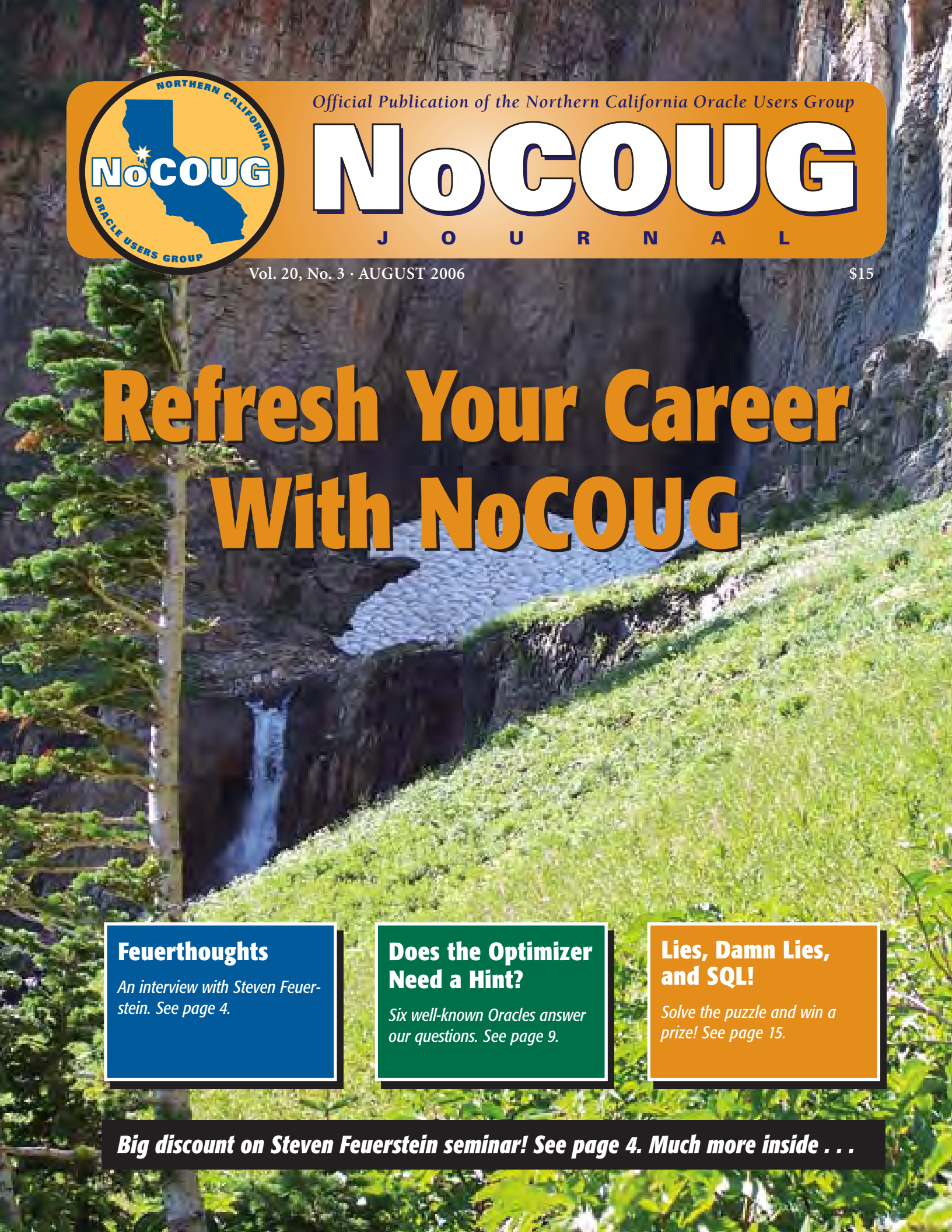
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 20, No. 3 · AUGUST 2006 $15



Refresh Your Career With NoCOUG

Feuerthoughts

An interview with Steven Feuerstein. *See page 4.*

Does the Optimizer Need a Hint?

Six well-known Oracles answer our questions. *See page 9.*

Lies, Damn Lies, and SQL!

Solve the puzzle and win a prize! *See page 15.*

Big discount on Steven Feuerstein seminar! See page 4. Much more inside . . .

Editor's Note

The *Journal* would not be possible if there weren't anyone to write for it; thanks to the authors for tolerating my nagging and nitpicking. Special thanks to the technical review panel of Ravi Kulkarni, Yury Tomashevich, Sumit Sengupta, and Brian Hitchcock. Thanks to Tom Wagner for another great cover picture of Mt. Timpanogos. *Finally, thanks to our vendors—to Karen Mead for a swell job of editorial proofreading, to Ken Lockerbie for great graphics design, and to Steve Voris for a quality print job.*

I hope you find this issue refreshing. PL/SQL guru Steve Feuerstein speaks his mind on everything from PL/SQL to peace on earth. A new panel of "Oracles" helps us figure out whether the Oracle Optimizer needs a hint! Performance tuning expert Chris Lawson offers up the first installment of the Performance Corner, and Brian Hitchcock reviews the latest book from Jonathan Lewis—pronouncing it a clear winner.

As always, I would love to hear your comments about the *Journal* and your suggestions for improving it. Please drop off a note at **journal@nocoug.org**, or let's talk at the next conference—on August 17 at the Chevron campus in San Ramon.

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

Editor's Note 2
NoCOUG Board 2
Publication and Submission Format 2
Advertising Rates 2
President's Message 3
Letters to the Editor 3
Interview—Steven Feuerstein 4
Ask the Oracles—Does the Optimizer Need a Clue? 9
SQL Corner—SQL Sucks! 14
Lies, Damn Lies, and SQL! 15
Book Review—Cost-Based Oracles—Fundamentals 17
Performance Corner 21
Tips and Tricks 24
Sponsorship Appreciation 25
Treasurer's Report 25
News Roundup 26
Spring Conference Abstracts 27
Spring Conference Schedule 28

–ADVERTISERS–

BEZ Systems 19
Quovera 19
Quest Software 20
VeriCenter 20
Database Specialists 22
Embarcadero Technologies 22
Data Domain 23
IT Convergence 23
Confio Software 24

Publication and Submission Format

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

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

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

ADVERTISING RATES

The NoCOUG Journal *is published quarterly.*

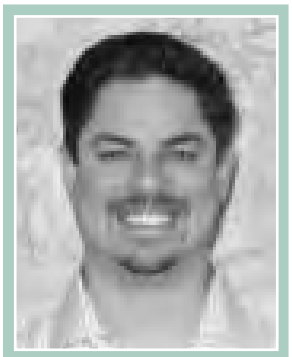
Size	Per Issue	Per Year
Quarter Page	$125	$400
Half Page	$250	$800
Full Page	$500	$1,600
Inside Cover	$750	$2,400

Personnel recruitment ads are not accepted.

journal@nocoug.org

Refresh Your Career with NoCOUG

by Darrin Swan


Darrin Swan

Training Days

NoCOUG brought you even more new opportunities to expand your Oracle knowledge this year by organizing training days with the best Oracle minds in the world. We received great feedback for *Optimizing Oracle—Performance by Design* by Jonathan Lewis. On August 18, Steven Feuerstein will speak on *21st Century PL/SQL*. Please visit **www.speak-tech.com/seminars/feuerstein_announce_aug06.html** and register before space runs out. Thanks to our Training Day Coordinator Hamid Minoui and Randy Samberg for their selfless labor. As a result of your great feedback, you can look forward to more training days in 2007.

NoCOUG Journal

The *NoCOUG Journal* is one of the signature benefits of membership, and our new editor, Iggy Fernandez, is continuing the tradition of excellence established by Lisa Loper and previous editors. Iggy is working hard to keep the journal fresh and interesting with the introduction of new columns such as *Ask the Oracles*. In this issue, our very own Chris Lawson has penned the first installment of *Performance Corner*. Chris is the author of *The Art and Science of Oracle Performance Tuning* and is a member of NoCOUG.

Summer Conference

We have an exciting lineup at our Summer Conference on August 17 at Chevron's campus in San Ramon. The keynote will be delivered by Steven Feuerstein, who will speak on *Software Programmers: Heroines and Heroes of the Twenty-First Century*. We also have 12 technical sessions for you, and the list of speakers includes author Don Burleson and Oak Table Member Kyle Hailey. There is something for everyone, from DBAs to developers—the back cover has the complete agenda. You won't want to miss it. I look forward to seeing you on August 17! ▲

Letters to the Editor

When I spotted the highlighted sentence "*We have two experts who like Flashback and one who doesn't,*" in Brian's Hitchcock's review of *Tales of the Oak Table*, I questioned whether my co-authors had expressed such definite feelings. Dave Ensor had argued that there are technical and logical reasons why the marketing message on Flashback queries is unrealistic; Connor McDonald had suggested them as one of several possibilities for working around a design flawed by read-consistency issues—cautioning readers to consider the inherent costs of read-consistency when investigating any such approach; and, in my own essay, I had discussed the use of Flashback queries in the context of reducing the inefficiencies of a workaround we had introduced to correct the logical error in a data extraction process—noting that there are only a limited number of scenarios in which the solution could be safely used to avoid some problems caused by read-consistency. The "Flashback Approval Hit-Ratio" you get for the book may well be 66% but it's worth checking how well the circumstances described by the authors match yours. Too many people look for simple *Yes-No* answers to questions that need thought, analysis and attention to detail and too many "experts" offer unsubstantiated directives—that *might* work *some* of the time in *some* circumstances—without explaining the costs and risks. The book offers a variety of opinions but you'll probably notice that the authors describe the circumstances, justify their comments, and aren't quite as contradictory as "hit-ratios" might suggest.

—Jonathan Lewis, England

Jonathan's letter was trimmed to fit the available space. —Editor

Thanks to *NoCOUG* for arranging *Optimizing Oracle*, a three-day seminar by Jonathan Lewis. It helped me understand why some of my SQL queries were performing badly and, the very next day after the seminar, I was able to reduce the duration of an ETL job from three hours to thirty minutes! I rate this seminar higher than the ones offered by Oracle University even though the fees were lower and hope that *NoCOUG* will organize many more seminars by stalwarts such as Jonathan in the future.

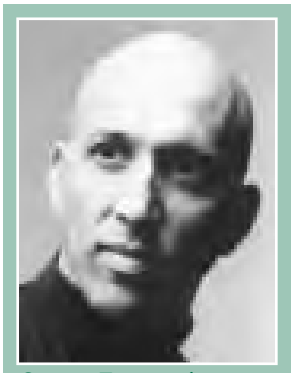
—Bala Pichai, California

Send your letters to **journal@nocoug.org**.

Feuerthoughts

An Interview with Steven Feuerstein

He couldn't move a mountain—nor pull down a big old tree
But my daddy became a mighty big man with a simple philosophy.
Do what you do, do well boy—do what you do, do well;
Give your love and all of your heart—and do what you do, do well.



Steven Feuerstein

NoCOUG welcomes Steven Feuerstein, considered one of the world's leading experts on the Oracle PL/SQL language, having authored or co-authored ten books on the subject—as well as a new book on MySQL! Steven currently serves as PL/SQL evangelist for Quest Software. He authors the popular PL/SQL Best Practice *column on the Oracle Technology Network. Steven is now busy building a unit testing tool for PL/SQL developers, code-named Qute—the Quest Unit Test Engine.*[1]

On Being an Oracle Guru

What does an Oracle guru do? Do you teach? Travel? Consult? Hobnob with captains of industry? Or do you hold down a day job like the rest of us?

Hmmm. Well, first I've got to tell you that I am not terribly comfortable with the "guru" thing. Mostly, I am just a very lucky guy who wrote the right book at the right time. After *Oracle PL/SQL Programming* was published in September 1995 and started selling like hotcakes, I was able to end my career as an hourly consultant. That meant in turn that I could devote more time to studying and playing around with PL/SQL. It's amazing what you can accomplish when you don't have a boss telling you what to do all the time!

[1] **www.unit-test.com**

So what do I do? These days, I spend virtually all my available time developing software to help programmers unit test their code. I am very excited about the impact that Qute—which we expect to release in October 2006—will have on the PL/SQL world. Beyond that, I do travel a fair amount, mostly doing trainings and seminars. My favorite venue is an Oracle User Group program. Oracle Corporation in Europe also likes to invite me over to train their customers. I spent two weeks in May skipping around through Scandinavia and the Baltics—Copenhagen, Stockholm, Oslo, Riga, and Tallinn in just one week—and finishing up in Prague, which is quite a beautiful city.

You certainly didn't become an Oracle guru overnight. Or did you? Where did you begin? How did you get to this stage?

I have two confessions to make: first, I have got to be one of the most narrowly specialized technologists in the world. I really just know one thing: PL/SQL. Fortunately, I know that pretty well—and fortunately also, Oracle has done a splendid job of designing, building, and enhancing the language. Sure, I used to write Fortran, but these days if you ask me about anything outside of PL/SQL you will draw a blank. Second confession: I am not really much of a technologist. I took three "101" courses on programming in college, and that is it for formal education. I have come to recognize over time that my strength is not that I am an amazing computer scientist sort of guy, but that I am an effective communicator. People seem to be able to actually read my stuff, understand it and then learn from it. And it hasn't hurt, from what I am told, to let my sense of humor appear in my books . . . .

OK, having said that, I got my beginning with Oracle as a presales person. I spent a couple of years following salespeople around to accounts and doing the dog-and-pony shows for them with SQL*Plus, SQL*Forms, etc. I can still remember my first public seminar for Oracle. Basically, what we were doing back in 1987 was running SQL*Plus scripts and saying, "Isn't this language amazing?" So I was given the standard script from Oracle for the presentation, drove up to Milwaukee with my boss, John Cordell, and got up in front of about 30 people to wow them. Now, I am a compul-

sive tinkerer, so I decided that there were some ways I could improve the script. Unfortunately, I am also not the most disciplined person so there were two bugs in this canned script! It was quite embarrassing, but I survived, and John was very nice about it.

Presales was interesting and sometimes exciting, but programming was way better, so I would constantly dabble with the Oracle tools, building little apps for my co-workers to use. This developed into TeamSell, a sales support application that caught the eye of Mike Fields, head of U.S. Sales in the early '90s. I was drafted to join a small dev team and we started building some very cool SQL*Forms apps to support the U.S. sales force. Then Larry canned Mike, and I was told to go back out on the road to help sell Oracle. I said no thanks, took the first consulting job I was offered, and ended up spending three years at McDonald's headquarters in Oak Brook. While there, I saw an appeal on CompuServe for Oracle authors and I thought "Why not? I can write." So I wrote that first book, which was the first independent text on PL/SQL, and it changed my life. From that point on, I was virtually a full-time student of the PL/SQL language, researching, writing, teaching, building code, etc. As I mentioned earlier, once you can structure your own time, all it takes is discipline—and reasonably good typing skills—and you can accomplish an awful lot!

Some people can perform seeming miracles with straight SQL, but the statements end up looking like pretzels created by somebody who is experimenting with hallucinogens.

I'm sure you have many interesting stories to tell about your travels and interactions. Tell us a story!

I am very pleased to have been so successful with PL/SQL. I truly have a wonderful life as a result. And I am sure that many companies have benefited from my writings. But what I find most rewarding is the impact I have had on the lives of individual programmers. It is not terribly uncommon to have a person walk up to me at a conference or training and tell me something like the following—which did happen in Chicago in 1998: "I was a union electrician at a steel mill in northern Indiana. When I got laid off, I went back to school and studied Oracle programming. I got hold of your book and it changed my life. Now I have a great job, my wife doesn't have to work and can raise our kids." Ah, that is so satisfying!

We like stories! Tell us another story!

I have traveled to something like 25 different countries, sharing my thoughts on PL/SQL. Whenever I go to a new city, I like to seek out the finest art museum and soak up the creative energies of the artists. I feel privileged to have been able to see so many incredible works of art over the years. I think that the impulse to create is one of the things that make humans special. I was in Prague in late May 2006 and took an extra day past my trainings to wander the city. Prague is a beautiful place and I encourage you all to visit, but what I found most incredible were the *sidewalks*. I am used to simple concrete slabs. In Prague, you will find that many sidewalks are actually mosaics made of two inch cubes of quartz and other rock, arranged in the most elaborate and intriguing patterns. You can read more about—and see—these sidewalks by visiting my blog.[2]

On PL/SQL

Why PL/SQL? Why not JDBC or SQLJ? What are the intrinsic advantages of PL/SQL? Flexibility? Efficiency? Can you give us some examples? Don't compiled languages have the advantage of speed over interpreted languages like PL/SQL?

Why PL/SQL? Why is my name Steven? Why do I live in Chicago? There are solid explanations for each of these, but it mostly comes down to: this is where my life has led me. I have made very few conscious career decisions in my life. I didn't seek out Oracle. I was just looking for another programming job, and it turned out the headhunter was filling presales job slots—my biggest concern about accepting the job was that I would have to wear a suit and tie each day. I sure didn't plan on becoming a "guru"—I just wrote stuff that people liked.

Having said that, PL/SQL is a very wonderful language. It is so simple and easy to use, and yet very powerful. It is certainly not as powerful or complete as Java, and there aren't nearly as many developers writing PL/SQL as Java, but it gets the job done. I like to call PL/SQL the "Cobol of the 21st century." It may not be terribly glamorous and it probably won't be used very much for new development 20 years from now, but we have used it to implement mission-critical production apps that run tens (hundreds?) of thousands of businesses around the world.

And 20, 30 years from now, programmers will still need to maintain and enhance those apps. Which is why it is *so important* to build comprehensive regression tests for the code we write today.

SQL is a set-oriented non-procedural language; i.e., it works on sets and does not specify access paths. PL/SQL on the other hand is a record-oriented procedural language, as is very clear from the name. What is the place of a record-oriented procedural language in the relational world?

Its place is proven: SQL is not a complete language. Some people can perform seeming miracles with straight SQL, but the statements can end up looking like pretzels created by someone who is experimenting with hallucinogens.

We need more than SQL to build our applications, whether it is the implementation of business rules or application logic.

PL/SQL remains the fastest and easiest way to access and

[2] **feuerthoughts.blogspot.com**

manipulate data in an Oracle RDBMS, and I am certain it is going to stay that way for decades.

Some might worry that using PL/SQL would chain them to Oracle and restrict their future options. What would you say to those you who wish to create database-agnostic applications?

Well, like I said earlier, PL/SQL is all I know, so I automatically self-select into working with and talking to people who *have* committed to both Oracle and maximizing Oracle technology.

If you want to create truly DBMS-independent code, you should at least take the time to write a complete layer of code, whether you call it a table API or a set of services, that *hides* the database activity. Then within that data access layer, you can write code that connects to one or more databases.

What development shops should work very hard to avoid, though, is writing any SQL statements in front-end logic. My feeling is that Java (and .Net and VB and . . .) programmers should *never* write SQL. They generally don't have much respect for the language and don't do a very good job writing it. Also, if you put your SQL in the application layer, it is much harder to maintain and optimize.

You write books, but you also create products and resources for PL/SQL developers. Tell us about some of your innovations, such as 21st Century PL/SQL and PL/Generator.

I like to create things. Sometimes this creative process is an excellent use of my time and sometimes . . . well, you know that joke about a tree falling in the forest? If you write a program and no one ever uses it, did you really write that program?

I have probably written more code that no one will ever run than most people have ever written, period.

So, yes, I have been creating all sorts of stuff over the years, ranging from products to resources. My first serious product effort was XRay Vision, a debugger for SQL*Forms V3. It was very cool stuff, written in SQL*Forms itself. Unfortunately, it came out when the product was on its last legs.

Since then, I have worked on all sorts of stuff: the original RevealNet Knowledge Base on PL/SQL (an interactive and very active form of information, now offered as the Knowledge Xpert™ from Quest); CGML, the Code Generation Markup Language, handy for describing design patterns; PL/Generator, a tool that generated comprehensive table APIs; PL/Vision, a library of some 1,000 PLSQL procedures and functions (now available as freeware on the Quest Pipelines[3]); and 21st Century PL/SQL, a webpage that collects ideas from the development community on how to improve the PL/SQL language.[4]

In the last few years, I have focused my creative juices on two tools: (1) Qnxo (Quality iN, eXcellence Out)[5], a design pattern factory tool that not only generates table APIs, but offers a set of about 700 scripts for PL/SQL programmers—I *hate* to reinvent the wheel; and (2) Qute (Quest Unit Test Engine), a unit-testing tool which allows you to describe through a UI the tests you want to run, and then generates the test code, runs the tests, and shows you whether or not your code worked, and if not, where it failed.

Quest Software just acquired both Qnxo and Qute. You will see Qute this fall at Open World with a new name and incredible functionality. It is going to completely change the way people think about and perform PL/SQL testing. And then we will apply the same techniques to MySQL-, SQL Server- and DB2-stored programs.

Do you have a pet peeve about PL/SQL? Have the limitations of the DBMS_OUTPUT function been fixed yet?

I just *love* to complain about PL/SQL! All in good fun and all with the intention of improving the language, of course.

Yes, DBMS_OUTPUT was a very easy target for my scorn. A maximum of 255 characters can be displayed? Ridiculous! And now, in Oracle Database 10*g* Release 2, it has been fixed: DBMS_OUTPUT.PUT_LINE will accept up to 32K of stuff to display.

Why did it take Oracle so long to do this—Chris Racicot, the PL/SQL Development Manager, told me that once they looked at the situation, they realized that all they had to do was change a single number! Argh! Because the PL/SQL team, while substantial in size, has a very long list of enhancements to work on, and many of them are still really big, complex, and absolutely critical—such as building an optimizer for the language, first released in Oracle Database 10*g* Release 1.

So things like DBMS_OUTPUT tend to fall by the wayside.

In another section of the* NoCOUG Journal*, we asked the question "Does the [Query] Optimizer Need a Hint" to a panel of performance experts. This is not directly related to PL/SQL, of course, but do you have an opinion on the matter that you would care to share?

Hmm. Well, I can always offer an opinion; it just might not be very well-informed. But in the United States these days, that almost seems to be a requirement to state an opinion, write laws, decide policy...oh, yes, PL/SQL, right.

Well, I am in many ways a very typical user of PL/SQL and Oracle technology generally: I just want it to work and work fabulously with an absolute minimum amount of attention

If you put your SQL in the application layer, it is much harder to maintain and optimize.

[3] **www.quest-pipelines.com**

[4] **www.oracleplsqlprogramming.com/IC/index.php**

[5] **www.qnxo.com**

and fuss on my part. So I am sure that the optimizer needs improvement and I look forward to that panel of experts helping Oracle figure out what is needed. For me, when I have a problem with my SQL, I use the tuning tools in Toad to identify ways to improve performance.

If I could only afford to buy one of your books, which one would you pick for me—and why?

Without a doubt—*Oracle PL/SQL Programming, Fourth Edition*. It is big, it is comprehensive and it is oh so much fun to read (or so people tell me, though I shudder when a programmer tells me that "I spent the whole weekend reading your book!").

If you can spare an extra $15 or so, I also suggest you pick up *Oracle PL/SQL Best Practices*. It is a concise presentation of my ideas for writing high-quality code. You will not, however, be able to incorporate *OPBP* into your fitness regime. It is too small and light. If you need to work on upper body strengthening, stick with *OPP4*!

What will we learn at your seminar?

After attending the 21st Century PL/SQL course, you will be aware of all the most important and interesting new features in the PL/SQL language. And much of that stuff is centered on PL/SQL collections, array-like structures that I consider to be the latest frontier and challenge for PL/SQL developers. If you don't know about or are not completely comfortable with collections, attend this seminar.

With collections, you can employ FORALL and BULK COLLECT to turbocharge your SQL statements, and you can build table functions that allow you to call a PL/SQL function as if it were a relational table. Great stuff!

Of course, you won't be an instant expert in applying the many features I will cover on that day. You will, however, have lots of new ideas, plus you will learn that you can always visit my PL/SQL portal[6], and download all of my training materials, along with supporting code, to follow up on the course with your own personal studies—feel free to copy and use all that code!

On the State of the Industry

All the jobs are going to India, Russia, and China. I know veteran IT professionals who now own health food stores, sell teddy bears on the Web, sell insurance, and patrol the streets of Berkeley on bicycles (bicycle cops). What's your advice for IT professionals? Should we give up and find other professions?

How do I feel about friends, coworkers, and others I meet in the U.S .struggling, when previously everything was golden? I feel awful about it.

I also feel great about people in India, Russia and China finally able to participate in the great world of software development, and thereby improve their own quality of life and those of their children.

I feel awful about friends, coworkers, and others I meet in the U.S. struggling, when previously every-thing was golden. I also feel great about people in India, Russia, and China finally able to participate in the great world of software development, and thereby improve their own qualify of life.

Software programming is a great profession and I think that anyone with training in the field should work hard to stay in it. Where else can you get a company to pay you to sit around and think about stuff, and write it down?

Yes, but how to hold on to those jobs? Speaking from my own experience, I would say that the trick is to specialize, to find a niche that fills a need that either not too many others are filling or know about, or is particularly challenging to do. If you can make yourself indispensable, you will find valuable and rewarding work.

Is Oracle facing a credible threat from open source databases such as MySQL and Postgres? Some even think that Oracle is trying to kill off the competition by purchasing InnoBase and Sleepycat. Computer Associates recently released commercial Ingres into the public domain—so a great deal of advanced technology is now available to open source developers. MySQL is working toward SAP certification and even wants to grab a piece of the data warehouse pie. What's the future of open source database technology? Why has it been so successful thus far? Will it continue its winning ways?

I do believe that database technology is becoming commoditized, at least on the low end—small databases, non-mission-critical applications, etc. This process will continue. Clearly, with enough time, even when people work without compensation on software, they can create amazing products.

It does seem like Oracle is trying to forestall and minimize the impact of MySQL, but I can't imagine that will work for very long—hey, that's part of the reason I co-authored a book on MySQL with Guy Harrison.

Oracle saw long ago that it needed to sell more than database technology, hence the move—with very mixed success to date—into applications.

As an owner of Oracle stock, I sure hope that Larry gets it right.

[6] **www.oracleplsqlprogramming.com**

The trick is to specialize, to find a niche that fills a need that not too many others are filling or know about, or is particularly challenging to do. If you make yourself indispensable, you will [always] find valuable and rewarding work.

Database technology has enabled the Information Age with all its wonders and benefits. On the negative side, our privacy and freedoms have been eroded as every detail of our lives from our shopping habits to our travel habits to our telephone calls is being captured in a database. The cell phone companies even have the ability to track us every minute of the day and may be storing every move we make in databases. How should society tackle this problem? Is legislation the answer?

Oh, my. That is a big question and requires an answer that far exceeds the scope of this interview. Fortunately, it is *precisely* the topic of my keynote presentation at NoCOUG's summer conference on August 17: *Software Programmers: Heroines and Heroes of the Twenty-First Century*.

Here's the bottom line—with ideas drawn liberally from Lawrence Lessig: Code, software, is a new form of law—human-created constraints on the behavior of other human beings.

We, software developers, have a responsibility to make sure that our software is used for beneficial purposes.

We write software; therefore we are in a sense lawmakers or enforcers of law. We, software developers, have a responsibility—individually and as a group—to make sure that our software is used for beneficial purposes.

Software developers should actively engage in whistleblowing, in challenging management on what they are building, when a corporation or government is using software in harmful ways.

More on that on August 17!

On Life Outside Work

What are your interests outside work?

Family, first and foremost, of course. I have been with my amazing, brilliant, and beautiful wife, Veva Silva, for 25 years. Our two kids, Chris and Eli, are wonderful human beings and both are artists—Chris is a well established muralist[7] and Eli is a jazz guitarist in the making.

Let's see. I turn 48 this September, which means my body is no longer the young, agile machine it used to be. So I spend a fair amount of time maintaining that machine—if you are having problems with your back, shoulders, neck, wrists, etc., get into a regular fitness routine. Most important: lots of abdominal exercises and stretching.

Beyond that, I have long complemented my software work with political activism. I spent years protesting our policies in Central America. Lately, I have gotten involved in Middle East peace issues. I serve as the president of the board of directors of the Refuser Solidarity Network[8], which educates the public about the Israeli "refusenik" movement.

Twenty or thirty years from now, I want to have some very good answers for my grandchildren when they ask me what I did during these years. So I get involved and do what I can. I urge all of you to do the same, according to your interests and beliefs.

I often wonder what advice I would repeat to my child if I had only a few minutes left on earth. We only have a few minutes left in this interview. What advice do you have for us—your fellow human beings?

As Spidey's uncle said, with great power comes great responsibility. We in the U.S., each one of us, actually do have great power. But we have largely abdicated responsibility. We should do everything we can to re-establish a real democracy, one that benefits the majority.

Our focus and priority should be on our children, the future of our species.

Finally and more generally, our focus and priority should be on our children, the future of our species. Make sure you spend lots of time with children, your own and others, enriching their lives, sharing your experience, giving them unconditional love and support. The world will be a far better place for it.

Thanks for providing me with this soapbox. I do so love to go on. ▲

Steven can be reached at **steven.feuerstein@quest.com**.

[7] **www.chrissilva.com**

[8] **www.refusersolidarity.net**

Does the Optimizer Need a Clue?

Ask the Oracles!

Gaja Krishna Vaidyanatha: Let me start with a life-altering philosophical question to my male readers! When your wife—or significant other—gives you a hint, what does it really mean? If you are a smart man, you will answer, "A hint from her is a directive." If you have answered the above question correctly, you probably understand the Oracle Optimizer very well. A *Hint* in a SQL statement is—in no uncertain terms—a directive to the Optimizer.

But wait—do Hints always work? Let us add some real-life perspective. It is Super Bowl Sunday and you are ensconced in your comfy leather couch, watching the game on TV and eating out of a bag without consciously tasting its contents. You are witnessing the dying moments of the grand finale of the NFL season. Your beautiful wife, who is fixing dinner while keeping tabs on the game, gives you a hint: "Honey, the trash is full!" You hear the hint, but choose to ignore it. Given that the context was inappropriate, you firmly believe that the hint is *invalid*! Your significant other—to your absolute dismay—then plants herself in front of the television and asks, "Honey, did you hear what I just said?"

When your wife or significant other gives you a hint, what does it really mean?

What your wife demonstrated was a real-life example of how to "Explain Plan." She gave you a hint, but followed up by asking you to "Explain Plan"! The moral of the story: if you introduce a Hint into your SQL, follow up by asking Oracle to "Explain Plan" and verify that the Optimizer is using it.

You may be interested in knowing that the Oracle Optimizer possesses "selective hearing" or "filtering" just as you and I do. There are many situations where the Optimizer registers the Hint that you give it, but chooses to ignore it even if the Hint is syntactically accurate. You can—and should—confirm this by asking it to "Explain Plan." To flash back to the final quarter of Super Bowl Sunday—taking the trash out during the dying moments of that game just did not seem to add up. It did not make sense. It was *invalid*! Thus you ignored it.

In a perfect world, we and the Oracle Optimizer may not require Hints. This is because, in a perfect world, we will always possess accurate statistics and the perfect context. The point to note here is that the Optimizer has to make split-second decisions based on previously collected statistics. Nine out of ten times, when an Optimizer picks the "wrong plan," it does so because of insufficient or inappropriate statistics. And the single most relevant reason when the statistics go bad is in the cardinality of column values.

The Optimizer and we do not live in a perfect world!

So you may ask, "What if I always computed statistics—using a sample size of 100%—on all of my objects and gave the Optimizer 100% accurate statistics? Will that be the perfect place to be?" The answer is no, not always. In fact, during a recent performance tuning engagement, we found that deleting the statistics on one of the tables and its associated indexes actually made a certain query run faster and consume fewer resources. The bottom line in that particular case was that the Optimizer chose an inefficient join method even when it had access to 100% accurate statistics, and that it did a significantly better job with default cardinality assumptions and dynamic sampling methods. Note that computing statistics using a sample size of 100% may not even be feasible if some objects are very large.

So then—does the Optimizer need a Hint? It sure does! Not always—just every now and then. Why? It is because the Optimizer and we do not live in a perfect world!

---

*Gaja Krishna Vaidyanatha is the principal of DBPerfMan LLC (**www.dbperfman.com**), an independent consulting firm specializing in Oracle Database Performance Diagnostics & Management.*

In recent years, he has worked in a product management role, providing strategic and technical direction to applica-

tion performance and storage management solutions for companies like Veritas Corporation, Oracle Corporation, and Quest Software.

He is the primary author of Oracle Performance Tuning 101 *and one of the co-authors of* Oracle Insights: Tales of the Oak Table. *He has presented many papers at regional, national, and international Oracle conferences. He can be reached at* **gaja@dbperfman.com**.

Guy Harrison: Introduced in Oracle 7, Hints allow the SQL developer to embed instructions in SQL statements that influence the Optimizer's choice of execution plan. When introduced, Hints were a significant improvement on the existing methods of changing execution plans, which essentially involved tricking the Optimizer into choosing a particular plan.

Hints can be either to advise the Optimizer or to override it. In the former case, we tell the Optimizer something about the desired behavior of the SQL. For instance, the FIRST_ROWS Hint allows the developer to advise Oracle to optimize for a certain number of rows—for example, for the first "page" of data in a query screen. However, most Hints allow the developer to override the Optimizer—by specifying a particular index, join order or method, or other optimization technique.

The developer must take responsibility for the performance of his or her code.

Hints that advise the Optimizer of the developer's expectations regarding the number of rows for optimization or the desired degree of parallelism are fairly benign, since they do not prevent the Optimizer from choosing any particular plan. Instead they provide the Optimizer with additional factors to consider when choosing between candidate plans.

However, Hints that directly request particular plans often cause problems over time, since they prevent the Optimizer from choosing a new plan when circumstances change. Execution plans will normally change in response to changes in table sizes or data distribution, changes in available indexes, or upgrades to the Oracle software. Hints might prevent the Optimizer from choosing a better plan as circumstances change or—more seriously—prevent the Optimizer from picking a good plan when a change (such as dropping an index) renders the original plan impossible.

In early incarnations of the cost-based Optimizer, these concerns were relatively unimportant, since the Optimizer's success rate was sufficiently low as to require Hints in SQL statements in a wide variety of circumstances. However, with today's Optimizer there are far fewer situations in which the Optimizer will produce a poor execution plan. Furthermore, alternatives to Hints now exist — we can use SQL profiles (10*g*) or stored outlines (9*i* and subsequent versions) to directly influence or even "freeze" a plan without having to modify the text of that SQL—which allows us to change the execution plan even when we don't have access to the application code.

The developer has the advantage of true—not artificial—intelligence.

Despite these advances I continue to believe that the developer must take responsibility for the performance of his or her code and—if necessary—may need to include an Optimizer Hint to do so. Although the Optimizer is an increasingly sophisticated piece of software, it is not aware of all the application requirements, data semantics, and other relevant information available to the developer. Furthermore, the developer has the advantage of being able to try alternative approaches and has the benefits of true—not artificial—intelligence. In many cases, the developer will be completely justified in directing the Optimizer to follow a specific execution plan.

In conclusion, the increasing sophistication of the Optimizer and the availability of alternatives such as stored outlines and SQL profiles make the use of Hints less of a necessity than in the past. However, Hints are still an important tool in your SQL tuning toolbox.

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

Jonathan Lewis: If you design your application perfectly, make all the smart choices with data structures, create all the relevant constraints, generate suitable statistical information about your data, and write carefully crafted code, then you will still find that there are a few statements that need Hints before the Optimizer follows the "best" execution path.

There are many reasons why the Optimizer may need help —including bugs, simple deficiencies in the Optimizer model, and the inherent problems of collecting, storing, and processing the complex statistics needed to describe real-world data—thoroughly described by other authors in this compendium.

There are many reasons why the Optimizer may need help.

Counterintuitively, another reason why the Optimizer may need help is that there are some extremely cunning strategies built into the runtime engine. Consider, for example, the following simple query.

```
select outer.*
from  emp outer
where outer.sal > (
 select  avg(inner.sal)
 from    emp inner
 where   inner.dept_no = outer.dept_no
);
```

There are two major strategies that the Optimizer could adopt for this query—create a result set with the structure (deptno, avg_sal) and do a join to the driving table (an un-nest operation), or scan the driving table and execute the sub query whenever necessary (a filter operation). If you want the un-nesting to happen, you could include the UNNEST Hint in the sub query; for the filter operation you could include the NO_UNNEST Hint.

Of course, you might try to avoid using Hints by manually un-nesting—rewriting the query as follows.

```
select  outer.*
from    emp outer,
        (
           select dept_no, avg(sal) avg_sal
           from   emp
           group by dept_no
        )  inner
where      outer.dept_no = inner.dept_no
and        outer.sal > inner.avg_sal;
```

Alas, if you do this in recent versions of Oracle you might then need to stop the Optimizer from doing a cunning—but possibly catastrophically inefficient—piece of complex view merging by including the NO_MERGE Hint in what is now the inline view (or a NO_MERGE (INNER) in the main query).

But why might you want to control the strategy that the Optimizer chooses for the original query anyway? Because there is a clever trick, known as scalar sub query caching, that can occur at runtime to minimize the number of times the filter sub query is executed—but it is impossible for the Optimizer to know how many times the filter sub query will actually run. (The Optimizer may be able to work out the minimum number of times the filter sub query has to run, but that's not necessarily a good estimate of the actual runtime activity.)

There are many reasons why the Optimizer may need help.

In this specific example it is likely that un-nesting will be the better option; in other cases it will be less obvious. And if the Optimizer chooses the wrong option, you have to give it a Hint or rewrite the query to make it do the right thing.

But look at the comment I made about rewriting this query. In Oracle 8*i*, my rewrite with the inline view could be a good idea—in Oracle 9*i* the inline view might get merged, with disastrous effects on performance.

The same type of issue appears with Hinting—you find a Hint that seems to solve a problem in one version of Oracle and causes a problem when you upgrade to the next version. (The ORDERED Hint is a good example of this in 8*i*, and the PUSH_SUBQ Hint is a good example in 9*i*). The biggest problem with Hints is that they are badly documented; it is almost impossible to find out exactly what each specific Hint is supposed to do, and if you don't know what a particular Hint does, how can you work out why it seems to solve a particular problem?

Hints can be very useful to solve urgent problems, but don't use them as a first resort.

Hints can be very useful to solve urgent problems—but my general advice is (a) don't use them as a first resort, (b) check whether the real problem is in the statistics, (c) if you really need to hint your SQL, you probably need an average of at least one Hint per table to lock in the execution path you expect, and (d) assume that you're going to have to revisit and retest any hinted SQL on the next upgrade.

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

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

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

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

Good Hints give the Optimizer additional information; bad Hints constrain it.

But habitual bad-Hint use is a bad thing. Especially since version 9.2, the Oracle Optimizer generally makes excellent decisions based upon the information you (or your DBA) give it. Having

When your Optimizer does something "crazy," don't reach for the Hints; find out why a good decision-maker has made a bad decision.

the Optimizer is a lot like having a really smart SQL performance expert in your office.

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

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

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

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

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

Prior to co-founding Hotsos in 1999, he served for ten years at Oracle Corporation as one of the company's leading system performance experts. At Oracle, he founded and served as vice president of the 80-person System Performance Group.

He has educated thousands of Oracle consultants, support analysts, developers, and customers in the optimal use of Oracle technology through writing, teaching, and speaking at public events.

Chris Lawson: I consider facility with SQL Hints to be of crucial importance to the serious performance specialist. When I help a client screen prospective performance DBAs, I always ask questions about common SQL Hints, such as USE_HASH, or ORDERED. Here's why: As the size of database tables grow, it becomes more and more likely that good performance will require a SQL Hint. It's not that the Optimizer works badly for big databases; rather, the consequences just become much more severe for being a little bit wrong.

Not knowing SQL Hints will seriously handicap anyone trying to improve performance—especially in large databases. For instance, when joining huge tables in a data warehouse, a small mistake in join order can add hours to the runtime of a batch job. As the number of tables in the join increase, the importance of join order (as well as the join method) magnifies.

Not knowing SQL Hints will seriously handicap anyone trying to improve performance.

There's another reason that Hints are important. For critical batch jobs, it's important that we achieve consistency in runtimes. Our customers simply can't tolerate huge variations in runtimes. Adding a SQL Hint is one way to achieve reasonably consistent runtimes. So, for example, if you know that a full table scan is the right choice most of the time, it's reasonable to lock in that execution plan.

There are a few Hints that I use frequently: PARALLEL, USE_HASH, FULL, NO_MERGE, LEADING, and ORDERED. Note that most of these are related to joins. This is where the Optimizer often needs help. Of course, some Hints don't really correct the Optimizer, but are used for special purposes. The Parallel Hint is one example. As your database grows, you will become very familiar with the Parallel Hint.

A SQL Hint is often appropriate when you know something special about the data distribution that gives you inside information on the best join order. Even though the Optimizer concludes that you are choosing an inferior join order, you know better, and instruct the Optimizer to choose your plan—even if the Optimizer doesn't really like your choice.

No Hint is better than the wrong Hint!

Remember—SQL Hints are powerful and oftentimes necessary, but using them incorrectly can be devastating. On one application I tuned, almost all the SQL Hints were inappropriate, and my first step in tuning was simply to remove the Hints! When applying a SQL Hint, remember that you are assuring the Optimizer that you know better than it does. The Optimizer

is willing to trust you, but you take on full responsibility for the consequences—good or bad. So, understand what a Hint does before you slap it on. No Hint is better than the wrong Hint.

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

Dan Tow: Some say that Oracle's cost-based Optimizer has ended the need for manual SQL tuning. This may sound plausible in theory, yet somehow manual SQL tuning has provided 75% of my livelihood for years. There are two common root causes for SQL performance problems:

First, someone may have made a mistake. This might be bad application or database design, missing indexes, a subtle error in the SQL functionality, or a DBA mistake, among others. No Optimizer can solve these problems.

Second, even with a perfect configuration and statistics, the Optimizer is sometimes unable to choose the right plan without a functionality-neutral change to the SQL—usually Hints.

These are both common, real-world causes of poorly performing SQL, with about 20% being of Type 2. These problems happen in the real world, and manual tuning of the worst-offending SQL is a highly efficient means to resolve both types of problems, without wasting time on the 99% of the SQL that already runs well. However, only Type 2 problems bear directly on the question "Does the Optimizer need a Hint?"—so it usually does not.

SQL tuning has provided 75% of my livelihood for years.

There are many reasons why even an excellent Optimizer may need Hints to deliver a good enough plan. Suffice it to say that a human tuner working at length on a short list of the worst-performing SQL may know much more than the Optimizer can see or consider during a sub-second parse step. To ask the Optimizer to entirely replace human tuning even for just the Type 2 problems described above is to ask it to win—or tie—every time, even against well-trained, experienced human tuners who enjoy enormous built-in advantages.

The uncontroversial "intent" Hints, such as FIRST_ROWS and ALL_ROWS, help even a hypothetical perfect Optimizer, conveying the developers' objective without limiting the Optimizer's choices. In contrast, the more controversial plan-limiting Hints may help today, but they can prevent future releases of the Optimizer from making even better choices.

Furthermore, a plan constrained by Hints has less freedom to adapt to changes to the data distributions that may make today's good plan terrible a year from now. Plan-limiting Hints are a two-edged sword, but if you limit Hints as follows they do far more good than harm:

First, don't tune with Hints just for minor improvements to the runtime. A fix that yields at least a twofold runtime improvement is a big, juicy "bird in the hand," compared to the possibility of further improvements representing some fractional "bird" in some future "bush." It is likely that the evolution of the application will result in changes to the SQL before those improvements materialize.

Hint carefully, but, when justified, by all means, Hint!

Second, when performing manual tuning—which generally is only necessary for the worst SQL—tune only with the intention of creating robust execution plans.

It is easy to choose an execution plan that is sensitively dependent on a dozen assumptions and data points being precisely correct—the Optimizer does it often! This plan may degrade horribly if any of those assumptions or datapoints are even moderately wrong. Such plans are not robust, although they may be technically "optimal"—i.e., fastest—for now. However, in my whole career, I haven't once needed a non-robust plan to get good enough performance. In practice, robust plans usually follow well-indexed paths and nested-loops joins to the larger tables, in a well-chosen join order, and these robust plans almost never need to change!

Hint carefully, but, when justified, by all means, Hint! ▲

Dan Tow has 16 years of experience focused on performance and tuning, beginning at Oracle Corporation from 1989 to 1998, where he headed the performance and tuning group for all of Oracle applications and invented a systematic, patented method (U.S. Patent #5761654) to tune any query efficiently. This method is extended and elaborated in his book, SQL Tuning. *Dan lives in Palo Alto, California, and is reachable at* **dantow@singingsql.com**.

SQL Sucks! – Part II

by Iggy Fernandez

Iggy Fernandez

Man is timid and apologetic; he is no longer upright; he dares not say "I think," "I am," but quotes some saint or sage . . . We are like children who repeat by rote the sentences of grandames and tutors, and, as they grow older, of the men of talents and character they chance to see,—painfully recollecting the exact words they spoke; afterwards, when they come into the point of view which those had who uttered these sayings, they understand them and are willing to let the words go; for at any time they can use words as good when occasion comes.
—Ralph Waldo Emerson

Summary

In the first part of this multi-part article, we explained that SQL is what is called a "non-procedural" language; i.e., an SQL query simply identifies a subset of data in the database without specifying *how* to go about extracting that data. This has led to the pervasive belief that application programmers have no responsibility for SQL performance. In this installment, we dive into the theoretical underpinnings of the SQL language—an understanding of which is crucial to taking responsibility for SQL performance. In the next installment, we will put everything together.

Dangerous Beliefs

The non-procedural nature of the SQL language has led to many dangerous beliefs centered around the theme that application programmers have no responsibility for the performance of the SQL statements they write. Here is a representative list.

- **Dangerous Belief #1:** DBAs bear chief responsibility for the performance of SQL statements.
- **Dangerous Belief #2**: Applications should be designed without reference to the way data is stored, e.g., index-organized tables, hash clusters, partitions, etc.
- **Dangerous Belief #3**: Application programmers should not tailor their SQL statements to make use of existing indexes. DBAs should instead create traps to catch badly performing SQL at runtime and create new indexes as necessary to make them perform better.
- **Dangerous Belief #4**: It is not necessary to review the Query Execution Plan of an SQL statement before releasing it into a production environment. It is further not necessary to *freeze* the Query Execution Plan of an SQL statement before releasing it into a production environment. It is desirable that Query Execution Plans change in response to changes in the statistical information that the query optimizer relies upon. Such changes are always for the better.
- **Dangerous Belief #5**: The most common cause of poorly performing SQL is the failure of the DBA to collect statistical information on the distribution of data for the use of the query optimizer.[1] This statistical information should be refreshed frequently.[2]

A True Story

I grub for my living in a dusty corner of a mighty telecommunications company, babysitting a brood of databases and feeding them whenever they are hungry for disk space. One day an irate developer submitted a high-priority request that we find out why Oracle was "not responding to simple queries."

We found that that the developer had submitted a seven-way join without any joining criteria whatsoever, i.e., a query of the form "SELECT . . . FROM Table#1, Table#2, Table#3, Table#4, Table#5, Table#6, Table#7!" Poor Oracle was gamely trying to perform a seven-way Cartesian product of the tables but probably needed 100 years to complete the task since the estimated query cost recorded in the V$sqlplan view was 9,275,840,000,000,000!

When we asked the developer why he had not specified any joining criteria, he said that he first wanted to determine if Oracle could handle a "simple" query before submitting a complex query. We offered to send him the Query Execution Plan for his "simple" query but he said that he did not know

[1] Consider, for example, the following statement found in an article published in a recent issue of the journal of the IOUG: *One of the greatest problems with the Oracle cost-based optimizer was the failure of the Oracle DBA to gather accurate schema statistics. . . The issue of stale statistics and the requirement for manual analysis resulted in a "bum rap" for Oracle's cost-based optimizer, and beginner DBAs often falsely accused the CBO of failing to generate optimal execution plans when the real cause of the sub-optimal execution plan was the DBA's failure to collect complete schema statistics*—**www.ingentaconnect.com/content/ioug/sj/2006/00000013/00000001/art00003**

[2] The following statement by Donald Burleson puts the finger on the dangers of collecting fresh statistical information for the use of the query optimizer: *It astonishes me how many shops prohibit any un-approved production changes and yet re-analyze schema stats weekly. Evidently, they do not understand that the purpose of schema re-analysis is to change their production SQL execution plans, and they act surprised when performance changes!*—**www.dba-oracle.com/art_orafaq_cbo_stats.htm**

how to interpret Query Execution Plans. He probably did not know much about SQL either![3]

Relational Algebra

SQL is largely based on the "algebra of relations," i.e., the ways in which relations (tables) can be combined with each other to form new relations. An SQL statement then is an algebraic expression in which the "operands" are tables instead of numbers. Here are five examples of relational operations.

"Selection"	Form another relation by extracting a subset of the rows of a relation of interest using some criteria.
"Projection"	Form another relation by extracting a subset of the columns of a relation of interest.[4]
"Union"	Form another relation by selecting all rows from two relations of interest. If the first relation has 10 rows and the second relation has 20 rows, then the resulting relation will have at most 30 rows.[5]
"Difference"	Form another relation by extracting only those rows from one relation of interest that do not occur in a second relation.
"Join"	Form another relation by concatenating records from two relations of interest. For example, if the first relation has 10 rows and the second relation has 20 rows, then the resulting relation will have 200 rows—and if the first relation has 10 columns and the second relation has 20 columns, then the resulting relation will have 30 columns.

It is possible to create new operations by combining the "primitive" operations in the above table. For example, "Natural Join" is a combination of Join and Selection.

We illustrate the five operations defined in the above table with an example. Consider the following table definitions.

- Suppliers is a table that contains SupplierName as its only column.
- Parts is a table that contains PartName as its only column.
- SuppliedParts is a table that contains SupplierName and PartName as its two columns. The occurrence of a certain combination of SupplierName and PartName in

[3] Steve Feuerstein, the guru of PL/SQL, said, in an interview for the *NoCOUG Journal*: *Java and .Net and VB programmers should never write SQL. They generally don't have much respect for the language and don't do a very good job writing it.*

[4] Duplicates are eliminated from the result.

[5] Duplicates are eliminated from the result.

Lies, Damn Lies, and SQL!

Sumit Sengupta from Columbus, OH, sent us this puzzle. Describe a combination of circumstances in which the COUNT function could produce the anomalous results seen in the example below.

```
SQL> DESCRIBE employees;
 Name              Null?     Type
 ----------------- -------- ---------------
 NAME               NOT NULL VARCHAR2(25)
 SALARY             NOT NULL NUMBER(8,2)
 COMMISSION_PCT    NOT NULL NUMBER(4,2)
SQL> SELECT COUNT(name) FROM employees;
COUNT(NAME)
-----------
          3
SQL> SELECT COUNT(salary) FROM employees;
COUNT(SALARY)
-------------
          2
SQL> SELECT COUNT(commission_pct) FROM
employees;
COUNT(COMMISSION_PCT)
---------------------
          1
SQL> SELECT COUNT(1) FROM employees;
 COUNT(1)
----------
          0
```

The prize offered for the most thorough answer is a SanDisk Sansa M240 1 GB MP3 Player. The SanDisk Sansa M240 is the most popular flash-based MP3 player sold by Amazon.com, ahead of the iPod Nano, and can be used with music subscription services such as Napster and Rhapsody.

The contest is open to all NoCOUG members. Send your answers to **journal@nocoug.org** by August 30. The decision of the judges is final. ▲

this table indicates that the supplier in question supplies the indicated part. Here is some sample data.

Suppliers

SupplierName
Ashley
Bertram
Carlton

Parts

PartName
Hammer
Nail
Screw

SuppliedParts

SupplierName	PartName
Ashley	Hammer
Ashley	Nail
Ashley	Screw
Bertram	Hammer
Bertram	Nail
Carlton	Screw

The question we try to answer is "Which suppliers supply all parts?" The answer is that only Ashley supplies all parts. Here is how we can formally obtain this answer with the help of the five relational operations defined previously. We create several intermediate result tables along the way.

1. First we use the Join operation and form an intermediate result table by concatenating records from the Suppliers table and the Parts table. All combinations of SupplierName and PartName occur in this table.

SupplierName	PartName
Ashley	Hammer
Ashley	Nail
Ashley	Screw
Bertram	Hammer
Bertram	Nail
Bertram	Screw
Carlton	Hammer
Carlton	Nail
Carlton	Screw

2. Next we use the Difference operation and form a second intermediate result table by extracting only those rows from the table obtained in the previous step that do not occur in the SuppliedParts table. The occurrence of a certain combination of SupplierName and PartName in this new intermediate table indicates that the supplier in question does not supply the indicated part.

SupplierName	PartName
Bertram	Screw
Carlton	Hammer
Carlton	Nail

3. Next we use the Projection operation and form yet another intermediate result table by extracting only the first column from the table obtained in the previous step. This is the list of suppliers who do not supply at least one part.

SupplierName
Bertram
Carlton

4. Finally we use the Difference operation again and obtain the final result we were seeking by extracting only those rows from the Suppliers table that do not occur in the intermediate result table of the previous step. This is the required list of suppliers who do supply all parts!

SupplierName
Ashley

SQL Solution

Here is the SQL language solution of the question, "Which suppliers supply all parts?" Once again we derive the result in stages to aid understanding. At each stage, we highlight the partial formulation of the previous stage.

1. First we "join" the Suppliers and Parts tables to obtain all combinations of SupplierName and PartName. We use the following language.

```
select SupplierName, PartName
from Suppliers, Parts
```

2. We next eliminate all combinations of SupplierName and PartName that do not occur in the SuppliedParts table.

```
select SupplierName, PartName
from Suppliers, Parts
minus
select SupplierName, PartName
from SuppliedParts
```

3. We then extract supplier names from the intermediate result obtained in the previous step, thus yielding the list of suppliers who do *not* supply *at least one* part.

```
select SupplierName from
(
 select SupplierName, PartName
 from Suppliers, Parts
 minus
 select SupplierName, PartName
 from SuppliedParts
)
```

4. Finally, we remove suppliers obtained in the previous stage from the list of suppliers in the Suppliers table, thus yielding the list of suppliers who *do* supply *all* parts!

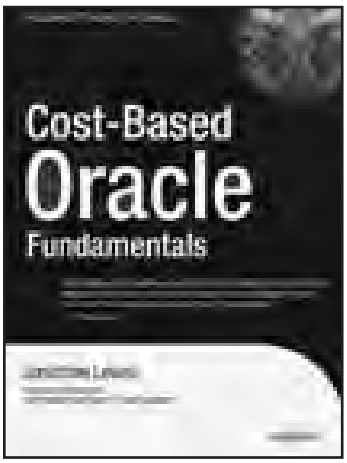
```
select SupplierName from Suppliers
minus
select SupplierName from
(
 select SupplierName, PartName
 from Suppliers, Parts
 minus
 select SupplierName, PartName
 from SuppliedParts
)
```

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

Cost-Based Oracle–Fundamentals

A Book Review by Brian Hitchcock

Summary



Overall review: Excellent, one of the best technical books I've seen.

Target audience: Experienced DBAs with specific interest in the details of how the Oracle Cost-Based Optimizer (CBO) works. This is not a book for beginners.

Would you recommend this book to others? Absolutely, but only if they are willing to spend considerable time and effort understanding the material.

Who will get the most from this book? Readers must have a midlevel understanding of Oracle database and DBA concepts.

Is this book platform specific? No.

Why did I obtain this book? I saw it on Amazon and wanted to read it. NoCOUG reimbursed me for the cost.

Overall review

This book covers many aspects of the Oracle Cost-Based Optimizer, how it works, how it gets confused, and why upgrades can cause so much trouble. The process is not simple and requires covering a lot of material to explain. The reasons why upgrading can cause performance problems is a subject that keeps coming up. While an upgrade might only cause the optimizer to make a new (and very poor) choice for one query, that one query can interfere with all the others and cause a significant slowdown for the whole database.

This is not a book for a first-time Oracle DBA. It contains an unusually high amount of real information and it requires concentrated study to get the most out of it. If you aren't already familiar with indexes, execution plans, etc., you won't get very far. Having said that, the author provides excellent summaries at the end of each chapter, and these summaries are a very good place for anyone to begin. While I encourage anyone interested in the CBO to read the whole book from start to finish, I think you could get a lot of valuable insights from reading the chapter summaries and then perhaps reading the individual chapters that you find most interesting. Further, while I think junior DBAs could be lost in this book, they could greatly benefit if they started reading and then looked up anything (and there will be lots of things) that they didn't already know about in other sources. This book could be used as an excellent way to become much more expert on the subject as long as the reader is willing to go to other sources for additional information.

This is not a book for beginners.

This book is also exceptional in that it is well crafted. I don't usually comment on this because most Oracle books are put together in a competent way, but this one is much better than the norm. The author's writing style is very fluid. You don't notice the writing, which means it is great. In most technical books, the author jumps from subject to subject without always completing a thought or point. I didn't see any of that here. This book was well written, edited, and produced. This book was fun to read even while it was at times challenging to absorb all the information. I actually looked forward to reading the next chapter because I wanted to find out what happened next. The Introduction tells us about files on the CD but there isn't any CD with the book, but I did check that all the code in the book is available for download from websites maintained by the author and the publisher. With the exception of the CD issue, I didn't find any errors in the text, which is unusual and reflects great work from all involved.

Relationships that are obvious to us are not known to the optimizer.

Chapter 1 What Do You Mean by Cost?

How the cost of a SQL statement is computed is reviewed starting with version 8*i* and going through 10*g*. This discussion was very interesting and helped me understand why our older databases could generate such poor execution plans. The author covers how the optimization code has evolved and this helped me understand why an 8*i* to 9*i*, upgrade can cause significant performance problems. This chapter reinforces the importance of testing upgrades on real datasets. The way the Cost-Based Optimizer works changes from version to version, so you must test the new version against a dataset that is the same size as your existing database. The size of your dataset can cause different issues in different versions of Oracle. The

author also provides three good reasons why reality doesn't always meet expectations.

Chapter 2 Tablescans

Four different strategies are presented for cost-based optimization. I didn't realize how complicated it was. And this is the point—it's easy to criticize the optimizer but when you really look at what has to be done, it makes you appreciate how hard the task is. CPU costing is described and the impact on upgrades explained. The optimizer can apply data for I/O and CPU performance, data that is measured or supplied by you, in the costing process. An example showing how the cost of the same query would be computed for both 8*i* and 9*i* is good. One of the best things I got from this chapter is that early versions of the CBO assume every data block visited required a physical read, as if none of the blocks would ever be cached. As the optimizer was improved, it moved away from this assumption and relied more on statistics. Again, a valuable insight into just how arbitrary the optimization process can be. The analysis of partitions and bind variables is great.

Chapter 3 Single Table Selectivity

More reasons that bind variables can cause trouble are revealed. When bind variables are involved, things can get complicated. The CBO may simply assume 5% cardinality because it doesn't have any other information about the data. Or, if bind variable peeking is used, the CBO could make decisions based on the actual data present when the SQL is first parsed. This may or may not be better than an assumption of 5%, since the CBO only peeks once when first parsing. The author provides several good discussions of the impact of bind variables, specifically "Overusing Bind Variables" (page 38), "Bind Variables and Ranges" (page 52), and "Bind Variable Peeking" (page 54). I recommend studying these sections carefully to gain an appreciation of the complex tradeoffs you have to make when deciding when to use bind variables. There are many well-known (and very good) reasons to use bind variables, but it is also good to understand how they can limit or change the options available to the CBO.

The way that some application vendors decide to "interpret" NULL values is fascinating—another reason optimization can be so difficult. Another great insight is that if columns of data are somehow related (an example of birth dates and zodiac signs is used) the optimizer will probably make poor choices. Relationships that are obvious to us are not known to the optimizer. The author presents many examples with lots of code. It can be hard to keep up but it is worth it. The examples lead you to the conclusion.

Chapter 4 Simple B-tree Access

Computing the cost of using an index is more complicated than I thought it was. Again, the process of optimization just isn't as easy as we think it should be. Examples show how a compound index may not be better than a simpler index. The reason that an upgrade from 9*i* to 10*g* can cause the optimizer to switch from table scan to index and why table scan was better is very good. The discussion of indexes on small tables is also good. There are many reasons that using an index may not be faster depending on the specifics of the data distribution.

Chapter 5 The Clustering Factor

The clustering factor is a measure of how randomly the data is distributed. The optimizer relies on this factor and if it doesn't accurately reflect your data, the optimizer will make poor choices. This chapter explains how things like reverse key indexes, ASSM, column order, and extra columns all affect Oracle's computation of the clustering factor. I didn't appreciate how much the data distribution affects Oracle's ability to generate a good execution plan.

Chapter 6 Selectivity Issues

The discussion of how the use of non-numeric data types can cause optimization problems is great. If an application is set up in which dates are stored not as Oracle date data type but as numbers or character strings, the optimizer assumes that there will be a continuous set of data when in reality we have gaps between sets of data. For example, the end of January (20060131) is followed by February first (20060201), but since the dates are stored as numbers or character strings, Oracle assumes there could be data in between these dates. This leads the optimizer to make huge mistakes. The use of NLS makes this worse, and deciding to use some extreme value to represent NULL (December 31 of the year 4000 for a NULL date) is also bad. If you use an application that is trying to be "database independent," you need to watch for these issues. There is a lot to learn from this chapter.

Chapter 7 Histograms

A histogram is just a picture of your data. How histograms can help the optimization process is covered. They help the optimizer when the data set is non-uniform, but they are expensive so you don't want to use them in all cases. Bind variables can cause histograms to be ignored. Given the cost of maintaining histograms, it's good to study the situations in which they won't be used by the CBO. The author covers this in detail (page 166). Keep an open mind about using and not using bind variables; the tradeoffs are complicated. There are several diagrams in this chapter that really helped me understand how Oracle uses histograms and the many ways they can be misused or ignored.

Chapter 8 Bitmap Indexes

The CBO doesn't care if the data is scattered or clustered and unless the data is static, bitmap indexes are more trouble than they are worth. Using CPU costing can affect the use of bitmap indexes. This chapter was difficult.

Chapter 9 Query Transformation

How the CBO can change a query during the costing process changes from version to version, which is one reason that upgrades can cause performance issues. The author describes using various hidden init.ora parameters. I don't think this will be particularly useful for most DBAs in production environments. A very interesting example covers NULL and NOT IN and why constraints are valuable. The use of hints is covered as well as the increased maintenance they require and the upgrade risks they create. If you are using third-party software, do you know if the vendor used Hints? I think the author assumes the reader has control of the SQL.

We are advised to generate execution plans for all critical queries before any upgrade. While I agree that this would be a good thing, I don't think most DBAs have the resources to do this or to review all the same execution plans after an upgrade.

Chapter 10 Join Cardinality

I found it very interesting how Oracle "computes" the cardinality of joins, which it uses to determine the best join order. NULL issues come up again and it turns out that the order of the tables in the SQL actually does matter.

Chapter 11 Nested Loops

I am familiar with how nested loops work, but it was interesting to see how the execution plan that Oracle comes up with may not agree with what happens when Oracle actually executes the plan at runtime.

Chapter 12 Hash Joins

This was interesting because I didn't have a clear picture of how a hash join is done and why it can be faster. I also didn't know that you could generate a trace file specifically for hash joins with the 10104 trace flag. A very good read: second only to the next chapter.

Chapter 13 Sorting and Merge Joins

I think this was my favorite chapter. I found it interesting, I got the most information from it, and I think it will be useful to me at work. The author is excellent at communicating ideas, as illustrated by his card-dealing analogy. The description of the three ways that sorting gets done was great. I had no idea that there are these various stages, and now I can more fully appreciate why sorting to disk is such a bad thing. But it's always a tradeoff. Increasing memory to reduce sorts to disk causes increased CPU usage. There can be situations where you get the best performance with reduced memory and a single pass sort to disk. There are specific boundaries where the sorting changes from multi-pass to disk, to single pass to disk, and finally to all sorting in memory. Adding more memory doesn't change things unless the increased amount of memory crosses one of these thresholds. This chapter makes all of this clear and provides a much more sophisticated explanation of how sorting works. Also good are the description of memory allocation and a concise summary of why most of what you can tinker with in Oracle isn't a good idea for the long term. Highly tuned systems are not robust. All the hours you might spend tuning your 9*i* environment could be completely lost when you upgrade to 10*g*. You need to balance the possible immediate benefits of any tuning exercise against the longer-term risk that your tuning will makes things worse after the next upgrade. A specific example is covered that demonstrates how the runtime engine does things differently from what you get from the execution plan. Again, remember that after all the time you spend getting the execution plan you want, Oracle may go off and do things differently.

All the hours you might spend tuning your 9i environment could be lost.

Chapter 14 10053 Trace File

This chapter is very focused on the specifics of interpreting the trace file for the optimizer. I have only needed to look at a 10053 trace file once, but when I needed to, it was useful. Even if you haven't needed to do this, it is interesting to follow what the CBO is doing step by step as it decides what the cost of each possible execution plan is. Sometimes the process seems very straightforward and other times it seems rather random.

I look forward to the other two volumes that the author promises on the subject.

Appendix A Upgrade Headaches

This brings together the many observations made throughout the book about how upgrades can (and will?) cause performance issues.

Appendix B Optimizer Parameters

This lists the init.ora parameters that affect performance. Many of these are hidden parameters that I would not change.

Conclusion

If you want to spend the time needed to follow all the material and you want to learn this much about the Oracle Cost-Based Optimizer, this book is nearly ideal. It is very readable and very well written, edited, and produced. I believe the time I put into reading and looking up related subjects was well worth it. I look forward to the other two volumes that the author promises on this subject, and will certainly buy the trilogy when it becomes available from a major Hollywood studio on DVD. Finally, I also learned a little Latin (Tempus Fugit) and that's not happened to me when reading an Oracle book before.

About the Author

Jonathan Lewis has been specializing in Oracle for the last 17 years. He is a renowned international speaker, both for his presentations at public conferences and for his seminars and tutorials. Jonathan is currently a director of the UK Oracle User Group and writes regularly for their *Oracle Scene* magazine. He authored the acclaimed book *Practical Oracle 8*i, and runs the popular **www.jlcomp.demon.co.uk** website.

Brian Hitchcock has worked at Sun Microsystems in Newark, California, for the past 11 years. He is a member of a DBA team that supports 2400+ databases for many different applications at Sun. He frequently handles issues involving tuning, character sets, and Oracle applications. Other interests include Formula One racing, finishing his second Tiffany Wisteria lamp, Springbok puzzles, Märklin model trains, Corel Painter 8, and watching TV (TiVo rules!). Previous book reviews by Brian and his contact information are available at **www.brianhitchcock.net**.

An *Impossible* Solution

By Chris Lawson

Chris Lawson

There are more things in heaven and earth, Horatio, than are dreamt of in your philosophy. —Hamlet, in the play by William Shakespeare

Performance specialists are always on the lookout for interesting tuning problems—especially the "impossible" ones. So let's temporarily forget about all those SQL problems solved by adding indexes, gathering statistics, or rewriting an "Exists" clause. It's much more fun to delve into the really tough challenges, preferably those running on large production systems with many angry users.

I recently came across one such performance problem. The beauty of this problem is that it initially appeared to be impossible to fix (short of redesigning the application.) When I first saw this problem, I honestly thought I could do nothing to fix it. The solution, however, turned out to be very simple —with impressive performance gains.

In this paper, we'll show you the performance puzzle and lead you through the solution. First, though, let's set the stage by seeing what was bothering the users.

The Frustrated Engineers

At the headquarters of a large engineering construction firm, project engineers use a custom application they call "Pipe-Plan." This application helps the managers optimize their project planning—personnel placement, scheduling, parts cataloging, and so on.

One important function lets an engineer check the parts usage at all 1,000 job sites. The user simply enters the part number or description, and the program displays the desired information. Unfortunately, the project engineers were seeing a delay of about 30 seconds each time they ran this function. This delay might not seem like a lot, but in contrast, most of the other functions returned in just a few seconds.

The Essence of the Problem

It was easy to isolate the problem SQL. The SQL in this query was well designed, with appropriate indexes on the underlying table. Here's what the SQL looked like:

```
Select Sum (In-spec), Sum (Out-of_Spec)
from Bill_of_Materials BOM
Where Part_Number = 'PIPE-100-INCH';
    [the actual SQL used a bind variable]
```

Note that the Bill_of_Materials table is huge—about 300 million rows. This means that very few of the table blocks will be cached. So if we need to retrieve 1,000 rows (one per job site), there will be nearly 1,000 disk reads. In addition, we expect that there will be some disk I/O for the index lookups.

A quick check of the V$Sql view yielded actual empirical data. Oracle needed 1,500 disk reads to execute the query. Since our disks weren't too fast (only 50 reads/sec), this explains the 30-second delay. This statistic confirmed that we had indeed found the root cause of the performance problem. The question then became, how do we either eliminate or expedite these reads?

Possible Solutions

There are a few innovative ways of speeding up the query. For instance, one way is to eliminate the need to query the huge Bill_of_Materials table. We could preprocess data, so that the pipe data would be available without reading this gigantic table. We might make a materialized view, refreshed overnight, that contains the desired information for all parts. Then, our problem SQL could query the materialized view instead. The materialized view would be far smaller than the Bill_of_Materials table; many more blocks would likely be cached, so disk I/O would go down drastically.

This first idea was rejected because the materialized view would frequently be out of date. For instance, it would not contain any parts recently added by another analyst. The engineers would thus not see these new parts until the next materialized view refresh. Keeping data current via "fast refresh" or using triggers was not desirable because other jobs frequently performed huge numbers of inserts and updates.

A second possibility would be to perform the troublesome processing in the background. The engineer would enter the Part ID, and then press "Go." The screen would return immediately so that the user could perform some other task. The requested information would be available shortly afterwards. This second approach was rejected due to the cumbersome process that the analysts would have to follow. Also, the change would involve considerable design changes to the application.

Fortunately, there was yet another way of solving this tough performance problem. It turned out to be very simple to implement and transparent to the users.

Our Approach

Our solution to solving the disk I/O bottleneck was not to avoid the disk I/O, but simply to get it over with sooner. The usual methods for speeding up table access would not work, however. For instance, we certainly couldn't perform a full scan

of the Bill_of_Materials table. Even with a high degree of parallelism, the delay would be huge. Clearly, we needed to perform a lot of single reads, but somehow we had to get a bunch of processes working together, sharing the work.

One possible way to get multiple processes going would be to manually launch a bunch of Sql*Plus sessions, then divide up the work somehow. Assuming that there is spare CPU and disk capacity, this would be like "multithreading" the problem SQL. Although this might involve a major design change to the form, it would likely speed things up quite a bit.

A Strange but Better Way

It turns out that there's a simple way to initiate multithreading—without resorting to redesign of the application. We will indeed use Oracle parallelism, but in the opposite way that it is usually employed. Whereas parallelism usually involves full scans of tables (or indexes), we will coax it to do lots of single block reads—sequential reads, in Oracle terms. This seems crazy, but it actually works. Here's how we do it:

Step 1

First of all, if we're going to subdivide the work into multiple threads, the optimizer has to have a driving table that encompasses the total number of units to process—in this case, the large number of job sites. So, we'll simply add the table, Job_Sites, which is a list of all 1,000 locations, and join it to our Bill_of_Materials table.

```
Select Sum (In-spec), Sum (Out-of_Spec)
from Job_Sites JS,  Bill_of_Materials BOM
Where Part_Number = 'PIPE-100-INCH'
And JS.Site# = BOM.Site#;
```

Step 2

The next step is to invoke Oracle's parallelism in our special way. We use the following hint:

```
/*+Leading (JS) Parallel (JS 30) Use_NL (JS BOM) */
```

The SQL hints above instruct the optimizer to start 30 threads beginning with the Job_Sites table. We also want each thread to perform a nested-loop join to the huge Bill_of_Materials table. Here's the final SQL:

```
Select /*+Leading (JS) Parallel (JS 30) Use_NL
(JS BOM) */
Sum (In-spec), Sum (Out-of_Spec)
from Job_Sites JS,  Bill_of_Materials BOM
Where Part_Number = 'PIPE-100-INCH'
And JS.Site# = BOM.Site#;
```

With the above SQL construction, Oracle automatically starts up separate sessions—just as if we manually started a bunch of Sql*Plus sessions. The key to this technique is that we specify the parallelism on the Job_Sites table—i.e., the small table! That's because we want all the processes to begin on that table. Note that this is the opposite of what we normally do with Oracle parallelism, where the parallel hint goes on the huge table.

Surprisingly, tests show that we can successfully launch a huge number of threads without contention. In fact, the opti-

mum number is far more than the parallelism degree we would normally specify. Sample tests confirm that using 30 threads on a server with 16 CPUs works nicely. Admittedly, we don't quite get 30x improvement, but close. The actual measured gain is about 25x.

Testing

When our problem SQL is running, we should see 30 processes, each performing a join to the Bill_of_Materials table. So if we look at the wait events for these children, we should see most children waiting on a sequential read of Bill_of_Materials. Here's a script to see what the threads are waiting on:

```
select x.server_name
         , x.pid as x_pid
         , x.sid as x_sid
         , w2.sid as p_sid
         , v.osuser
         , v.schemaname
         , program
         , w1.event as child_wait
         , w2.event as parent_wait
   from  v$px_process x
       , v$lock l
       , v$session v
       , v$session_wait w1
       , v$session_wait w2
   where x.sid <> l.sid(+)
   and   to_number (substr(x.server_name,2)) =
l.id2(+)
   and   x.sid = w1.sid(+)
   and   l.sid = w2.sid(+)
   and   x.sid = v.sid(+)
   and   nvl(l.type,'PS') = 'PS'
and x.status not like 'AVAIL%'
and w2.event not like 'SQL*Net%'
   order by 1,2;
```

Restrictions and Tips

This multi-threading technique is most useful for joining to huge tables containing hundreds of millions of rows. For smaller tables, the disk I/O will be much lower, so this technique will not be needed. Instead, you can simply use the conventional nested-loop or hash join methods. Also, this technique will not work with queries containing clauses such as "WHERE IN" or "WHERE EXISTS." The presence of "OR" conditions may also block proper operation. In these cases, Oracle's optimizer is apparently not able to divide up the work and coordinate the nested-loop slaves. Again, to ensure you're getting the full benefit, it's important to confirm that all the children processes are performing single-block reads. Look for the wait event "db file sequential reads" in the output from the script above.

Wrap-Up

After the job finishes, the V$Sql view will show the total of the elapsed runtime for all child processes. The clock time seen by the user will simply be V$Sql.Elapsed_Time divided by the number of executions (i.e., the degree of parallelism.) For a fuller explanation of this method, see *Creative Multithreading Using Oracle Parallelism*, available at **www.oraclemagician.com**. ▲

*Chris Lawson (**Chris@OracleMagician.com**) is an Oracle consultant and writer, and the editor of* The Oracle Magician *magazine (**www.oraclemagician.com**). Chris is also the author of* The Art & Science of Oracle Performance Tuning *(Apress), which describes performance-tuning techniques in great detail.*

Tips and Tricks

by Roger Schrag

TKPROF formats the information contained in the trace file generated when you set SQL_TRACE to TRUE. Beginning with Oracle 9*i*, if you set STATISTICS_LEVEL to ALL, the trace file records how many Consistent Reads, Physical Reads, and Physical Writes are required at each step of your SQL query—as well as the amount of time spent at each step. Here is an example.

```
 Rows   Row Source Operation
-----  -------------------------------------------------
  10    SORT GROUP BY NOSORT (cr=2004 r=656 w=0 time=9656825 us)
 990      TABLE ACCESS BY INDEX ROWID ORDERS (cr=2004 r=656 w=0 time=9647523 us)
1001        NESTED LOOPS (cr=23 r=6 w=0 time=13367 us)
  10          TABLE ACCESS BY INDEX ROWID CUSTOMERS (cr=7 r=3 w=0 time=822 us)
  10            INDEX RANGE SCAN CUSTOMERS_N1 (cr=2 r=1 w=0 time=424 us)
 990          INDEX RANGE SCAN ORDERS_N1 (cr=16 r=3 w=0 time=9810 us)
```

In this example, we see that the Nested Loops join between the CUSTOMERS table and the ORDERS_N1 index took 13,367 microseconds to complete and required 23 Consistent reads and 6 Physical reads. 2 of the Consistent Reads resulted from the Range Scan of the CUSTOMERS_N1 index, 5 from accessing the CUSTOMERS table, and 16 from the Range Scan of the ORDERS_N1 index. The query slowed down significantly when the ORDERS table was next accessed. This step took an additional 9,634,156 microseconds to complete and required an additional 1981 Consistent Reads and 650 Physical Reads. We realize that most of the time required by the SQL query was spent accessing the ORDERS table. This sort of information is invaluable when tuning an SQL query.

Note that many releases of Oracle 9*i* up through Oracle 9.2.0.4 have a bug that causes Oracle to collect the additional information even if STATISTICS_LEVEL is not set to ALL. Many people were therefore puzzled when the additional information disappeared from TKPROF reports when they applied the 9.2.0.6 patch set. ▲

Many Thanks to Our Sponsors

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

Long-term event sponsorship:

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

Thank you! Year 2006 Gold Vendors:

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

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

TREASURER'S REPORT

Diane Lee, *Treasurer*

Beginning Balance March 1, 2006		**$ 43,280.20**
Revenue		
Membership Dues	7,030.00	
Meeting Fees	270.00	
Vendor Receipts	6,400.00	
GAP Grant	3,000.00	
Interest	56.79	
Total Revenue		**$ 16,756.79**
Expenses		
Regional Meeting	54.37	
Journal	7,359.43	
Membership	130.92	
Administration	–	
Website	–	
Board Meeting	619.34	
Marketing	441.86	
Vendors	44.40	
IOUG Rep	625.00	
Total Expenses		**$ 9,275.32**
Ending Balance June 30, 2006		**$ 50,761.67**

Can *You* Say SOA?

by Ravi Kulkarni

Our intrepid database gumshoe trolls the highways and the byways of the World Wide Web, looking for newsworthy items.

Ravi Kulkarni

Time to Upgrade Again

Oracle has released Version 9 of PeopleSoft and moved its customers one step closer to a service-oriented architecture (SOA).[1] The new release of PeopleSoft is integrated with Oracle's Fusion Middleware, which allows applications from different vendors to interoperate. Oracle has promised new product releases for its other high-profile acquisitions too. Siebel CRM will move to the center of things and interoperate with the best parts of Oracle E-Business, PeopleSoft CRM, and JD Edwards CRM. Suddenly Oracle's strategy begins to make sense.

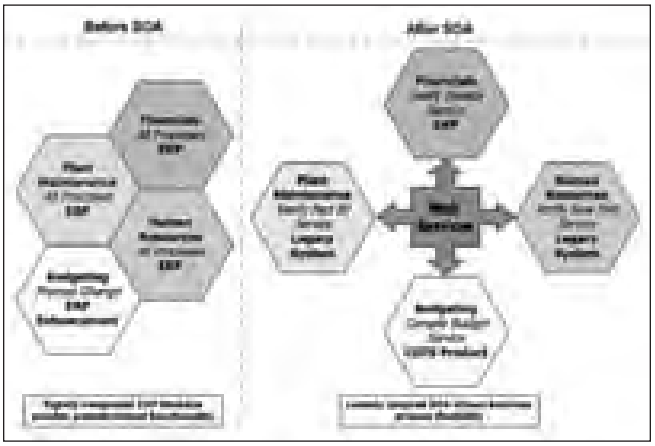
The Sky's the Limit

Oracle has set the new world record TPC-H Three Terabyte benchmark using Oracle 10*g* Release 2 RAC running on an HP blade-cluster with 64 nodes—each with one dual-core AMD Opteron processor and Red Hat Enterprise Linux 4. Is there any limit to what RAC can do?—**www.tpc.org/tpch/results/tpch_perf_results.asp**.

Move Over Bangalore?

Oracle plans to open a Global Support Center (GSC) in Dalian, China. But the call-center titans in Bangalore, India, can rest easy. The only purpose of the Chinese facility is to provide local-language assistance to Oracle customers in China, Taiwan, and Korea; it's always a good idea to speak to your customers in their own language. If you need a refresher course on Chinese geography, the friendly folks at Encarta have maps—go to **encarta.msn.com/map_701511958/Dalian.html**.

[1] See **www.army.mil/escc/erp/soa.htm**. Picture courtesy of the Enterprise Integration Oversight Office of the U.S. Army.



Alphabet Soup

What do the following companies have in common—other than interesting names? 360Commerce, Context Media, Demantra, G-Log, HotSip, i-flex, Innobase, Net4Call, Oblix, OctetString, PeopleSoft, Portal Software, ProfitLogic, Retek, Siebel, Sleepycat, Telephony@Work, TempoSoft, Thor Technologies, TimesTen, and TripleHop. The answer is that all of them have been acquired recently by Oracle or are going to be acquired by Oracle soon. "*By combining with strategic companies, Oracle strengthens its product offerings, accelerates innovation, meets customer demand more rapidly, and expands partner opportunity. An integral part of Oracle's Mergers and Acquisitions philosophy is its consistent commitment to customer service and product support while achieving financial return objectives and creating value for shareholders.*" At least that's how Oracle's PR machine spins it. Drill into the juicy details at **www.oracle.com/corporate/acquisition.html**.

Singing for his Supper

Sir Elton John will sing at the "Customer Appreciation Event" at Oracle OpenWorld 2006 in San Francisco in October. "*He stands for innovation and goodwill, and this is the image we want to portray at Oracle,*" said Judy Sim, Senior Vice President of Worldwide Marketing and Customer Programs. Sir John will bring a special red piano to the event. Will he sing songs about Oracle? A quick check of the new *legal* Napster turns up 60+ songs about Oracle (but none about SQL Server or DB2). Larry should be pleased. See for yourself at **www.napster.com/search/results.html?type=Track&query=oracle**. ▲

Ravi Kulkarni has 20 years of IT experience in roles ranging from programmer to CEO. He is currently an Oracle applications administrator for IBM in Phoenix, AZ. His interests outside work include yoga and early childhood development. His address is **kulkarniravi@yahoo.com**.

Spring Conference Abstracts

	Room 1220	Room 1240	Room 1130
11:00 a.m. to 12:00	**Six Steps to Unit Testing Happiness**—Steven Feuerstein, Quest Software We all know that we should thoroughly test each individual program or unit of code—this is called "unit testing." Yet few of us feel that we do test our code adequately. This presentation will review high-level principles for unit testing and also provide pragmatic tips for improving your test process.	**Oracle 10*g* SQL Tuning Secrets**—Donald Burleson, Burleson Consulting The topics covered include the new Oracle parameters that affect SQL performance, the use of Hints to change SQL execution plans, re-writing SQL queries in more efficient forms, materialized views, replacing SQL with PL/SQL, the new automated CBO statistics collection, and using the new Oracle 10*g* CPU costing approach.	**Introduction to Oracle Spatial Using Public Data**—Richard Flores, Independent Consultant Oracle Spatial can play an important part in the storage and decision support analysis of geographic data for business. Using examples, this presentation will introduce Geographic Information Systems (GIS) and computer cartography, the basics of Oracle Spatial, and the acquisition and use of freely available geographic data.
1:00 p.m. to 2:00 p.m.	**How Much Do Concurrent Updates Impact Query Performance in Oracle?**—Roger Schrag, Database Specialists Oracle has a read-consistency model that sets it apart. You can update rows in a table while I query the same rows. However, this comes at a cost to performance. In this presentation, we will look at TKPROF reports and v$ views, and measure how much slower a query runs when the tables are undergoing concurrent updates.	**Creating a Self-Tuning Oracle 10*g* Database**—Donald Burleson, Burleson Consulting Oracle started to create the mechanisms for a self-tuning database with Oracle 9*i* and has begun the automation of many tuning actions in Oracle 10*g*. This is an indispensable presentation for Oracle professionals who want to know how to automate their manual decision rules within the automation framework of Oracle 10*g*.	**EDITOR'S PICK** **J2EE: Black Box in the Oracle World**—Don Bergal, Confio Software Most Java performance tools lack visibility into the database layer, while most database performance tools lack visibility back into the Java layer. This training session shows how to extend Oracle Wait Time methodology to Java applications and gain the end-to-end visibility required to solve performance problems.
2:15 p.m. to 3:15 p.m.	**Top 36 Wait Events**—Kyle Hailey, Independent Consultant Have you ever wondered about the explanations, causes, and solutions for Oracle wait events? Have you ever been frustrated with the lack of documentation on Oracle wait events? Here is a presentation on the causes and solutions to the 36 most frequently encountered wait events, which account for over 95% of the wait time in Oracle databases.	**SQL Sucks**—Iggy Fernandez, Verizon Database engines are severely handicapped in their ability to find good execution strategies for SQL queries because they assume that all the columns of a table have independent data distributions. Also, they don't know how many rows will be created when two tables are joined. Query Optimization is therefore doomed to failure. We analyze the problem and discuss the workarounds.	**Oracle 10*g*, CoolThreads and Containers**—Michael O'Connor, Sun Microsystems We will explore Sun's latest innovations—including Chip Multi-Threading and Open Source Solaris 10 Containers—and provide best-practice recommendations for safely consolidating multiple Oracle 10*g* database instances onto a single Sun SPARC- or AMD Opteron-based host.
3:45 p.m. to 4:45 p.m.	**Issues and Remedies Surrounding Blocking Locks**—Vilin Roufchaie, DBPulse Group This presentation covers some of the important technical details and decision-making procedures that production managers and DBAs will be faced with—and the many factors they must take into consideration before pulling the plug on a blocking session that could potentially be executing an important application.	**Logical E/R Modeling: The Definition of Truth for Data**—Jeffrey Jacobs, Covad Communications Logical Entity-Relationship models—also referred to as "conceptual" or "semantic" models—define the information requirements of the enterprise, independent of the resulting implementation. A well-defined E/R model is the key to successful development of data-oriented applications. We provide an overview of the fundamentals of E/R modeling.	**Extending OEM Grid Control Functionality with Plug-Ins**—Brian Doyle, BEZ Systems OEM Grid Control R2 provides a facility called the Management Plug-In, where customers can define and deploy their own target types within the product. The bulk of the discussion will be on how to extend the base set of captured data through the creation of Management Plug-Ins that will allow new sets of data to be captured and stored within the OEM repository.

NoCOUG Summer Conference Schedule

August 17, 2006, at Chevron, San Ramon, CA

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

8:00A.M.–9:00	**Registration and Continental Breakfast**—Refreshments served
9:00–9:30	**General Session and Welcome**—Darrin Swan, NoCOUG President
9:30–10:30	**Keynote:** *Software Programmers: Heroines and Heroes of the Twenty-First Century*—Steven Feuerstein, Quest Software
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Room 1220: *Six Steps to Unit Testing Happiness*—Steven Feuerstein, Quest Software
	Room 1240: *Oracle 10g SQL Tuning Secrets*—Donald Burleson, Burleson Consulting
	Room 1130: *Introduction to Oracle Spatial Using Public Data*—Richard Flores, Independent Consultant
12:00–1:00 P.M.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Room 1220: *How Much do Concurrent Updates Impact Query Performance in Oracle*—Roger Schrag, Database Specialists
	Room 1240: *Creating a Self-Tuning Oracle 10g Database*—Donald Burleson, Burleson Consulting
	Room 1130: *J2EE: Black Box in the Oracle World*—Don Bergal, Confio Software
2:00–2:15	**Break**
2:15–3:15	**Parallel Sessions #3**
	Room 1220: *Top 36 Wait Events*—Kyle Hailey, Independent Consultant
	Room 1240: *SQL Sucks!*—Iggy Fernandez, Verizon
	Room 1130: *Oracle 10g, CoolThreads and Containers*—Michael O'Connor, Sun Microsystems
3:15–3:45	**Raffle and Refreshments**
3:45–4:45	**Parallel Sessions #4**
	Room 1220: *Issues and Remedies Surrounding Blocking Locks: Asking the Right Questions and Taking the Appropriate Actions* —Vilin Roufchaie, DBPulse Group
	Room 1240: *Logical E/R Modeling: The Definition of Truth for Data*—Jeffrey Jacobs, Covad Communications
	Room 1130: *Extending the OEM Grid Control Functionality With Plug-Ins*—Brian Doyle, BEZ Systems
5:00–??	**NoCOUG Networking and Happy Hour at Marriott San Ramon, 2600 Bishop Drive, San Ramon**

Session descriptions appear on page 27.

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

Cover photos © Tom Wagner. Mount Timpanogos, Utah.

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED

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