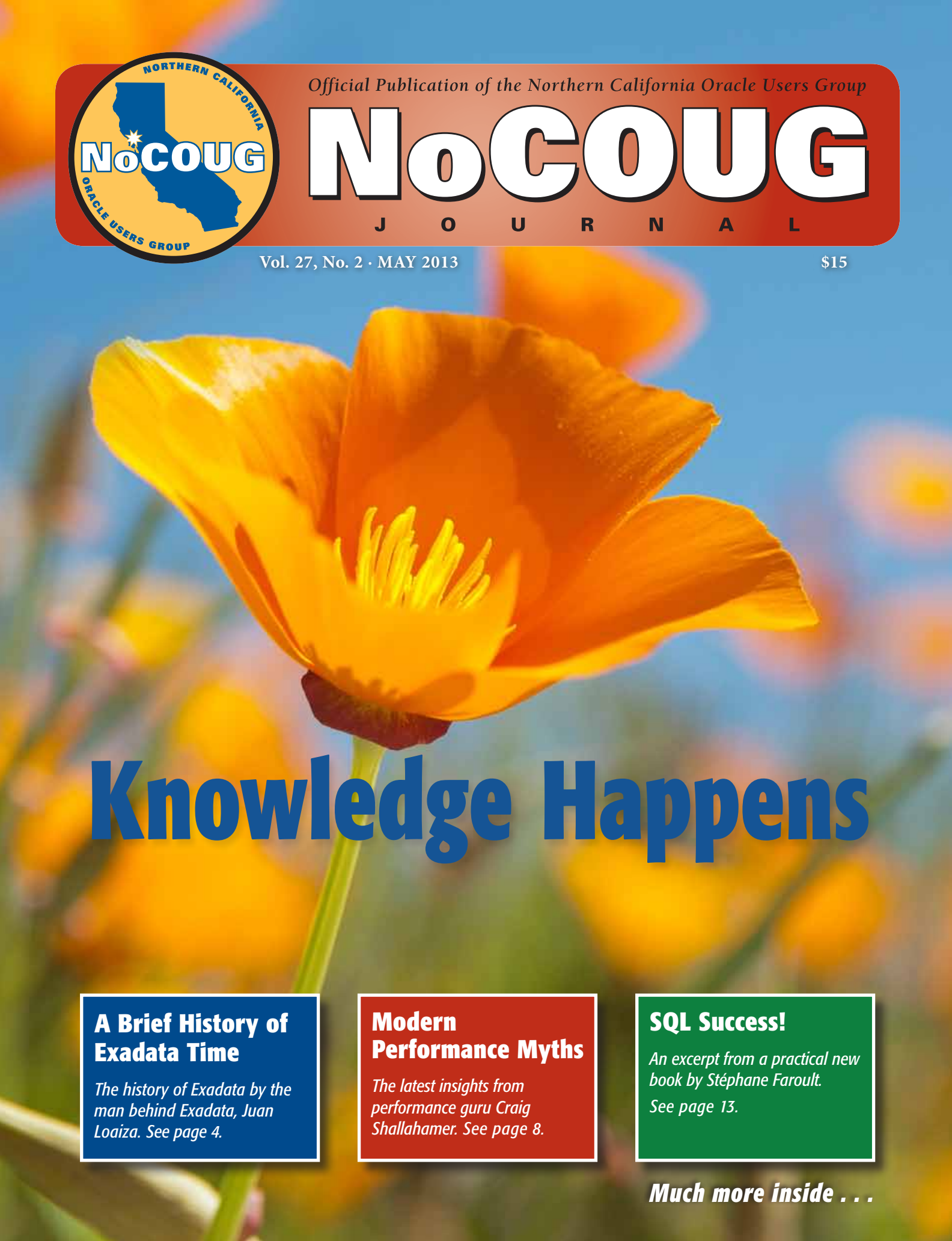
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 27, No. 2 · MAY 2013 $15



Knowledge Happens

A Brief History of Exadata Time

The history of Exadata by the man behind Exadata, Juan Loaiza. *See page 4.*

Modern Performance Myths

The latest insights from performance guru Craig Shallahamer. *See page 8.*

SQL Success!

An excerpt from a practical new book by Stéphane Faroult.

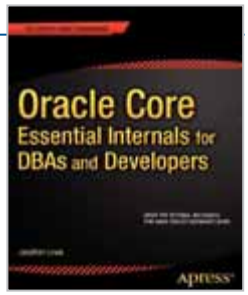
See page 13.

Much more inside . . .

Oracle Core: Essential Internals for DBAs and Developers

A Book Review by Brian Hitchcock

Details



Authors: Jonathan Lewis

ISBN: 978-1-4302-3954-3

Pages: 280

Year of Publication: 2011

Edition: 1

List Price: $39.99

Publisher: Apress

Overall Review: Excellent, concise coverage of Oracle database fundamentals.

Target Audience: Anyone interested in Oracle RDBMS internals.

Would you recommend this book to others: Yes.

Who will get the most from this book? DBAs.

Is this book platform specific: No.

Why did I obtain this book? See overall interview below.

Overall Review

I attended Jonathan's presentation at Oaktable World (also known as Oracle Closed World), and copies of this book were given to all attendees. While I hadn't planned to read a book about Oracle database internals, the topic has always interested me. I was also intrigued because this book is not huge. The typical Oracle book these days looks more like a phone book or a doorstop. Many books claim to cover everything you need to know and are just too long. I was eager to learn the most important concepts about Oracle internals but, at the same time, I didn't have any interest in trying to read a long book.

It has been a long time since I have needed to look at performance issues in the database. With applications getting more and more complex, many performance issues move to the middle tier, well removed from the database. Therefore, I had no immediate need to read this book. But, as so often happens, one door closes and another door opens. The only good approach to this aspect of life in the real world is to move quickly enough to avoid getting squished in the closing door—and to get through the open door before it, too, closes and you are left in the hallway all alone. Before I finished this review, I was reassigned to a new group, and within a week I was asked to look at a performance issue that I found was being caused by ITL waits. This is the universe's way of reminding us to keep an open mind. You don't know what will be useful today, tomorrow, next week—or in your new group!

Many times while I was reading this book, I simply wrote in my notes "I never thought of that!" There is a lot of useful information in this refreshingly short book.

Introduction

I liked the author's explanation for why he wrote this book. He explains that while you can find information about parts and pieces of Oracle, *"What you won't find is a cohesive narrative that puts all the right bits together in the right order to give you a picture of how the whole thing works and why it has to work the way it does."*

I also like this summary of what you will learn as you read this book. To quote the author, *"[I]t boils down to undo, redo, data caching, and shared SQL."* It is refreshing to find an author who doesn't use the complexity of Oracle to confuse the audience. Yes, Oracle is very complicated, but there are basic concepts that you need to know and that you can use to understand the specific parts of Oracle you have to deal with every day.

The author provides links to more detailed sources, so you can learn more if you want to, as well as updates to each chapter online.

Chapter 1—Getting Started

The author explains how he will begin and what will be discussed in what order. First up is a discussion of Oracle processes. There is good detail here. For example, there is a location in SGA that serves as a clock that coordinates the activity in the instance. This is the System Change Number (SCN). I liked calling the SCN a "clock." This made me think about SCN from a different perspective.

Next up is a great summary of what happens when a user submits a query to the database. This is what you really need to keep in mind when working with Oracle. I want to quote this paragraph in its entirety: *"An end user sends requests in the form of SQL (or PL/SQL) statements to a server process; each statement has to be interpreted and executed; the process has to acquire the correct data in a timely fashion; the process may have to change data in a correct and timely fashion; and the instance has to protect the database from corruption."* This concise overview is a good way to tie together everything that will be covered in the following chapters.

Chapter 2—Redo and Undo

We start out with an interesting question: what is the most important bit of Oracle technology? The answer is not something new from marketing. It is the change vector! This is the way Oracle describes the changes to data blocks, and it's what enables both redo and undo.

This is also what allows Oracle to minimize the conflict between readers and writers. This has been hugely important to Oracle marketing over the years: writers don't block readers … or is it the other way around, or both?

The way Oracle approaches changing data is not what you think. It is reasonable to expect the process to find the data and then change it. The four-step process is described that will create the following:

First, a description of how to make the change (redo change vector). Then, a description of how to reverse the change (undo record). Next, a description of how to create the description of how to reverse the change and, finally, change the data (redo change vector).

If this sounds confusing, I encourage you to read the full text of this discussion. It very much changed my understanding and appreciation of undo and redo.

A detailed example is given that includes symbolic dumps of the data block as the process of changing data progresses through all four steps. I'm not much good with symbolic dumps, but it's all here for those that like that sort of thing.

Another example of things in this book that I had not thought about before is that Oracle keeps two copies of everything: one copy in the data files and another copy in the redo log files. The figure provided in this section really helped me see this.

ACID properties (Atomicity, Consistency, Isolation, Durability) are discussed in detail. Why the undo mechanism enables concurrency (readers and writers not blocking each other) is explained. The generation of redo is a very simple mechanism in that, basically, you "write and forget." A feature of redo that I'd never heard about before is (starting in 10*g*) private redo and in-memory undo. Until a transaction completes, all change vectors are stored in private redo, and only at commit does the transaction need the redo log allocation latch to write redo.

The details of the complexity of the rollback process are presented. This has bothered me for years. Why does it take so long to roll back a transaction? It turns out that more buffer visits on undo blocks are needed when rolling back than when the transaction was initially executed. Now I have an answer for my users when rolling back takes a long time.

Chapter 3—Transactions and Consistency

The changes made by different users must be kept separate until their transactions commit, and the database must be able to quickly change which changes are visible to which users.

The reason rollback can cause long db startup time (after shutdown abort for example) is further explained: *"Since rolling back real changes is (or ought to be) a rare event compared to committing them, Oracle is engineered to make the commit as fast as possible and allows the rollback mechanism to be much slower."* Think about it … this tells us that rolling back is not what Oracle is built to do fast. If you roll back a long-running transaction, don't be upset that it takes a long time. This explains why rollback is so slow.

Another detail about rollback that I had never thought about is presented as follows: *"It isn't commonly realized, by the way, that when Oracle has applied all the relevant undo records, the last thing it does is update the transaction table slot to show that the transaction is complete—in other words, it commits."* Even rolling back requires a commit. It makes sense once it is explained. These are very interesting details of rollback, stuff I don't think you'll find anywhere else. I also learned how a single undo block can contain undo records for multiple transactions. This happens when a transaction commits and the free space left in the last UNDO block can be assigned to another transaction.

The Interested Transaction List (ITL) is explained. The ITL exists to identify transactions that recently changed a data block. An interesting bit of Oracle trivia is that a value of c1 02 is Oracle's internal representation of decimal 1. LOL!

Another new concept: Consistent doesn't mean historic. While discussing consistency, a detailed example shows how we sometimes construct a version of the block that has never actually existed! This is necessary to support read consistency. Also described is how to compute the work done to maintain read consistency. It turns out that Flashback query does all the work of read consistency and a lot more of it.

In the discussion of Commit SCN, there is this interesting detail: Sometimes Oracle shares the work among multiple sessions. For example, if a large number of blocks need to be cleaned up, this would take a long time for a single session to complete. Some of this work will be done by each of the next bunch of sessions, even though these sessions may not have caused these blocks to need cleanup.

In the section covering Commit Cleanout, it comes out that checkpoint makes Oracle copy any dirty blocks to disk, but it doesn't make Oracle remove them from the buffer cache.

In the section on ORA-1555, we have this comment: *"If you don't know that Oracle error 1555 translates into "snapshot too old" you can't be a real DBA."* This caught my attention because when I first started working on Oracle Applications, I was advised to stay away from that because "it isn't real DBA work"—and now I've worked with Fusion Applications, so I'm really far removed from being a "real" DBA!

Finally, in a section on LOBs, something that has always confused me is explained. There are special methods for handling undo and redo on LOBs. The discussion is very good, and I recommend that you read this part. Also, the summary is great: *"Essentially, Oracle doesn't update LOBs."* I have always wondered about this.

Chapter 4—Locks and Latches

This chapter provides a very clear and concise explanation of locks and latches. Locks are polite and tend to be held for long time. Latches are pushy and should be held very briefly. See? That was clear and concise! Pins and mutexes are also covered. Pointers, linked lists, and hash tables are explained to support the discussion of locks and latches. There is a good explanation of what could happen to linked lists if we didn't

have locks and latches. Despite the often-quoted Oracle marketing line that "readers don't block writers" and the other way around, the author tells us, *"[W]hen you get down to the raw memory level, there are moments when readers must block writers, and where a single write must block all other operations."* Wow! The things you can think!

I found the explanation of what a latch is to be fascinating. *"Essentially a latch is the combination of a memory location in the SGA and an atomic CPU operation that can be used to check and change the value of that location."* An atomic CPU operation is explained as a single CPU instruction that can both check and change a location in memory. Further, multiple CPUs mean that any set of operations can be interrupted, hence the need for a single CPU operation that is "atomic." I've never had the latch and the CPU operations explained like this.

The detailed explanation of how this would all break down if we didn't have an atomic CPU operation is great. The drawbacks of exclusive latches and how Oracle deals with this from 9*i* is another "wow" moment, as are the examples of what Oracle writes in memory location for a latch.

It's interesting how things change over time . . . or don't. A specific example of this is the choice of units for wait event times. Measuring wait event time in 1/100th of a second started with Oracle 6, when CPUs that ran a few MHz were "fast."

The description of how the wakeup mechanism works (instead of just "sleeping") is great. Specific advice on how to prevent long wait time for latches is provided. The explanation of the difference between a lock and an enqueue provided insight into something that has always confused me.

Based on the author's experience, the truth about deadlocks—versus the official story that one process is chosen at random to be killed—is that the longest waiting session gets killed.

Most useful to me is the message that despite all the talk about deadlocks, there is no perfect solution for handling deadlocks. If I had a nickel for every time I had to explain to a user that there wasn't anything I could do about the deadlocks that simply happen in their code, I'd be retired somewhere far, far away from Redwood Shores. I never thought before about one more aspect of deadlocks—that they are not limited to two sessions.

Chapter 5—Caches and Copies

I greatly appreciate the specific advice the author provides, including this example: *"Personally I prefer to see 8KB as the size for the default as this is the size that sees most testing in the field and is generally the option least likely to result in odd problems."*

Good advice: keep it simple! Another example regarding db_writers_processes: *"[C]ontrary to frequent comments on the Internet, this is a parameter that rarely needs to be adjusted on a production system."* So much time is spent (wasted?) by "experts" endlessly debating the merits of configuration changes that, in reality, have little or no effect at all.

Chapter 6—Writing and Recovery

I was confident that I understood the recovery process, but I learned some new things in this chapter. The database writer will not write a changed block before the log writer has written the redo that describes the changes. This write-ahead logging is critical to the recovery mechanism. The redo logs (online and archived) are the definitive version of the database. And the part I hadn't thought about before is that the datafiles are just a recent, approximate snapshot of the database. I had not seen such a detailed discussion of what triggers the log writer to start writing.

Now we have one of the very few times I was not completely happy with this book. I read the following statement: *"PL/SQL doesn't always wait for the write to complete."* Really? I want to (need to?) believe that the redo is written before anything else happens. I want to hear more about this. Isn't this a big concern? This is followed by a description of an ACID anomaly. Again, I would think the issue the author describes would be a big deal, but the author doesn't seem upset, although I remain concerned.

I had never seen anything about how Oracle "wastes" redo space in certain circumstances. The author explains that when writing the redo log to disk, Oracle "never looks back" to keep the code as simple as possible, and this explains why, sometimes, Oracle doesn't make full use of all the available redo log space. The discussion of the database writer versus the log writer includes what happens when a block has changed so recently that the redo log for this latest change hasn't been written. One new (to me) aspect of checkpoints is that a query can cause a checkpoint. I didn't know that.

Related to redo logs, the author expresses concern over subtle changes that may have been made to support standby databases. I found this to be very interesting. While discussing Flashback Database, the author points out, *"The opportunities offered by inventive use of the redo logs are wonderful."* I agree.

Chapter 7—Parsing and Optimizing

This chapter starts with interesting trivia. The dictionary cache is also called the "row cache" because, in the past, it used to cache individual rows of data from the data dictionary instead of data blocks.

I had never heard of bootstrap objects. These are how Oracle knows how to find the first things it needs to know from the database to find out about everything else in the database. Various aspects of parsing are discussed, including an explanation of what a parse call is and the sequence of activity for parsing. Also presented are how cursor caching is done and issues around closing cursors.

Here is an amusing item from the discussion of the details of the extents of the shared pool: *"[E]ach freeable chunk will be linked to a recreatable chunk and the owner of the recreatable chunk is responsible for freeing the linked freeable chunks when it frees the recreatable chunk."* Another wow!

Chapter 8—RAC and Ruin

First, I am simply quoting the chapter title. Any association between RAC and the word "ruin" is strictly the author's choice. The author explains: *"I felt the need for a little alliteration in the chapter title, and it is very easy to ruin things if you don't understand a little bit about how RAC works; but RAC*

(continued on page 24)

such checking will slow these operations down. *[emphasis added] If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently. Inputs causing the inconsistencies which remain in the data bank state at checking time can be tracked down if the system maintains a journal of all state-changing transactions.*"

This approach can be dubbed "eventual consistency" and is used by NoSQL systems; that is, distributed replicas may not be in sync. It is ironic that a principle elucidated by Dr. Codd in the first paper on relational theory should have found its first application in systems that *reject* relational theory. ▲

Answers to quiz questions: 1 (a) Oracle Corporation; 2 (a) IMS; 3 (a) No DBMS was used.

BOOK EXCERPT *(continued from page 18)*

then the name that will be returned will change. The problem is that indexed values are stored in the index. If some settings are modified and, all of a sudden, applying a function to a column no longer returns the same value as the pre-calculated value that was stored in the index, it becomes impossible to retrieve the data. If you could index the function that returns the name of a month (SQL Server will prevent you from doing so) you would store in the index key values that a different language setting would make irrelevant. Clearly, this isn't acceptable: the purpose of indexes is to provide an answer faster, not to change the answer by saying *no data found* (or perhaps, more to the point, something such as *¡No se ha encontrado ningun dato!*) when you search for Enero in an index that was built when the current language was English and refers to a lot of rows for *January*.

Note: *Enero* is Spanish for *January*.

You can be on shifting ground even with date functions that return numerical values, which you might believe to be language neutral. For instance, the number of the day in a week is counted differently in different countries. If you ship to several countries a program that uses the number of the day of the week, you cannot guarantee that the behavior will be the same everywhere. ▲

Stéphane Faroult first discovered relational databases and the SQL language back in 1983. He joined Oracle France in their early days (after a brief spell with IBM and a bout of teaching at the University of Ottawa) and soon developed an interest in performance and tuning topics. After leaving Oracle in 1988, he briefly tried to reform and did a bit of operational research, but after one year, he succumbed again to relational databases. He has been continuously performing database consultancy since then, and founded RoughSea Ltd in 1998. He is the author of The Art of SQL *[O'Reilly, 2006] and* Refactoring SQL Applications *[O'Reilly, 2008].*

BOOK REVIEW *(continued from page 21)*

doesn't necessarily lead to ruin." In this chapter, the author focuses on what we need to understand about RAC, which he tells us is global enqueues and cache coherency.

I was pleased to read that the difficult part of RAC is getting it installed and running in the first place. There is a great diagram and list of key points describing RAC. The explanation of virtual IP addresses was very good.

After this introduction to RAC, the big question about RAC is put forward: *"But it's complicated and why would you want to deal with something complicated?"* We are told to remember that complexity also means more people, more downtime, etc. With this background, the author explains the benefits of RAC and provides good insights, one of which is the following: *"[T]he rate at which an instance can handle redo generation is the ultimate bottleneck in an Oracle system."* One side effect of a big (RAC) system is that it has so much overcapacity that it hides performance issues until it's too late.

I've read a lot of things about the overhead of RAC so I was pleased with the clarity I found here. For RAC, once you get to three instances, the level of overhead doesn't get any worse.

The Global Resource Directory (GRD) and cache fusion are explained, and a diagram explaining how more nodes mean more interconnect traffic is presented. A good discussion covers details of how RAC handles sequences and caches them. This includes issues around sequences, auditing, and scalability.

This chapter ends with a very important summary of RAC and the warning that moving to RAC can cause the performance of badly designed applications to become worse.

Appendix—Dumping and Debugging

Here the author explains how he used Oradebug to investigate and demonstrate the inner workings of Oracle.

Conclusion

I appreciate the author's brevity. He quickly gets to the point. There is always more detail, but the crucial concepts can be illustrated quickly and efficiently, as they are in this book. While reading, there were many times when I wrote in the margins "I never thought of that," "I've always wondered about that," and "Wow!" I highly recommend that you read this book, no matter what your level of Oracle expertise may be. ▲

Brian Hitchcock worked for Sun Microsystems for 15 years supporting Oracle databases and Oracle Applications. Since Oracle acquired Sun, he has been with Oracle supporting the On Demand refresh group and, most recently, the Federal On Demand DBA group. All of his book reviews and presentations—and his contact information—are available at **www.brianhitchcock.net**. ***The statements and opinions expressed here are the author's and do not necessarily represent those of Oracle Corporation.***