs, but we don't know which one. A couple of these solutions actually contradict each other (collect vs. delete statistics). These recommendations are based on the previous experience (often quite extensive) of those who proposed them, and they might match well the symptom observed, that "dictionary queries take a long time to return." However, there is one common problem to all of the proposed solutions: the analysis phase is missing. No one has done any analysis or testing to verify the problem, or to verify that their proposed solution even addresses the problem. Everyone is, in essence, guessing.

To show you the magnitude of guesswork and where it leads, here is another example, this time from the OTN Forums:

"My database running on AIX 5.3, oracle getting the version 9.2.0.5.0, after migration it is getting very slow. Kindly provide a solution to tune the database and increase performance."

This is probably an extreme example of ignorance and/or laziness that pushes the author to search for a quick fix solution. Now let's see how this plea for help is being followed up. First of all, a number of people asked for clarification on what is actually running slowly and for more details about the environment—fair enough. However, there was also a shower of well-meaning advice.

One well-intended bit of advice was:

"You can delete and re-gather dbms stats for your application schemas after the upgrade."

And these were the results after the original poster tried implementing the advice:

"getting same problem

continuously database have lock and the dbcache hit ratio is 60% only.

total sga size is 20GB

db_cache_size 13gb"

This next proposed solution is a bit better. It's hinting towards actually analyzing the problem:

"*From OEM you can view the performance and the SQL statements which are being fired every moment and then find out about missing indexes or tune the SQL.*"

Then follows advice from someone who is seemingly a guru in Oracle database performance tuning. That advice comes in the form of 11 bullet points. Applied, each of them could fix certain problems, or make performance worse in this particular case. Making 11 changes and hoping that one of them fixes a problem is not an optimal approach.

Following is yet another suggestion on configuring asynchronous input/output. It could be a valid path in certain cases, but is it valid in this case?

"*Have you got asynch I/O configured on AIX?*"

The original poster did enable ASYNC I/O and, of course, it didn't help.

The list of randomly proposed solutions went on and on, and the discussion ended up sidetracked far from solving the original problem. Did the original poster ever get the help s/he was after? I don't know. It didn't appear so.

Why Do We Guess?

The most trivial factor leading to guess-based solutions is laziness, a natural human quality. Why embark on a path of investigation when there is a chance that one of the proposed solutions just fixes the problem? The reason is that while a random fix might actually work for some commonplace issues, it introduces a significant chance of fixing the wrong problem, making things worse, or simply hiding the symptoms. Fortunately, in my professional life I've met very few DBAs who are too lazy to analyze a problem. Most prefer to troubleshoot and fix a problem once and for all.

Unfortunately, there are other factors besides laziness. Often, companies are not investing enough to provide their support engineers with the right tools and knowledge. Spending money on a performance tuning tool or on the Diagnostic Pack option will pay back in spades when it comes to, say, troubleshooting an issue that causes online orders to time out, or that causes the factory floor to be idle. The same goes with investing in education and hiring qualified performance consultants or services. The investment pays back the next time a problem is experienced with a business-critical function, and such problems are solved many times more quickly than otherwise.

Another common factor is the time pressure. A very common attitude is this: "we don't have time to analyze; we just need to fix it now." That attitude is a siren song. The reality is that the guess-and-try path is much longer on average than a targeted scientific analysis of a problem situation. It often takes a number of guess-and-try iterations until you find the one that works, and there is still a danger of not fixing the root cause—simply because the root cause wasn't identified in the first place.

Yet another reason for premature decisions is the "call for action from above." A loud call for action from management is a variation of the time-pressure problem. Management pressure can be very intense, and it's usually more the effect of the political atmosphere in a company than of anything else. A prime example is when a DBA is pressured to DO SOMETHING NOW to fix a problem and leave the understanding part for later. Such enormous pressure comes from the business because, for example, it might cost millions of dollars for each hour of a factory floor being idle. In such cases, it is reasonable to take measures to reduce business impact to acceptable levels while also undertaking a full investigation into the root problem. Too much management pressure often leads to what I term "blame-storm" meetings, in which fingers are pointed and everyone tries their best to deflect blame. A factor that's often most visible during blame-storm sessions is unwillingness to admit that one does not know where a problem really lies, that one does not really know the root cause. Nobody wants to look stupid, and not knowing what we are supposed to know as professionals tends to make us feel that way. Management often takes advantage of that feeling to add yet more pressure, which ends up as more fuel for the fire.

The best way to address the management pressure is to show that you *know* what to do, and that you have a clear action plan in place that is being followed and that will lead to clearly identified deliverables. It does require some good communication and persuasion skills to calm management, and to present a plan in a way that management will buy in, but the effort does pay off.

A very efficient "play" is to buy time—let someone implement a harmless guess-based solution (providing you are not to be blamed for its failure) while you are focused on the real troubleshooting. In the meantime, focus all your energy on proper analysis.

If you've caught yourself smiling reading these lines, you know what I'm talking about. For the rest of you, be very careful with the strategy of buying time. It might backfire—I warned you!

Understanding a Problem

When faced with a problem, the number-one thing we need to understand is the problem itself. Understanding a problem is the very first step toward a successful solution. Nothing is worse than spinning wheels solving a completely wrong problem. That is why we want business owners and end users to provide some input into our discussion about a problem, even though that input brings with it some pressure to solve the issue.

A great example of what I'm talking about was brought to my attention by Niall Litchfield, a coauthor of this book and fellow member of the Oak Table Network. The quote below is published on **BattleAgainstAnyGuess.com**:

"*One of the great problems with guesswork is that you can be led down some very blind alleys indeed. This commonly occurs when you guess what a problem is, rather than diagnose it, and then embark upon a solution. Every time you find yourself doing this, think of Miriam.*"

Miriam in this case refers to a woman who wrote in to an

advice columnist. She had left for work. Her car had stalled. She walked back home to find her husband wearing her underwear and makeup. Completely put out by that turn of events, she wrote to an advice columnist. And the answer came back:

A car stalling after being driven a short distance can be caused by . . .

Clearly the columnist failed to read far enough to understand the true problem at hand.

When you are defining a problem for others to solve, think whether it's explained clearly, and take care of any possible dual meaning in your words. Even the most obvious explanation can be misunderstood. Be direct. Keep things simple. Speak in short sentences. Try to distill your initial problem statement to one succinct sentence. All these things will help.

But there is responsibility on the other side as well! When you start to solve a problem, don't immediately assume that you understand it. Always double-check what the real issue is. Agree on some tests, or examples to demonstrate the problem, that you can later use to prove the problem fixed. There is no point, for example, in fixing a heavy batch process when the real problem is that interactive users are suffering from timeouts on their order-entry screens.

Logical Conclusions vs. Historical Observations

Munich, May 2006. Tom Kyte, a fellow member of Oak Table Network, was doing a two-day seminar. One of his topics was about bind variables. He shared a very interesting example he observed at one Oracle customer site that I am borrowing with his permission.

According to the end-users' observations, if it was raining heavily Monday morning, then database performance was terrible. Any other day of the week, or on Mondays without rain, there were no problems. Talking to the DBA responsible for the system, Tom found that the performance problems continued until the DBA restarted the database, at which point performance went back to normal. That was the workaround: Rainy Mondays = Reboots.

Any analyst approaching a new performance problem should always gather users' feedback to determine the scope of the issue and the impact on the business. However, one should be very careful in making conclusions from those discoveries. In this case, it was very easy to get sidetracked into environmental causes such as humidity, water levels, and so forth.

What a savvy analyst would learn from these observations is the exact pattern of occurrences in the past, as well as how to reliably catch the issue on the live system in the future. He would also learn that the issue is intermittent, and there is likely some correlation with Mondays' rains.

It turned out that one of the application users was typically starting the day very early, before 7 AM, and the first thing she did was to log into the system and open her main screen—the very screen that she and all her colleagues would be using all the time during the day. Let's call this user Mrs. Early Bird.

Unfortunately, if it's raining heavily on Monday, she must take her children to school, and then fight traffic, causing her to be later to work than otherwise. When it rained on other days, she had other arrangements.

In the meantime, another department was starting the day at 8 AM, each day preparing a special report for the 9 AM Monday meeting based on some historical data. One component of that report was using exactly the same statement as the screen opened by Mrs. Early Bird and later by tens of her colleagues. That statement was using bind variables, which many consider a "best practice" (ding!) of database development. The only difference between the statement used by the interactive application and that used in the report was in the bind variables' values.

Furthermore, a cold database backup was performed each weekend that involved an instance bounce. Thus, all cached SQL and execution plans were, of course, lost. Now, if you know what bind variable peeking is, you are likely to already be on to the true cause and effect chain here. If Mrs. Early Bird starts her Monday early, she executes the key statement first. When Oracle executes an SQL statement with bind variables the first time, it tries to peek into the values of the real variables in an attempt to do a better estimate of cardinalities based on the assumption that most of the following executions will be of the same nature. The application screen produced an execution plan using a nested-loop join with index range scan that performed very quickly for the set of bind variable values that application users tended to use all day long.

However, the statement executed with values from the historical report performed better with a full table scan and a hash join, because it involves rows from a significant part of the table. It was thus much faster and cheaper to perform a full scan rather than a nested-loop join, and the optimizer recognized that. The problem was that the bind variables values used by the report were unique and were used literally once a week—in preparations for that 9 AM Monday report. Unfortunately, that report, executed first in the week, set the tone for the rest of the users of that environment. On days when Mrs. Early Bird did not come in early, the execution plan for the statement would be set by the report, and all havoc broke loose with performance.

Heavy rains on Monday and performance problems on that particular system led to a perfect example of what is called nonsense correlation. The correlation could lead one to a solution, but only if one took care to avoid assuming a causal relationship.

Knowledge Is Power

Nothing helps in troubleshooting as much as knowledge of a problem domain and the technical products used in the system. For a database engineer, the problem domain means application design, business process, maintenance process (such as upgrade or backup), and so forth. Of course, the product we are talking about is Oracle Database, but often the technical stack includes several other products working together.

Broad and up-to-date knowledge is the key to success for a seasoned database administrator these days. Let's take something other than performance troubleshooting as an example. When troubleshooting a failed Oracle RMAN restore, it is crucial to understand how RMAN works with backup pieces and sets, how SCNs are tracked and why they are needed, how Oracle redo is generated and applied to the database during

restore, and all other bells and whistles such as incremental restores and RESETLOGS incarnations. That level of knowledge and understanding comes from studying available materials, as well as experience.

Likewise, troubleshooting sporadic node reboots in an Oracle RAC cluster requires knowledge of cluster design, and it requires knowledge of components in Oracle Clusterware and how they operate. Successful troubleshooting of Oracle Data Guard requires understanding of the processes behind Oracle Grid Control configuration screens and wizards.

But it's not enough to know only core database technology. In the first example, in order to troubleshoot failing RMAN tape backups successfully, a DBA often needs a good working knowledge of tape management software. If hands-on knowledge isn't there, it helps to at least be capable of talking to the storage administration team in the terms they understand.

Understanding of storage technologies is crucial to designing a database environment that is scalable and reliable. A modern DBA need to know about the different RAID levels, how SAN arrays are connected via the SAN network, what the implications are of using advanced features such as snapshots and cloning, and more.

Understanding of the development tools and methods has always been crucial, and it's even more important these days. As time-to-market for products and solutions is decreasing more and more, DBAs have fewer and fewer opportunities to engage in the development life-cycle.

DBAs often have no time to learn the specifics of a particular environment. In the past, knowledge of Oracle Forms would cover a significant portion of what a DBA needed to know about development tools. Those days are long gone. These days, there are dozens of frameworks and languages used by developers. A savvy DBA will want to have at least general understanding of those frameworks and languages.

There are a number of ways to keep yourself updated on new technologies and learn how things work. Here are some to consider:

- Books are a traditional source of knowledge for Oracle DBA. These days more and more books are published, so readers need to do a bit of research to make sure a particular book is a reliable source of high-quality information. Be careful to base your decisions on the quality rather than the number of books published; quantity doesn't automatically translate into quality.

- The Internet has made it possible to access thousands of articles that people share via blogs, forums, email lists, and other social networks. Just as with books, but to the greater extent, be careful not to trust blindly everything written on the Internet. Always question what you read, even if it comes from a known expert in the field. This chapter, of course, is an exception.

- Experience is something that a seasoned DBA can never replace with books and other reading. Just as solders develop their skills in battle, DBAs acquire critical skills by managing real database environments. This is the chance to apply the theory in the real life.

- A personal playground is something every DBA should create for themselves. These days, every workstation or laptop is capable of running a minimal database configuration. Availability of virtualization software makes it even easier; on my laptop with 4 GB of memory and a dual-core CPU, I'm able to run up to three nodes in a mini-RAC cluster. Of course, when it comes to real workloads, nothing beats a dedicated lab environment, but you wouldn't believe how much you can learn on a small database environment installed on your laptop.

RTFM

One source of knowledge that I want to emphasize specifically is vendor documentation. Yes, I'm talking about Oracle documentation that is freely available with any recent Oracle database release. Reading vendor documentation is often referred as RTFM or "Read The Fine Manual" (though people argue what F exactly stands for).

The *Oracle Database Concepts Guide* has become my favorite book from Oracle's documentation set. It's a great starting point for learning the basic concepts of all database components and how they fit together. The Oracle Database product has become very complex in the past years, and it's become impossible to know in depth every feature and how it works, but every DBA should know where to start when it comes to something new. The *Oracle Database Concepts Guide* often serves as that starting point.

Oracle's *SQL Language Reference* is another guide that you should keep handy and review with every new database release. Its syntax diagrams and comments are priceless, and many times have clued me in to important features that I never knew existed.

The *Oracle Database Reference* is where you'll find information about all of the dynamic and static dictionary views, init.ora parameters, wait events, and other goodies. I have learned to consult the version of this manual from the latest release even when working with a previous version. That's because the *Oracle Database Reference* is often updated more slowly than the features themselves. You can sometimes glean precious information by looking at the documentation for the latest Oracle software release even if you are working with previous versions.

Finally, while Oracle Metalink (or My Oracle Support as it's called these days) is not part of the official documentation, every Oracle DBA should be able to navigate in support articles and know how to research Oracle Support notes and bug reports. Those often help to fill the blanks in the puzzle when standard documentation doesn't provide enough detail.

Oracle Database is a complex product. It's physically impossible for a single person to know all its bells and whistles and to have in-depth knowledge of each feature and area. Take Oracle XMLDB, Oracle Streams, and Oracle Spatial as examples. Unless you have worked with these features, you are unlikely to know in detail how they work. However, you do want to have a general idea of such features. Most importantly, you want an idea of where to go to learn more. Just knowing where to go can help you get up to speed relatively quickly in any area of Oracle Database.

Facing the Unknown

Oracle Database is not only a complex product, it's also proprietary software. Oracle Corporation introduced significant instrumentation and provided lots of new documentation in the last decade, but there are still many blanks about how the product works, especially when it comes to the implementation of new features and of some advanced deployments that hit the boundaries of software and hardware. Whether it's because Oracle wants to keep some of its software secrets or because documentation and instrumentation are simply lagging, we always face situations that are somewhat unique and require deeper research into the software internals.

When I established the Battle Against Any Guess Party, a number of people argued that guesswork is the cruel reality with Oracle databases because sometimes we do hit the wall of the unknown. The argument is that at such point, there is nothing else left but to employ guesswork. Several times people have thrown out the refined term "educated guess." However, I would argue that even in these cases, or especially in these cases, we should be applying scientific techniques. Two good techniques are deduction and induction.

When we have general knowledge and apply it to the particular situation, we use deductive reasoning or deductive logic. Deduction is often known as a "top-down" method. It's easy to use when we have no gaps in our understanding. Deduction is often the path we take when we know a lot about the problem domain and can formulate a hypothesis that we can confirm or deny by observation (problem symptoms).

Inductive reasoning is often considered the opposite of deductive reasoning and represents a bottom-up approach. We start with particular observations, then recognize a pattern, and based on that pattern we form a hypothesis and a new general theory.

While these techniques are quite different, we can find ourselves using both at different stages as verification that our conclusions are correct. The more unknowns we face, the more we favor inductive reasoning when we need to come up with the generic theory while explaining a particular problem. However, when we form the theory via inductive logic, we often want to prove it with additional experiments, and that's when we enter into a deduction exercise.

When taking a deductive approach first, when applying known knowledge and principles, we often uncover some inconsistencies in the results that require us to review existing theories and formulate new hypothesis. This is when research reverts into inductive reasoning path.

Deduction and induction each have their place; they are both tools in your arsenal. The trick is to use the correct tool at the correct time.

Paradigm Shifts

The term *paradigm shift* was originally introduced by Thomas Kuhn in his book *The Structure of Scientific Revolutions* (University of Chicago Press, 3rd edition 1996). A paradigm is our perception of reality as we see it—our view of the world. Based on our previous experience, we interpret current observations and make seemingly logical conclusions. A paradigm shift is a sudden change in the point of view of how we see and interpret things.

Centuries ago, people used to think that Earth was the center of the universe. The realization that neither the Earth nor even the sun is the center of the universe is a typical paradigm shift. Another great example of a paradigm shift is Stephen Covey's story from *The 7 Habits of Highly Effective People* (Free Press, 2004). While traveling, he observed a quiet man with two sons running like headless chickens and behaving provocatively so as to irritate everyone around. When Covey finally asked the man why he didn't do something to control the kids, the answer followed "We just got back from the hospital where their mother died. I don't know how to handle it and I guess they don't either." This realization throws everything upside down and completely transforms one's perception of the situation.

One of the most revolutionary paradigm shifts that I personally had about Oracle Databases came from reading the book *Optimizing Oracle Performance* (O'Reilly, 2003) by Cary Millsap and Jeff Holt, fellow members of Oak Table Network. That book is what shifted my paradigm of performance troubleshooting. Performance is all about time, so time is what you need to focus on and analyze. It is response time that matters. Now that I understand the idea, it seems so natural that I can't imagine why I ever believed in relying upon indirectly related counters to troubleshoot Oracle database performance problems.

As much as you want to use generally known principles and knowledge, do question these principles regularly even if the questioning seems crazy at the moment. Stepping away from deductive reasoning doesn't mean you are guessing as long as you realize what you are doing—maybe you are about to make the next paradigm shift! However, don't forget that thousands of times, you'll likely just be wrong.

Experience Is Danger

I hear very often that nothing helps in troubleshooting so much as years and years of real-world experience in the field. While this is generally true, experienced analysts tend to fall into a trap of quick guessing based on previous situations.

Top-notch DBAs sometimes have successfully used their gut feeling and intuition to lead them towards the right solutions quickly. However, intuition should never dictate a solution, because there is always a danger that the particular case could be in some way different from what has been experienced so far. I've been in such situations a few times, and I've observed a number of very smart and experienced DBAs falling into the same trap.

It's very easy to reach erroneous conclusions based on previous experience. Let me give you an example. Working a lot with Oracle Real Application Clusters (RAC), I learned pretty much all the common symptoms and causes for node evictions in an Oracle RAC cluster. I knew there are a few components that cause node eviction, and I could always quickly determine by running Oracle 10*g* Release 2 on Linux (let's be specific) that it's one of three Clusterware components that can evict a node: the CSSD daemon, the OCLSOMON daemon, and the hangcheck-timer module (which is actually part of the Linux operating system).

If I blindly follow my experience, I could easily miss that the Oracle 10.2.0.4 patchset on Linux introduced a new OPROCD daemon, which becomes responsible for suicidal self-evictions instead of hangcheck-timer. Furthermore, I might never think about the OCFS cluster filesystem that has eviction functionality as well.

The weirdest node eviction case I ever experienced would have been extremely difficult to find by relying purely on my experience. It was when one client had an infrastructure platform that had its own watchdogs for server health. When a server's watchdog failed, a server reset was initiated remotely by a central watchdog. God knows how far I would have gone assuming that such a node eviction represented one of the "standard" RAC evictions.

We should use our experience to aid in scientific troubleshooting. We should never rely on ready-made solutions from past experience, but we can use that experience to take some shortcuts and save time and effort in analysis phase of a new problem. Our experience should help us narrow down the search for the root cause. We can't skip cause and effect analysis and, most importantly, validation of our conclusions.

Fixing the Root Cause?

We always talk about the root cause to fix. However, is it always crystal-clear what the root cause of a given problem actually is? Not really! There are always problems of the chicken-and-egg type.

Let's say we are troubleshooting a significant slowdown in application performance (online orders in the Internet book shop), including timeouts that happen regularly around lunch time. We've gone through the required troubleshooting and documented our conclusions:

- User activity is growing by 50 percent during the lunch-hour; that is, there are 50 percent more new orders during that time period.
- The number of online orders is growing steadily every month.
- The time spent on physical I/O contributes 90 percent to response time based on a performance profile we've built.
- Three SQL statements are responsible for about 90 percent of the I/O.
- Random single-block I/O time almost doubles during the lunch period, from 8 ms to 15 ms.

Can we identify the root cause from all these items?

On the surface, it seems clear that performance issues are caused by increased user activity. So would it be fair to attempt to decrease user activity and address the root cause in that way? Probably not, because it means sacrificing business growth and revenue from online orders. The business presumably wants the growth.

Perhaps, the root cause is the random I/O response time increase. If we dig further, we might realize that our I/O subsystem is working at the limit of its capacity, and that additional I/O activity simply cannot be handled without significant degradation in performance. We could also find out that there is a heavy batch running on another database that apparently uses the same physical disks on the same SAN. Is I/O capacity our root cause? Or could the root cause be bad scheduling for the batch?

"We always talk about the root cause to fix. However, is it always crystal-clear what the root cause of a given problem actually is? Not really! There are always problems of the chicken-and-egg type."

Does all this mean we need to scale I/O, or can we reschedule the batch job or separate the databases to different disks?

Don't forget about our three top SQL statements consuming 90 percent of I/O in the database. Perhaps the root cause is inefficient SQL that needs fixing. Or perhaps we require a heavy redesign of the data model to make that SQL scalable.

How do we choose what areas to improve without taking a guess? The answer is in the requirements—never lose sight of the business requirements. In this case, our requirement is to handle the current peak number of online orders. We might also have a requirement to be capable of handling three times the current order rate in one year as our business keeps growing.

The next factor is the cost and—often forgotten—common sense. If that heavy batch can be moved to another time, then moving it is probably the cheapest approach to a solution. If SQL tuning is not an option or requires a costly redesign of the underlying database, then adding I/O capacity could be easier at the moment. On the other hand, sustaining three times more traffic next year might change the equation, and redesigning the data model might end up being the most reasonable thing to do after all.

In fact, it might be the only option, as otherwise the bottleneck could move from the I/O subsystem to contention in the database instance caused by inefficient SQL.

Whenever you meet a chicken-and-egg problem such as I've described, make sure you keep the requirements in mind. Think about the requirements and the cost, and apply common sense.

Best Practices and Myths

"Best practices" has become an extremely popular concept in recent years, and the way IT best practices are treated these days is very dangerous. Initially, the concept of best practices came around to save time and effort on a project. Best practices represent a way to reuse results from previous, similar engagements. Current application of best practices has changed radically as the term has come into vogue.

What are now called best practices used to be called "rules of thumb," or "industry standards," or "guidelines." They were valuable in the initial phase of a project to provide a reasonable

starting point for analysis and design. Unfortunately, modern best practices are often treated as IT law—if your system doesn't comply, you are clearly violating that commonly accepted law.

Blindly implementing best practices is nothing different from guesswork; we are applying some past-proven solutions without measuring how they stand against our requirements, and without testing whether they bring us any closer to the targets we have. Industry has become so obsessed with best practices that we commonly see projects in which reviewing an environment for compliance with best practices is the ultimate goal.

Implementing and complying with best practices in the field is becoming an inherent part of an IT support engineer's job description. Just look at job postings for DBAs, or search for "DBA best practices" on any online job board.

So how do we put best practices to good use? The key is to understand best practices, how they work, and why each of them is applied. What problems is a best practice solving, and what are the side-effects?

We can then map best practices to our environments and see whether any of them bring us closer to satisfying our requirements.

Best practices might be useful during the initial phases of the project to come up with the most reasonable initial configuration and design with the least efforts and costs. Solving existing problems by blindly implementing best practices is pure guesswork and is generally more expensive than focusing on the problem and walking a scientific troubleshooting path.

Here is a recent example from my company. One of the clients recently asked their DBA team to perform a health check on their new environment to see if it is compliant with best practices, and to understand why they have regular performance problems. It was a complex environment with Oracle E-Business Suite on Oracle RAC, and four areas of focus were identified. The client insisted on a best practices review.

The team initiated two streams for this client. In the first stream, they reviewed configuration for compliance to best practices and documented all findings and the potential benefits of implementing missing best practices. None of those findings directly solved the performance problems of that client, according to our analysis.

In another parallel stream, one of my colleagues engaged the end-users and analyzed their problems. This engineer found very inefficient SQL. Apparently, the environment was customized heavily and that was done pretty sloppily, resulting in many layers of views.

The best-practices review stream took about 90 percent of the time, without getting us any closer to solving the client's most critical problem, regular performance issues. Implementing any of the best practices that the team identified would be nothing but pure guesswork. Possibly some of them could make a slight improvement, but guessing is still guessing.

The second danger of best practices is that they easily become myths. The technology keeps improving and issues addressed by certain best practices might not be relevant anymore in the next software version. A typical example of old guidelines (that is, best practices) is keeping checkpoint frequency low or using raw devices to get the best possible performance. The life-span of best practices is generally longer than their applicability time-frame. Inertia is part of human nature.

BattleAgainstAnyGuess.com

BattleAgainstAnyGuess.com is a web site I've created for everyone in the community to unite and stand against guess-based troubleshooting. You'll find stories, articles, and even the occasional video to help you in your efforts to take a rigorous and scientific approach to problem-solving.

Those of us who have signed up as members of the site lightheartedly refer to ourselves as the BAAG Party. We welcome you to join us. Join as a member. And if you have something to share, let me know that you wish to post it.

BattleAgainstAnyGuess.com can be a good reference whenever you see others guessing too much, or starting to ask guess-provoking questions. Every time you are about to guess the cause and solution to a problem, stop and think of the BAAG Party. ▲

"Blindly implementing best practices is nothing different from guesswork; we are applying some past-proven solutions without measuring how they stand against our requirements, and without testing whether they bring us any closer to the targets we have. Industry has become so obsessed with best practices that we commonly see projects in which reviewing an environment for compliance with best practices is the ultimate goal."

I Heart Recursive Common Table Expressions

by Iggy Fernandez

Iggy Fernandez

I enjoy attending RMOUG Training Days because it is an educational workshop, not a gigantic trade show like Oracle OpenWorld. The presentations are not "customer success stories" or sales pitches; they are technical presentations on technical topics that matter to database administrators and developers. Here is a picture taken at the speaker reception at Training Days 2010 sent to me by Dan Hotka who has a quarter of a century of experience with Oracle Database. Dan is now inventing the future of Oracle training with "Oracle Training at Your Desk;" I highly recommend it.

Seated from left to right: Guy Harrison, Steven Feuerstein, John Beresniewicz, Maria Colgan, and Daniel Liu. Standing from left to right: Iggy Fernandez (me), Chen Shapira, and Sumit Sengupta.

One of my presentations at Training Days 2010 was on recursive common table expressions in Oracle 11g Release 2. Oracle was late to the table with recursive common table expressions which have been part of the SQL standard since 2003 but, to Oracle's credit, it has provided the CONNECT BY clause for hierarchical queries from the very beginning. However, recursive common table expressions can be used for much more than hierarchical queries. Also note that Oracle uses the non-standard terms "subquery factoring" and "recursive subquery factoring" for "common table expressions" and "recursive common table expressions" respectively.

Here is a quick recap of common table expressions of the non-recursive kind. They are an alternative to inline views and make a complex SQL query much easier to read and maintain. Another advantage is that they only need to be evaluated once if they are used more than once within the query. My favorite example of using common table expressions to make complex SQL statements readable is the winning solution to the First International NoCOUG SQL Challenge by Alberto Dell'Era from Italy. A simpler example—Suppliers Who Supply All Parts—is in my book.

On to recursive common table expressions. A recursive common table expression (recursive CTE) contains subqueries called "anchor members" and "recursive members." The rows produced by the anchor members are processed by the recursive members. The recursive members produce other rows that are fed right back to them for further processing. Recursion stops only when the recursive members fail to produce additional rows. The explanation in the Oracle documentation is fairly cryptic but a good explanation can be found on the Microsoft Developer Network.

- Run the anchor member(s) creating the first invocation or base result set (T0).
- Run the recursive member(s) with Ti as an input and Ti+1 as an output.
- Repeat Step 3 until an empty set is returned.
- Return the result set. This is a UNION ALL of T0 to Tn.

Here's a simple example of a recursive common table expression: a number generator. The following example generates the consecutive numbers from 1 to 9. The anchor member generates the first row which is then processed by the recursive member. The recursive member uses the name of the recursive CTE as a placeholder for the output of the anchor member or a previous execution of the recursive CTE. In this example, each execution of the recursive member produces one more row. Recursion stops when nine records have been produced.

```
WITH

  -- The following number generator is a simple example of a
  -- recursive CTE. It produces the consecutive digits from 1 to 9.

Numbers(n) AS
(

  -- The "anchor member." It contains exactly one row (N = 1).
```

```
  SELECT 1 AS N
  FROM dual

  UNION ALL

  -- The "recursive member." Notice that it references the name
  -- of the recursive CTE as a placeholder for the results of
  -- the anchor member or the previous execution of the
  -- recursive CTE. Each iteration of the recursive member
  -- produces the next value of N. Recursive execution stops
  -- when N = 9.

  SELECT N + 1 AS N
  FROM Numbers
  WHERE N < 9

)

SELECT *
FROM Numbers;
```

The above example can be simply duplicated using the CONNECT BY clause as follows:

```
SELECT level AS N
FROM dual
CONNECT BY level <= 9;
```

Next consider a standard hierarchical query; an org-chart of managers and employees. First, here's the old solution using the CONNECT BY clause; it is short and sweet.

```
SELECT
  LPAD(' ', 4*(LEVEL-1)) || first_name || ' ' || last_name
    AS name
FROM employees
START WITH manager_id IS NULL
CONNECT BY manager_id = PRIOR employee_id;
```

The solution using recursive common table expressions is much more verbose. Note especially the SEARCH DEPTH FIRST clause; refer to the Oracle documentation for an explanation.

```
WITH

RecursiveCTE (employee_id, first_name, last_name, lvl) AS
(

  -- The "anchor member" of the recursive CTE. It locates
  -- employees who don't have any manager; presumably there is
  -- at least one such employee.

  SELECT
    employee_id,
    first_name,
    last_name,
    1 AS lvl
  FROM
    employees
  WHERE manager_id IS NULL

  UNION ALL

  -- The "recursive member" of the recursive CTE. Notice that it
  -- uses the name of the recursive CTE as a placeholder for the
  -- results of the anchor member or the previous execution of
  -- the recursive CTE. Each iteration of the recursive member
  -- locates the employees who report to the employees located
  -- in the previous iteration. Recursive execution stops when
  -- all employees have been located.

  SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    lvl + 1 AS lvl
  FROM
    RecursiveCTE r INNER JOIN employees e
    ON (r.employee_id = e.manager_id)

)

-- Go deep in order to produce records in exactly the same order
-- as the CONNECT BY clause. The default order of processing is
-- BREADTH FIRST which would produce all managers at the same
-- level before any of their employees; this is not not the
-- order in which the CONNECT BY produces rows. The pseudocolumn
-- seq# has been designated here to capture the order in which
-- records are produced by the recursive CTE; it will be used in
-- the main query.

SEARCH DEPTH FIRST BY employee_id ASC SET seq#

-- This is the main query. It processes the results produced by
-- the recursive CTE.

SELECT
  LPAD(' ', 4*(lvl-1)) || first_name || ' ' || last_name AS name
FROM RecursiveCTE
ORDER BY seq#;
```

From the two examples above, it might appear that a recursive CTE is little more than a verbose way of specifying what could be more succinctly achieved with the CONNECT BY clause. However recursive common table expressions are more powerful than the CONNECT BY clause. For example, consider the following example from the TSQL Challenges team:

- Given a list of products and a list of discount coupons, we need to use the following rules to find the minimum price for each product:
- Not more than two coupons can be applied to the same product.
- The discounted price cannot be less than 70% of the original price.
- The total amount of the discount cannot exceed $30.

Syed Mehroz Alam has provided an elegant solution to the above problem using recursive common table expressions.

My final exhibit is a Sudoku puzzle. It turns out that a Sudoku puzzle can be elegantly solved with recursive common table expressions. The solution was discovered by Anton Scheffer. The listing below is a heavily annotated version of Anton Scheffer's solution with some cosmetic changes for better understandability.

```
-- The following SQL statement solves a Sudoku puzzle that is
-- provided in the form of a one-dimensional array of 81 digits.
-- Note that a Sudoku puzzle is really a 9x9 square grid.

-- A recursive common table expression (CTE) is used to solve
-- the puzzle. The "anchor member" of the recursive CTE contains
-- the unsolved Sudoku puzzle. The "recursive member" generates
-- partially solved Sudokus. Each iteration of the recursive
-- member completes one blank cell. Recursive execution stops
-- when no more blank cells are left or if no value can be found
-- for a blank cell (meaning that the Sudoku has no solution).
-- All solutions are produced if the puzzle has multiple
-- solutions.

SET sqlblanklines on
SET linesize 132
SET pagesize 66

WITH

-- The following number generator is itself an example of a
-- recursive CTE. It produces the consecutive digits from 1 to 9.
```

```
Numbers(n) AS
(

  -- The "anchor member." It contains exactly one row (N = 1).

  SELECT 1 AS N
  FROM dual

  UNION ALL

  -- The "recursive member." Each iteration of the recursive
  -- member produces the next value of N. Recursive execution
  -- stops when N = 9.

  SELECT N + 1 AS N
  FROM Numbers
  WHERE N < 9

),

RecursiveCTE(PartiallySolvedSudoku, BlankCell) AS
(
  -- The "anchor member" of the recursive CTE. It contains
  -- exactly one row: the unsolved Sudoku puzzle.

  SELECT
    cast(rpad('&&SudokuPuzzle', 81) AS VARCHAR2(81))
      AS SudokuPuzzle,
    instr(rpad('&&SudokuPuzzle', 81), ' ', 1)
      AS FirstBlankCell
  FROM dual

  UNION ALL

  -- The "recursive member" of the recursive CTE. It generates
  -- intermediate solutions by providing values for the first
  -- blank cell in the intermediate solutions produced by the
  -- previous iteration. Recursive execution stops when no more
  -- blank cells are left or if none of the intermediate
  -- solutions generated by the previous iteration can be
  -- improved (meaning that the Sudoku puzzle has no solution).
  -- All solutions are generated if the puzzle has multiple
  -- solutions.

  SELECT

    -- Construct an intermediate solution by completing one
    -- blank cell.

    cast(
      substr(RecursiveCTE.PartiallySolvedSudoku, 1, BlankCell-1)
      || to_char(Candidates.N)
      || substr(RecursiveCTE.PartiallySolvedSudoku, BlankCell+1)
      AS VARCHAR2(81)
    ) AS PartiallySolvedSudoku,

    -- Locate the next blank cell, if any. Note that the result
    -- of instr is 0 if the string we are searching does not
    -- contain what we are looking for.

    instr(
      RecursiveCTE.PartiallySolvedSudoku,
      ' ',
      RecursiveCTE.BlankCell + 1
    ) AS NextBlankCell

  FROM

    -- The intermediate solutions from the previous iteration.
    RecursiveCTE,

    -- Consider all 9 candidate values from the Numbers table.
    Numbers Candidates

  WHERE NOT EXISTS

  -- Filter out candidate values that have already been used in
  -- the same row, the same column, or the same 3x3 grid as the
  -- blank cell. Note that a Sudoku puzzle is really a 9x9 grid
  -- but we are using a one-dimensional array of 81 cells
  -- instead. Recursive execution will stop if none of the
  -- intermediate solutions generated by the previous iteration
  -- can be improved.

    -- The position within the one-dimensional array of the first
    -- cell in the same row of the 9x9 grid as the blank cell is
    -- trunc((BlankCell-1)/9)*9 + 1. The position of the Nth cell
    -- is obtained by adding N-1. For example, if BlankCell = 41,
    -- then the positions of the cells in the same row are 37,
    -- 37+1, 37+2, 37+3, 37+4, 37+5, 37+6, 37+7, and 37+8.

    -- The position of the first cell in the same column of the
    -- 9x9 grid as the blank cell is mod(BlankCell-1,9) + 1. The
    -- position of the Nth cell is obtained by adding 9*(N-1). For
    -- example, if BlankCell = 41, then the positions of the cells
    -- in the same column are 5, 5+9, 5+18, 5+27, 5+36, 5+45,
    -- 5+54, 5+63, and 5+72.

    -- The position of the first cell in the same 3x3 grid as the
    -- blank cell is trunc((BlankCell-1)/27)*27 +
    -- mod(trunc((BlankCell-1)/3),3)*3 + 1. The position of the
    -- Nth cell in the same 3x3 grid is obtained by adding (N-1) +
    -- trunc((N-1)/3)*6. For example, if BlankCell = 41, then the
    -- positions of the cells of in the same 3x3 grid are 31,
    -- 31+1, 31+2, 31+9, 31+10, 31+11, 31+18, 31+19, and 31+20.

    (
      SELECT N FROM Numbers

      WHERE to_char(Candidates.N) IN
      (
        -- Check the contents of the row containing the blank cell

        substr
        (
          RecursiveCTE.PartiallySolvedSudoku,
          trunc((BlankCell-1)/9)*9 + 1
            + (N-1),
          1
        ),

        -- Check the contents of the column containing the blank
        -- cell

        substr
        (
          RecursiveCTE.PartiallySolvedSudoku,
          mod(BlankCell-1,9) + 1
            + 9*(N-1),
          1
        ),

        -- Check the contents of the 3x3 grid containing the blank
        -- cell

        substr
        (
          RecursiveCTE.PartiallySolvedSudoku,
          trunc((BlankCell-1)/27)*27
            + mod(trunc((BlankCell-1)/3),3)*3 + 1
            + (N-1)
            + trunc((N-1)/3)*6,
          1
        )
      )
    )

  -- Stop processing when no more blank cells are left.

  AND BlankCell > 0

)

SELECT PartiallySolvedSudoku "Partially Solved Sudoku"
FROM RecursiveCTE;
```

P.S. The recursive examples in this article will only work in Oracle Database 11*g* Release 2. Anton Scheffer has a solution for Sudoku using the MODEL clause and a solution for Sudoku using collections which work with Oracle Database 10*g* Release 2. ▲

Many Thanks to Our Sponsors

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

Long-term event sponsorship:

Chevron

Oracle Corp.

Thank you! Year 2010 Gold Vendors:

- Confio Software
- Database Specialists, Inc.
- Enteros
- Precise Software Solutions
- Rolta TUSC

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

TREASURER'S REPORT

Naren Nagtode, *Treasurer*

Beginning Balance		
January 1, 2010		**$ 26,779.37**
Revenue		
Membership Dues	21,570.00	
Meeting Fees	550.00	
Vendor Receipts	6,400.00	
Advertising Fee	1,000.00	
Training Day	1,200.00	
Sponsorship	–	
Interest	4.75	
Paypal balance	–	
Total Revenue		**$ 30,724.75**
Expenses		
Regional Meeting	6,039.25	
Journal	3,934.97	
Membership	346.19	
Administration	1,462.83	
Website	75.49	
Board Meeting	567.41	
Marketing	100.00	
Insurance	–	
Vendors	206.66	
Tax	1,600.00	
Training Day	–	
Accounting	–	
P.O. Box	–	
Total Expenses		**$ 14,332.80**
Ending Balance		
March 31, 2010		**$ 43,171.32**

One-Day Performance Seminar with Craig Shallahamer

Wednesday, August 18, 2010—CarrAmerica Conference Center, Pleasanton, CA

OraPub's One-Day 2010 Performance Seminar was created specifically for experienced DBAs and is squarely focused on advanced Oracle performance analysis. To develop the seminar, OraPub surveyed the most experienced students in the firefighting and advanced analysis courses and specifically asked what were the most memorable, valuable, and relevant aspects of the courses.

Craig Shallahamer is planning a day of intensive learning, which includes multiple aspects of Oracle performance tuning. We will learn how to analyze the resources consumed by competing processes and how to avoid hitting resource capacity limits. We will map the different performance problems and the specific techniques to approach each problem type. We will explore the relations between different components that impact performance and how they influence one another. Then we will take all this knowledge and learn how it applies specifically to the problem of free buffer waits, taking a deep dive into Oracle architecture in the process.

If you want to learn how to quickly solve Oracle performance problems, come take advantage of a full day of training by Craig Shallahamer on August 18, at the CarrAmerica Conference Center in Pleasanton for $400. Register by July 10 to receive an early bird price of $350.

Craig Shallahamer, founder of OraPub, Inc., is the ultimate Oracle guru. He is technical—having just released his latest book, *Oracle Performance Firefighting,* he has also authored more than 23 technical papers and the revolutionary book, *Forecasting Oracle Performance.* He is experienced—having co-founded the Core Technologies and the System Performance Groups at Oracle and taught more than 10,000 students on 6 continents in 23 countries. And, most importantly, he is a passionate educator who brings his life experiences and engaging presentation style to every class.

What makes this seminar unique?

- What you learn can be immediately applied and demonstrated to your management
- The content has been selected from OraPub's most advanced students
- Delivered by Craig Shallahamer, well-known Oracle performance expert and author
- A practical and insightful weaving of internals, method, and advanced performance analysis

Seminar Objectives

- Learning about the many methods Oracle uses to gather performance data, their advantages and limitations, and how the analyst retrieves and uses this data
- Analyzing Resource Consumption: Learn how to measure network, CPU, and IO consumption and, when possible, their capacity limit using both Oracle and non-Oracle data sources
- Mapping the Terrain: Learn how to create a map of the performance terrain enabling advanced analysis, including how to visualize the situation and extract insightful solutions
- Computing System Behavior Analysis: Learn how a specific performance solution affects the relationship between the user experience, application work completed, and resource utilization
- Resolving the Free Buffer Waits Event: Learn how to resolve the complex wait events "free buffer waits" by delving into Oracle internals and applying the previous seminar content

Early bird price (before July 10): $350 for NoCOUG members / $500 for non-members
Regular price: $400 for NoCOUG members / $600 for non-members

Contact training_day@nocoug.org or register at www.nocoug.org

Database Specialists: DBA Pro Service

DBA PRO BENEFITS

- *Cost-effective and flexible extension of your IT team*
- *Proactive database maintenance and quick resolution of problems by Oracle experts*
- *Increased database uptime*
- *Improved database performance*
- *Constant database monitoring with Database Rx*
- *Onsite and offsite flexibility*
- *Reliable support from a stable team of DBAs familiar with your databases*

CUSTOMIZABLE SERVICE PLANS FOR ORACLE SYSTEMS

Keeping your Oracle database systems highly available takes knowledge, skill, and experience. It also takes knowing that each environment is different. From large companies that need additional DBA support and specialized expertise to small companies that don't require a full-time onsite DBA, flexibility is the key. That's why Database Specialists offers a flexible service called DBA Pro. With DBA Pro, we work with you to configure a program that best suits your needs and helps you deal with any Oracle issues that arise. You receive cost-effective basic services for development systems and more comprehensive plans for production and mission-critical Oracle systems.

DBA Pro's mix and match service components

Access to experienced senior Oracle expertise when you need it
We work as an extension of your team to set up and manage your Oracle databases to maintain reliability, scalability, and peak performance. When you become a DBA Pro client, you are assigned a primary and secondary Database Specialists DBA. They'll become intimately familiar with your systems. When you need us, just call our toll-free number or send email for assistance from an experienced DBA during regular business hours. If you need a fuller range of coverage with guaranteed response times, you may choose our 24 x 7 option.

24 x 7 availability with guaranteed response time
For managing mission-critical systems, no service is more valuable than being able to call on a team of experts to solve a database problem quickly and efficiently. You may call in an emergency request for help at any time, knowing your call will be answered by a Database Specialists DBA within a guaranteed response time.

Daily review and recommendations for database care
A Database Specialists DBA will perform a daily review of activity and alerts on your Oracle database. This aids in a proactive approach to managing your database systems. After each review, you receive personalized recommendations, comments, and action items via email. This information is stored in the Database Rx Performance Portal for future reference.

Monthly review and report
Looking at trends and focusing on performance, availability, and stability are critical over time. Each month, a Database Specialists DBA will review activity and alerts on your Oracle database and prepare a comprehensive report for you.

Proactive maintenance
When you want Database Specialists to handle ongoing proactive maintenance, we can automatically access your database remotely and address issues directly — if the maintenance procedure is one you have pre-authorized us to perform. You can rest assured knowing your Oracle systems are in good hands.

Onsite and offsite flexibility
You may choose to have Database Specialists consultants work onsite so they can work closely with your own DBA staff, or you may bring us onsite only for specific projects. Or you may choose to save money on travel time and infrastructure setup by having work done remotely. With DBA Pro we provide the most appropriate service program for you.

DatabaseSpecialists

CALL 1-888-648-0500 TO DISCUSS A SERVICE PLAN

NoCOUG
P.O. Box 3282
Danville, CA 94526

NoCOUG Spring Conference Schedule

Thursday, May 20, 2010, at Oracle Conference Center, Redwood Shores, CA

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

8:00 a.m.–9:00	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Hanan Hit, NoCOUG president
9:30–10:30	**Keynote:** *Business-Driven IT Management With Oracle Enterprise Manager*—Sushil Kumar, Oracle Corp.
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Auditorium: *Best Practices for Upgrading to Oracle Database 11*g—Roy Swonger, Oracle Corp.
	Room 102: *Demystifying Oracle RAC Workload Management: Connection Balancing* —Alex Gorbachev, Pythian
	Room 103: *Hello Worldwide Web: Your First JSF*—Peter Koletzke, Quovera
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Auditorium: *Cost Effective Information Security with Oracle*—Roxana Bradescu, Oracle Corp.
	Room 102: *Demystifying Oracle RAC Workload Management: Run-time Load Balancing* —Alex Gorbachev, Pythian
	Room 103: *Oracle SQL Developer Unit Testing: Testing Your PL/SQL Code*—Kris Rice, Oracle Corp.
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Auditorium: *Extreme Performance Data Warehousing*—Maria Colgan, Oracle Corp.
	Room 102: *Oracle 11*g *Reference Partitioning: Benefits and Hazards*—Andrew Zitelli
	Room 103: *Recursive Common Table Expressions*—Iggy Fernandez, Database Specialists
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Auditorium: *Capacity Management for Oracle Database Machine Exadata v2* —Boris Zibitsker, BEZ Systems
	Room 102: *SQL Execution Plans, DBMS_XPLAN, and Cardinality Feedback*—Dave Abercrombie, Convio
	Room 103: *Integrating SOA and the Application Development Framework*—Shaun O'Brien, Oracle Corp.
5:00–	**Networking event hosted by Oracle**

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

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo–Stunt Plane Oracle Challenger © 2008 Ron Sellers, Oracle Corp.