



Official Publication of the Northern California Oracle Users Group

NoCOUG

J O U R N A L

Vol. 27, No. 3 • AUGUST 2013

\$15

Knowledge Happens

No! to SQL and No! to NoSQL!

Straight talk from Chris Date and Hugh Darwen.

See page 4.

CREATE ASSERTION: Impossible Dream?

Toon Koppelaars is such a dreamer.

See page 13.

Wielding the Sword of Analytics

Guest columnist Jonathan Gennick explains window functions.

See page 21.

Much more inside . . .



DBMoto® Data Replication and Change Data Capture for Oracle

- Easiest solution available for **Real-time Data Updates** to/from Oracle databases
- Immediate, fresh data for BI, analytics and reporting
- Change Data Capture for non-intrusive data updates
- Supports long list of databases: Oracle, IBM, Informix, Sybase, SQL Server, others
- Updates data in high-speed analytic systems: SAP HANA, HP Vertica, Actian Vectorwise, IBM Netezza
- **FREE EVALUATION** at info.hitsw.com/NoCOUG7



FREE PAPER:

"Top 5 Reasons to use DBMoto's Change Data Capture for Oracle"
<http://info.hitsw.com/NoCOUG7>

T +1.408.345.4001

| www.hitsw.com

| info@hitsw.com



Copyright © 2013 HIT Software, Inc., A BackOffice Associates, LLC Company. All rights reserved. HIT Software®, HIT Software logo, and DBMoto® are either trademarks or registered trademarks of HIT Software and BackOffice Associates, LLC in the United States and other countries. All other trademarks are the property of their respective owners.

Professionals at Work

First there are the IT professionals who write for the *Journal*. A very special mention goes to Brian Hitchcock, who has written dozens of book reviews over a 12-year period. The professional pictures on the front cover are supplied by Photos.com.

Next, the *Journal* is professionally copyedited and proofread by veteran copy-editor Karen Mead of Creative Solutions. Karen polishes phrasing and calls out misused words (such as “reminiscences” instead of “reminisces”). She dots every i, crosses every t, checks every quote, and verifies every URL.

Then, the *Journal* is expertly designed by graphics duo Kenneth Lockerbie and Richard Repas of San Francisco-based Giraffex.

And, finally, Jo Dziubek at Andover Printing Services deftly brings the *Journal* to life on an HP Indigo digital printer.

This is the 107th issue of the *NoCOUG Journal*. Enjoy! ▲

—NoCOUG Journal Editor

Table of Contents

Interview	4	ADVERTISERS	
President's Message	7	HiT Software	2
Special Feature.....	13	WHIPTAIL Storage.....	12
Book Review	16	Embarcadero Technologies	25
Product Review	19	Quilogy Services.....	25
SQL Corner	21	Confio Software	26
Conference Agenda.....	28	Delphix	26
		Database Specialists	27

Publication Notices and Submission Format

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

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

The submission deadline for each issue is eight weeks prior to the quarterly conference. Article submissions should be made in Microsoft Word format via email.

Copyright © by the Northern California Oracle Users Group except where otherwise indicated.

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

2013 NoCOUG Board

President

Naren Nagtode

Vice President

Hanan Hit

Secretary/Treasurer

Dharmendra (DK) Rai

Membership Director

Abbulu Dulapalli

Conference Director

Gwen Shapira

Conference Track Leaders

Nasreen Aminifard

Vendor Coordinator

Omar Anwar

Training Director

Randy Samberg

Meetup Coordinator

Gwen Shapira

Webmaster

Eric Hutchinson

Jimmy Brock

Journal Editor

Iggy Fernandez

Marketing Director

Ganesh Sankar Balabharathi

IOUG Liaison

Kyle Hailey

Board Member at Large

Ben Prusinski

Book Reviewer

Brian Hitchcock

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



Chris Date

No! to SQL! No! to NoSQL!

with Chris Date and Hugh Darwen



Hugh Darwen

“There are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies.”

—The Emperor’s Old Clothes,
1980 Turing Award lecture by C.A.R. Hoare

It was more than 40 years ago that Edgar “Ted” Codd invented the relational model. In that period, the DBMS market has come to be dominated by the DBMS vendors—such as Oracle, DB2, SQL Server, MySQL, and PostgreSQL—who adopted Codd’s visionary ideas. But a small band of relational purists have been complaining that the DBMS vendors have deviated from the teachings of Codd. Two members of that band are Chris Date and Hugh Darwen. Chris Date is the author of *An Introduction to Database Systems*, which has sold some 900,000 copies since its first edition in 1975, and the author and co-author of many other books, too numerous to list here. Hugh Darwen, working for IBM in the UK, was an active participant in the development of the ISO SQL standard from 1988 until his retirement from that company in 2004. He has co-authored a number of books with Chris Date.

No to SQL!

Most database practitioners would be surprised to hear your claims that the major database management systems of today are not fully relational. The logical first question is: what is the yardstick of “relationality”? I know that the creator of the relational model, Edgar Codd, wrote an article in *Computerworld* magazine in 1985 titled “Is Your DBMS Really Relational?” in which he said: “In this paper, I supply a set of [twelve] rules with which a DBMS should comply if it is claimed to be fully relational. No existing DBMS product that I know of can honestly claim to be fully relational, at this time.” But, only five years later, he raised the bar higher and went on to list 333 rules—a suspiciously neat and tidy number—in his 1990 book *The Relational Model for Database Management: Version 2*. Then there are the standards produced by the ISO committee on which Mr. Darwen was the IBM representative in the UK delegation from 1988 until 2004. The 2011 version of the ISO SQL standard has almost 4000 pages. Is it fair to keep moving the target from twelve conformance rules to 333 and then to 4000 pages of specifications? Isn’t an RDBMS “relational enough” if it abides by Codd’s original Twelve Rules, especially since he originally considered such products to be fully relational?

“[Codd’s Twelve Rules] weren’t all independent of one another, some of them didn’t make sense, some were unachievable, and some I think were simply wrong.”

Chris: To be relational (even “fully” relational) is only a *minimum* requirement for a DBMS—the relational model addresses only what might be called core DBMS functionality. With respect to that core, the model serves as an abstract recipe for what the user interface should look like. And what that recipe says is basically two things: 1. The data should be presented to the user as relations (and nothing but relations). 2. The operators available to the user should be ones that operate on relations (and that set of operators should be “relationally complete”). SQL fails on both counts. First, its basic data object is the SQL table, and there are at least seven points of difference (real logical difference, I mean) between SQL tables in general and relations. Second, its operators are, by definition, operators that work on SQL tables, not relations.

Now, the relational model isn’t a totally static thing—it has evolved over time—so critics who accuse us of moving the goalposts do have a point, in a way. But the criticism is academic, because none of the mainstream SQL products supports even the original (1969) version of the model! And in any case, the changes that have occurred since then are, as I said, evolutionary; the model does grow, but it doesn’t gyrate. It resembles mathematics in that respect; in fact, it’s fair to say the model is itself a tiny piece of mathematics, in a way.

A word on Codd’s Twelve Rules: I’m sorry to have to say this, but it’s my honest opinion that the Twelve Rules paper simply wasn’t up to the standard of Ted’s earlier relational writings. In fact, I criticized it at the time.¹ This isn’t the place to go into details; suffice it to say the rules weren’t all independent of one

¹ C. J. Date, “Notes Toward a Reconstituted Definition of the Relational Model Version 1 (RM/V1),” in C. J. Date and Hugh Darwen, *Relational Database Writings 1989–1991*. Addison-Wesley, 1992.

another, some of them didn't make sense, some were unachievable, and some I think were simply wrong. Given this state of affairs, I really think it's a pity so many people pay attention to those rules (or say they do, at any rate).

As for RM/V2, I'm afraid the same applies, only more so.

What about SQL's 4000 pages? Well, I'll say one small thing in SQL's defense here. To repeat, the relational model as such is only a minimum requirement. You can define that requirement in just a few pages. But SQL, rightly or wrongly, is trying to deal with a lot more than that minimum (for example, it has all of those OLAP functions). So that's one reason why the specification is so big. That said, however, I do also believe that if you could somehow carve out just the pages that deal with SQL's "relational" functionality—such as it is—you'd still have several hundred pages on your hands, because SQL is, quite frankly, a horrendously and unnecessarily complicated language.

The net of all this is: To be fully relational, it's necessary and sufficient that you support the relational model. And *The Third Manifesto* is our attempt to spell out in detail exactly what that means (see the website www.thethirdmanifesto.com).

Hugh: As far as we're concerned, the yardstick of "relationality" has to be *The Third Manifesto*, which first appeared, as a paper in a journal, in 1994 but has been subject to minor revisions on several subsequent republications. The most up-to-date version is available at our website, www.thethirdmanifesto.com. It's a detailed prescription to be followed in the design of a relational database language, and the few prototype implementations of it that have become available indicate to us that it does succeed in serving that purpose. Its first section consists of 26 numbered points, referred to as RM Prescriptions, where RM stands for Relational Model, of course. These 26 can be taken as our yardstick as such. The second section, RM Proscriptions, mentions some violations of the RM Prescriptions commonly found in attempts to implement the relational model, mostly in SQL. Other sections are devoted to (a) database issues that have nothing to do with the relational model and (b) strong suggestions that for one reason or another we can't regard as *sine qua nons*. The *Manifesto* is consistent with Codd's 1969–70 definition of the relational model, but it clarifies several points that, because he didn't make them very clearly himself, have resulted in a certain amount of confusion over the years.

Sadly, Codd's 1985 observation concerning the nonexistence of "fully relational" DBMS products still holds good in 2013 as far as the commercial scene is concerned. However, at least we do now have a few prototype implementations of our *Manifesto*, made by individual enthusiasts. These are available as free downloads and can be thought of as setting the bar for putative commercial implementations.

As for Codd's famous "Twelve Rules" of 1985 (actually there were thirteen, because the numbering started at zero): Well, they certainly excluded quite a few important points and overemphasized others. Also, they omitted much essential detail, and they weren't expressed very precisely. (Contrariwise, his 1991 book suffered from overkill in our view, as well as including many features—Codd's term—with which we firmly disagree.) So is a DBMS "relational enough" if it abides by those rules? No, it isn't. For one thing, they aren't detailed enough. For instance—and this is just one small but extremely important example—they don't make it clear, and nor did his original papers, that the at-

tributes of a relation must be distinguished by *name*. In fact, there's no definition of the term *relation* in those rules, so when we criticize SQL for allowing two or more columns of a table to have the same name, for example (or for allowing the same row to appear several times in the same table, for another example), we can't merely cite the Twelve Rules to justify our criticism.

For our second point, there's at least one of those rules that we can't accept, and that's Rule 3, *Systematic treatment of null values*;

"The relational model is a little piece of mathematics, while performance and scalability are engineering issues."

in fact, we categorically reject it. Our reasons for doing so have been given over and over again, in numerous publications, over the past 30 years or so (and I think Chris will have a little more to say in a moment in this connection). Most recently, I included a comprehensive treatment of the effects of nulls on SQL operations in my book, *SQL: A Comparative Survey*, available as a free download from bookboon.com. The exercise, possibly the first of its kind, was salutary. I can now point to my findings in that book when somebody accuses me of exaggerating when I complain about the ad hoc-ery that null gives rise to in SQL and the inconsistency with which it's treated from operation to operation.

The logical second question is: what benefits will a fully relational DBMS give to users? The need of the hour is performance and scalability to manage the explosion of data in a wired world. Will fully relational database management systems deliver the performance and scalability that today's users are clamoring for? And the logical third question is: why are DBMS vendors not exactly rushing to make their products fully relational?

Chris: I think we need to disentangle a few issues here. First of all, your line of questioning tends to suggest that if those relational benefits don't include performance and scalability, then they're irrelevant. Forgive me, but that's like suggesting mathematics is irrelevant because it's not engineering. In fact, as I've already said, the relational model is a little piece of mathematics, while performance and scalability are engineering issues. Let me elaborate.

It's well known—or, at least, it should be well known—that the relational model is silent on everything to do with matters of physical implementation. That silence is deliberate, of course. The idea was to give implementers the freedom to implement the model however they saw fit (in particular, in whatever way seemed likely to yield good performance) and not to constrain them unnecessarily. Thus, performance and scalability (and other similar issues) are all matters for the implementation—they have nothing to do with whether or not the DBMS is relational. Well . . . that said, let me now add that:

- a. To the extent nonrelational systems achieve performance and scalability, they do so, at least in part, by muddying the distinction between model and implementation—in effect, by exposing part of the implementation to the user, thereby undermining the important goal of data independence.

- b. Precisely because the relational model doesn't unnecessarily constrain the implementer, there's actually good reason to believe a relational DBMS should be able to do better than nonrelational systems on performance and related matters. In this connection, I'd like to mention Steve Tarin's work on The TransRelational™ Model, which is a radically novel and highly promising implementation technology for relational DBMSs.² (When I say it's radically novel, I mean it's quite dramatically different from all of today's mainstream SQL implementations.)

“Why are DBMS vendors not rushing to make their products fully relational? Obviously, because customers aren't asking them to.”

So what are (or would be) the benefits of a truly relational DBMS? Well, I've already mentioned data independence (which, as I've explained in many places, translates into *protecting user investment*). Another is simplicity—something that most certainly can't be claimed for SQL systems (I can show you examples of queries that take many lines of SQL code but can be expressed in one short line in a well designed relational language). And a third is, of course, the strong theoretical foundation the relational model provides. You wouldn't tolerate flying in a plane that wasn't built in accordance with the principles of physics and aerodynamics. You wouldn't tolerate living or working in a high rise building that wasn't built in accordance with sound architectural principles. Why would you tolerate using a DBMS that hasn't been built in accordance with solid database principles? In other words, I don't think people should ask “What are the benefits of being relational?” Rather, I think they should ask—or perhaps try to explain—what the benefits are of *not* being relational.

Finally: Why are DBMS vendors not rushing to make their products fully relational? Obviously, because customers aren't asking them to. And why aren't they asking? Because of the failure, I suppose, of relational advocates like ourselves to educate those customers adequately. But it's not for want of trying, I can assure you. Believe me, we find the situation extremely frustrating.

Hugh: What benefits would a fully relational DBMS give to users? Well, let me start by citing the objectives that Codd himself gave for his relational model in his 1974 invited paper “Recent Investigations into Relational Database Systems”:

1. To provide a high degree of data independence
2. To provide a community view of the data of spartan simplicity, so that a wide variety of users in an enterprise, ranging from the most computer naïve to the most computer sophisticated, can interact with a common model (while not prohibiting superimposed views for specialized purposes)
3. To simplify the potentially formidable job of the DBA
4. To introduce a theoretical foundation, albeit modest, into database management (a field sadly lacking in solid principles and guidelines)
5. To merge the fact retrieval and file management fields in preparation for the addition at a later time of inferential services in the commercial world
6. To lift database application programming to a new level—a level at which sets (and, more specifically, relations) are treated as operands instead of being processed element by element

However, there was no recognized distinction, in 1974, between “fully relational” and “not fully relational” systems. So we might add:

7. Not to compromise these objectives by failing to adhere strictly to the foundation defined to meet them
8. Not to mess things up even further by violating generally accepted principles of good language design

My book *SQL: A Comparative Survey* (see above) systematically compares SQL feature by feature with a language, **Tutorial D**, that has been designed to meet all eight of the above objectives (especially the last two).

In pursuit of these aims, the relational model as defined by *The Third Manifesto* offers the following significant features, for example:

- *Physical data independence*: the user interface excludes all reference to the way data is physically encoded in recording media
- *Uniformity of representation* of data at the user interface (in the form of relations) and concomitant uniformity in the way data is accessed
- *Completeness* of the set of operators for deriving relations from relations, for purposes such as expressing queries and defining integrity constraints
- *Absence of pointers* in the user interface—pointers effectively mean the IT department has to intervene between the database and its real (i.e., end) users, because, as Codd himself said in this connection, fewer people understand pointers than understand simple value comparisons
- *Obviation of the need for loops and branching* in application programs when accessing the database—these, like pointers, being a frequent cause of errors, especially when they're nested

I understand that a number of smaller players are attempting to build fully relational products. The Ingres Project D effort sounds particularly interesting because Ingres is already an enterprise-level DBMS with all the bells and whistles needed by the enterprise. What is the current status of Ingres Project D?

Hugh: You're referring to the few prototype implementations of our *Manifesto* made by individual enthusiasts that I mentioned in my answer to your first question. These are listed as Projects at our website, www.thethirdmanifesto.com. Unfortunately
(continued on page 8)

² Date, C. J. *Go Faster! The TransRelational™ Approach to DBMS Implementation*. Bookboon.com Ltd.: www.bookboon.com (free download).

Winning!

by Naren Nagtode



Naren Nagtode

Stephen Covey wrote what is perhaps the most important self-help book of all time: *The Seven Habits of Highly Effective People*. The habit that governs the rest is “Begin with the End in Mind.” Imagine your retirement party at the end of your career. Imagine the speakers telling the group about you. What do you want them to say? What words do you want them to use to describe you? Professional? Expert? Teacher? Thinker? Doer? Start today with those ends in mind. TV personality Dr. Phil says pretty much the same thing in *Life Code: The New Rules for Winning in the Real World*. Rule #1 of his “Sweet Sixteen” rules for success is “You must have a defined ‘image’ and never go out of character.” Here’s a good example, by one of the authors who is featured in this *Journal* issue: “Writer, Book Editor, Oracle DBA, SQL and PL/SQL Developer, Father, Husband, Son, Mountain Biker, and EMT.”

But how do you achieve your ends? How do you become a winner? One of the seven habits noted by Covey is “Sharpen the Saw.” In your professional life, this means continuously improving and enhancing your skill set. And that’s where NoCOUG comes into the picture.

For more than 25 years, NoCOUG has helped Oracle professionals like you be winners through our conferences and

Journal. At the summer conference on Thursday, August 15, at Chevron in San Ramon, NoCOUG members will be treated to a wide-ranging survey of the relational and non-relational worlds by Iggy Fernandez, whose keynote topic is “Soul-Searching for the Relational Movement: Why NoSQL and Big Data Have Momentum.” Iggy believes that the perceived deficiencies of relational technology are actually a result of deliberate choices made by the relational movement in its early years, and that NoSQL and Big Data technologies would not have gained popularity if they did not excel at certain new problems that relational implementations cannot handle well. He suggests that the relational movement needs to do some serious soul-searching instead of pretending that the new problems do not exist.

Other speakers at the conference will cover the highlights of Oracle Database 12c, which has just been released. There will be all of this and much more: the complete agenda is printed on the back cover of this *Journal*.

The conference and *Journal* are made possible through the efforts of the dedicated NoCOUG volunteers. They’re all winners too!

I’ll see you on August 15. ▲



Raffle winners at the spring conference in May. The raffle had more than \$2,000 worth of Oracle Press books, Oracle Press teddy bears, and other prizes donated by exhibitors. George Wang from Amdocs won the GoPro Action Camera donated by HiT Software.

INTERVIEW (continued from page 6)

Ingres Project D was abandoned unfinished in 2011, for a number of unrelated reasons that had nothing to do with the *Manifesto* as such. Until you asked this question, we had omitted to update the information at the website. That omission has now been rectified, but we've retained the description of the project's aims, for interest's sake.

By the way, Ingres Project D isn't exactly the only one inspired by our *Manifesto* that you would describe as enterprise level. There's also Alphora's product, Dataphor (see www.alphora.com), which was the first commercial implementation to come to our

“SQL is widely used, and obviously that situation isn't going to change any time soon. So what we have to do is try to educate SQL users to use the language wisely and well.”

attention. Dataphor is built as a relational front end to SQL systems. The initial aim was for its language, D4, to be in full conformance with the *Manifesto*, but unfortunately its developers eventually had to compromise over support for SQL's nulls. (It was fully compliant in Version 1, but that meant that existing SQL databases were incompatible with it, and so they had to capitulate to user demand.)

You are on record as vigorously opposing nulls and duplicate rows in SQL. On the other side of the debate, of course, is Don Chamberlin, the co-inventor of SQL. In *A Complete Guide to DB2 Universal Database*, he says that “model the real world” and “trust the user” were guiding principles of SQL. He also points out that systematic treatment of null values was one of Codd's requirements, and, in fact, Rule 3 of Codd's set of 12 rules states: “Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.” Chamberlin concludes his defense by saying: “In the end, I believe that the true arbiters of the null and duplicate-row issues will be users of database systems. If users find that these concepts are helpful in solving real problems, they will continue to use them. If on the other hand, users are convinced that nulls and duplicates are harmful, they will avoid these features and scrupulously use options such as NOT NULL, PRIMARY KEY, and SELECT DISTINCT. By supporting these options, DB2 makes it easy for users to ‘vote with their data.’” Your latest books suggest that you have become resigned to the fact that SQL—for all its flaws—is widely used and that, if one cannot avoid using it, then one should use it correctly. Please tell us about your recent books and your reluctant rapprochement with SQL.

Chris: A strong and sufficient reason for opposing nulls and duplicates is simply that—Codd's Rule 3 notwithstanding—they represent a clear and major violation of relational principles, with all that such violation implies. But if you want a more “practical” reason, then how about the fact that they lead to wrong answers? (Wrong answers to queries, I mean.) Space doesn't

permit a detailed explanation of this point here, but in any case we've gone into great depth on this issue, and related issues, in many other places.

As for Chamberlin's arguments: Well, Chamberlin is also on record as saying “I recognize that nulls and duplicates are religious topics.” We think the issues are scientific, not religious. Moreover, we reject the idea that users should or will “vote with their data.” First of all, SQL syntax and defaults are such as to make it hard to do the right thing and easy to do the wrong thing—they give you rope to hang yourself with, as it were. Second, we can expect users to do the right thing only if they're fully aware of all of the arguments on both sides, which in most cases they're demonstrably not. And if they do the wrong thing and then discover their mistake later, the damage is already done. What's more, it might not be possible, for all kinds of reasons, to correct the mistake later, either.

“Model the real world” is good, by the way. That's exactly what nulls and duplicates don't do.

Of course, you're absolutely right to say SQL is widely used, and obviously that situation isn't going to change any time soon. So what we have to do is try to educate SQL users to use the language wisely and well. Two of my most recent books from O'Reilly address this issue (and thanks for the chance to plug them!). *Relational Theory for Computer Professionals* is an attempt to explain relational principles as such (without a knowledge of those principles, there's no chance you'll be able to use SQL “wisely and well”). *SQL and Relational Theory* then goes on to show, for each aspect of relational theory, how to use SQL appropriately. In other words, it tells you what you should be doing. (It tells you what you shouldn't be doing, too!)

Hugh: Regarding books, I've already mentioned *SQL: A Comparative Survey*. That book is designed to be studied in parallel with another book of mine, *An Introduction to Relational Database Theory*, available from the same source.

As for that text by Don Chamberlin, an obvious response would be: “Well, he would say that, wouldn't he?” We find it

“The tenor of such citations smacks of ‘Codd wrote this, so it must be right.’ Why isn't Codd allowed to have made a mistake here and there? The rest of us do.”

rather annoying that Codd's own writings are so often cited in arguments like Chamberlin's. The tenor of such citations smacks of “Codd wrote this, so it must be right.” Why isn't Codd allowed to have made a mistake here and there? The rest of us do. If somebody wishes to express agreement with something Codd wrote, then they should say why they agree with it. After all, we've stated very clearly why we disagree with Rule 3, and Chamberlin doesn't explicitly refute, or even dispute, any of our points of disagreement. In any case:

- SQL's support for nulls isn't exactly systematic. The behavior of null in various operations, notably explicit comparisons, is consistent with its denotation as “value unknown,” but SQL gives it various other meanings, depending on

context, with which that behavior is not consistent at all. For example, the SUM of no numbers is null instead of being zero, the SOME of no truth values is null instead of FALSE, and the AVG of no numbers is null instead of being undefined. Also, its treatment in certain implicit comparisons is inconsistent with its treatment in explicit ones. For example, the result of SELECT DISTINCT on a table consisting of two rows that compare equal on all but one of their columns and each have null for the remaining column contains just one of those two rows. There's no other reasonable choice, of course—the problem lies in the treatment of explicit comparisons, which in turn arises from the very appearance of null in the first place.

- b. SQL's treatment is loosely based on an early proposal by Codd that Codd himself later abjured (albeit in favor of something even worse, involving two different kinds of null and a fourth truth value).

Note also that Chamberlin doesn't cite Codd in connection with the duplicate row issue. Presumably that wouldn't have been convenient, given Codd's publicly stated bitter opposition to SQL's duplicate row support. We find that people who like to defend SQL against our criticisms are often prone to this kind of cherry-picking in their choice of citations.

Chamberlin appeals to the users of database systems as the "true arbiters of the null and duplicate row issues." One problem I have with such statements is that the real end users aren't the ones the big SQL vendors take guidance from. Instead, they talk to the people who develop applications for those end users. Application developers are computer programmers. As such, they like to be in control (and thus maintain their job security); they don't like yielding control to the DBMS over matters they feel competent to deal with themselves. Moreover, developers tend to be more concerned with convenience in database definition and updating than with the ease of deriving useful and reliable information from the database.

In any case, I would argue that the choice offered to users isn't a genuine one, as no attempt has been made in SQL systems (as far as I know) to address perceived performance problems that might arise when a single base table containing nulls is decomposed into several base tables in order to eliminate those nulls.³

Chamberlin offers SELECT DISTINCT as a partial solution to the duplicate row problem, but this is a mite disingenuous. Most SQL systems make little or no attempt to take note of cases where SELECT ALL and SELECT DISTINCT are equivalent, even though the technology needed to detect such cases was known before the appearance of the first SQL products. As a consequence, SELECT DISTINCT usually runs a lot slower than SELECT ALL even when they're guaranteed to deliver the same result. SQL implementations thus force the user to work out if DISTINCT is needed (not always a trivial matter), as well as giving rise to traps for the unwary who accidentally leave it out when it's definitely needed. Hence there's no possibility for users to arbitrate between a system that always gives duplicate-free results and one that does so only on demand.

In any case, nulls and duplicate rows are by no means SQL's only harmful departures from relational theory. For example, whereas that theory places no significance on the order in which the attributes of a relation might appear, many SQL operations

depend on an ordering to the columns of a table. How are users to express a preference (or otherwise) for a system in which every column in every table has a name unique to that column, thus obviating the need for an ordering, when no such system

“Developers tend to be more concerned with convenience in database definition and updating than with the ease of deriving useful and reliable information from the database.”

exists on the commercial scene?

No to NoSQL!

The archetypal NoSQL product is Dynamo from Amazon.com. The 2007 ACM paper by Amazon.com states: “Customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers. . . . There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.” The Dynamo paper is where the popular claim originated that NoSQL products are faster, more scalable, and more available than relational products in certain clearly delineated scenarios such as online shopping carts. But is there any merit to the claim at all?

Chris: First off, let me make it very clear that I know almost nothing about Dynamo (or any other NoSQL product, come to that). But if the statement is correct that it provides “a simple primary-key only interface to meet the requirements of [certain rather simple] applications”—well, fine. I have no problem with that. If there's a class of applications that (a) are important for some pragmatic reason and (b) require only a limited subset of the system's full functionality, then I think it's perfectly reasonable for the system to provide a special interface tailored to just those requirements. Why, that's exactly what IMS did, with its Fast Path option! That special interface would support a carefully chosen subset of the full relational interface, and the implementation would be able to take advantage of the fact that the interface is circumscribed in just such a way. It might be able to make use of its own special stored data formats as well. And—just to spell the point out—I see no reason why the provision of that special interface and those special stored data formats

³ Darwen, Hugh. “How To Handle Missing Information Without Using NULL.” www.dcs.warwick.ac.uk/~hugh/TTM/Missing-info-without-nulls.pdf.

should have any negative effect at all on users who want to use the regular “full function” relational interface.

That said, if there’s a suggestion that Amazon’s various disaster scenarios, regarding tornados and the rest, are somehow more of a problem for relational systems than they are for nonrelational ones, then of course I reject that suggestion 100 percent. As so often, I strongly suspect that what’s going on here is some kind of confusion between what truly relational systems ought to be capable of and what today’s mainstream SQL products can actually do. If today’s SQL products fail to meet Amazon’s requirements, well, that might be a valid criticism of those products—but it’s not a valid criticism of relational systems in general.

To sum up: I do think we should discard SQL, as quickly as we can, and replace it by something better. Unfortunately, however, most of the people who currently want to discard SQL (or, at least, those who are most vocal about doing so) seem to want to do so for the wrong reasons. And there’s a strong likelihood that they’ll replace it, not by something better, but by something worse.

Hugh: So Amazon are using a Dynamo key value to access a giant blob whose structure is understood only by the applications. In that case they are happy to forgo all of those aforementioned six advantages given by Codd in 1974, not all of which are properly delivered by SQL products in any case. If the people at Amazon are satisfied that Dynamo provides everything they need, and they feel they have good reason to reject the use of any SQL products, then who are we to argue? The indictment seems to be of SQL products, not relational databases.

It’s easy to understand why the mainstream SQL products might be too “heavy” for Amazon’s needs. Those products have become extremely cluttered up with all sorts of features that would be of little or no use in the Amazon scenario: baroque support for user defined data types, pointers (in the form of REF

“If there’s a suggestion that Amazon’s various disaster scenarios, regarding tornados and the rest, are somehow more of a problem for relational systems than they are for nonrelational ones, then of course I reject that suggestion 100 percent.”

values), BLOBS and CLOBs, subtables and supertables, sequence generators, datalinks, locators, system versioned tables, and on and on.

Rel (dbappbuilder.sourceforge.net/Rel.html) is an implementation, by Dave Voorhis of the University of Derby, UK, of the relational language **Tutorial D** that we (Chris and I) devised for teaching and illustrative purposes. *Rel* is a very simple DBMS that meets all the criteria for being fully relational. If *Rel*, or one of the other prototype implementations of our *Manifesto*, were dressed up sufficiently for commercial purposes—including in particular the performance enhancements to be obtained by implementation of established optimization techniques and sophisticated storage structures—then it would be interesting to see if Amazon still preferred Dynamo.

All of that said, we admit that full scalability might be hard to achieve with a fully relational system. That’s because we require such a system to support expressions of arbitrary complexity for deriving whatever results might be desired from the database and for expressing whatever integrity constraints might be needed. Let’s suppose there are extreme cases where what runs in acceptable time with small databases is simply not feasible with large ones. Such cases would militate against the declaration of certain integrity constraints that current SQL products don’t even support. They also militate against the use of certain queries, in an OLTP context, that SQL products *do* support, but that problem can of course be avoided simply by not doing such queries. We conjecture that such cases would be unlikely to arise in Amazon’s shopping scenario. In any case, if they would still prefer Dynamo to a putative souped-up *Rel*, then so be it. If the benefits of a relational system we described in answer to one of your previous questions aren’t all needed in a particular situation, and provision of a more tailored solution in that situation is found to be cost effective, then who could argue that Amazon would have made a bad choice?

By the way, the bogey of scalability is sometimes advanced as justification for failing to provide any solution at all. A pertinent example is the lack of full support for integrity constraints in SQL systems (where something close to full support could be achieved by implementing the ISO standard’s CREATE ASSERTION statement, for example).⁴ But some databases are quite small and subject to quite infrequent updates. I use *Rel* for several small databases that I maintain for domestic and hobby purposes on my home computer. I have benefited significantly from the ability to define constraints that would be impossible to define in SQL without CREATE ASSERTION. Without those constraints, certain errors by me would have gone undetected, resulting in incorrect databases. I’m typically dealing with relations consisting of a few hundred tuples, and in some cases I’m updating no more than once per month. It seems unfair that small organizations, which have little clout with the SQL vendors, can’t also enjoy the benefits of such solutions, just because those solutions might not be practical for large organizations (with deep pockets, therefore listened to by the DBMS vendors) using OLTP on enormous databases. In this connection, one SQL DBMS implementer once told me privately that he agreed with me in principle but tellingly added that supporting CREATE ASSERTION would be very difficult in his product and lack of scalability for some kinds of constraint would give him a good get-out clause!

Another breed of NoSQL products that has gained considerable commercial momentum is “graph databases” such as Neo4J. In “Normalized Database Structure: A Brief Tutorial,” Codd carved out a special exception for such products: “It may be argued that in some applications the problems have an immediate natural formulation in terms of networks. This is true of some applications, such as studies of transportation networks, power-line networks, computer design, and the like. We shall call these network applications . . . Except in network applications, links should not be employed in the user’s data model.” Since the problems addressed by these products (e.g., shortest path) have no solution in relational calculus, do these products have a legitimate case to be non-relational?

⁴ See “CREATE ASSERTION: The Impossible Dream?” by Toon Koppelaars in this issue of the *NoCOUG Journal*.

Chris: Several points here. Yes, Codd did “carve out a special exception” for what he called network applications. But I’m not sure he was right to do so. As a simple counterexample, a company’s organization chart has “an immediate natural formulation in terms of networks” (in fact, often—though not always—in terms of a hierarchy, which is a simple special case). But it doesn’t follow that we need a network DBMS (i.e., one that exposes “links” or pointers to the user) in order to deal with corporate organizations, and of course we don’t.

“I do think we should discard SQL, as quickly as we can, and replace it by something better.”

Second, I’d like to point out that in the paper you reference, Codd also said this: “[Users often have] occasion to require tables to be printed out or displayed. What could be a simpler, more universally needed, and more universally understood data structure than a table? Why not permit . . . users to view all the data . . . in a tabular way?”

Third, any graph can always be represented—quite succinctly, in fact—in relational form. As for “shortest path” and other such problems, I say again that the relational model is only a minimum requirement. Even if you’re right when you suggest that the shortest path problem can’t be formulated in relational calculus—I presume you’re referring to the fact that the relational calculus as originally defined had no support for the famous “ancestral” problem—well, that’s not to say it never will have such support. In fact, a great deal of research has been done on adding such support (and implementing it efficiently, too).

Fourth, let’s agree for the sake of the argument that there are some problems that graph-based DBMSs can solve better than relational ones. I don’t have an issue with that. My position is this: We know the class of problems for which relational systems are suited is very large—but it’s not necessarily universal. But I very much doubt whether any other approach is universal either. So my objection isn’t to using, e.g., graph-based DBMSs to solve problems that they solve well; rather, my objection is to attempts to solve by nonrelational means problems that can reasonably, perhaps better, be solved by relational means. In other words, graph-based DBMSs (for example) might well have a role to play, but that role is *not* to take over the entire database world. To repeat something I’ve said elsewhere: I’ve never seen a proposal for “taking over the database world”—i.e., for replacing the relational model—in which the person doing the proposing really understood the relational model. Surely, if you want to claim that Technology *A* is no good and needs to be replaced by Technology *B*, then it’s incumbent on you to understand Technology *A* in the first place? And, more specifically, to demonstrate that Technology *B* solves not only all of the problems that Technology *A* does, but also some problem that Technology *A* doesn’t?

Hugh: Well, graph DBMSs and the like simply are nonrelational. Of course we don’t suggest that *all* databases should be relational—only that all general purpose ones should be. But if you were to ask if it’s legitimate to claim that solutions to problems like “shortest path” are inherently unobtainable with a relational system, then the answer is an emphatic “No!” As Chris has effec-

tively already said, having no solution in relational calculus doesn’t mean it’s impossible for a relational DBMS to provide the necessary operators. For example, **Tutorial D** already includes an operator, TCLOSE, for deriving a relation representing the transitive closure of its operand, a recursive relation. And even SQL includes support—rather elaborate support, in fact—for recursive table expressions in general. Any operator that’s closed over relations is admissible in a relational database language.

Now, the proponents of graph databases might wish to argue that their systems can provide much faster solutions to such problems than could ever be obtained by implementations of suitable relational operators. But suppose the graph DBMS were the *engine* of a relational DBMS, such that a relational expression is mapped under the covers to an equivalent expression or procedure in the graph DBMS’s language. Wouldn’t we then see the relational DBMS performing pretty much as well—or as badly!—as the graph DBMS on its own? And wouldn’t its user then be receiving all the benefits claimed for relational DBMSs in general *in addition* to those claimed for graph DBMSs in particular? It’s interesting to see in this connection that some of the DBMSs listed in the Wikipedia article “Graph database,” notably those available from Oracle, use some form of SQL as their query language.

Another breed of NoSQL products that has gained considerable commercial momentum is the so-called “Big Data” products like Hadoop that aim to process non-transactional data outside the transactional DBMS. It was apparent that the glaring drawback of this class of NoSQL products was the absence of SQL, and so there has been a rush to provide SQL-like functionality in this space, with Impala from Cloudera leading the way. Which leads to the question: Is it kosher to decouple relational algebra and relational calculus from the DBMS as Impala has done?

Chris: Before I became “a database person,” I was a languages person. I worked for several years at IBM Hursley (in the UK), which in those days was the home of PL/I. (Of course, you might never have heard of PL/I, and I’ll be the first to admit that as a language it looks a little antiquated now. But it was a big deal at the time—and a big revenue earner for IBM, I might add.) So when I first learned about Ted Codd’s relational model, I wanted to add relations and relational operators to PL/I—in order that PL/I would be able to operate on data in a relational database, of course, but not only for that reason; I always thought it would be useful to have “local” relations, meaning ones that weren’t in the database, and to be able to operate on those relations by means of joins and unions and so on. So if that’s what you mean by “decoupling relational algebra and relational calculus from the DBMS,” then I’m all for it.

“If the people at Amazon are satisfied that Dynamo provides everything they need, and they feel they have good reason to reject the use of any SQL products, then who are we to argue?”

“In our experience, those who disparage relational are almost invariably very far from being properly informed and almost invariably equate “relational” with SQL and its implementations.”

But you touch on something else here: “Big Data.” Sorry if I’m beginning to sound like a broken record—I guess that metaphor is pretty antiquated too!—but I see no reason why a relational system shouldn’t be able to handle “big data” perfectly well. Data size is, of course, an implementation concern, not a model concern. As I said in answer to your second question, the relational model is deliberately silent on all matters of physical implementation. Just because the implementation has to deal with enormous volumes of data, that’s no reason, as far as I can see, why the user interface has to be anything other than relational.

Hugh: My observations on graph databases seem applicable here too. Couldn’t Impala be thought of as the DBMS and Hadoop as Impala’s database engine? Well, up to a point, perhaps, but if Impala users have to use Hadoop itself for certain purposes (perhaps database definition? updating? constraint enforcement?), then we could hardly call Impala a fully relational DBMS, even if its language, as far as it goes, were in keeping with the relational model. In any case, there have been many examples over the

years of the need being perceived for a “decoupled” relational front end to a nonrelational system. For example, in the early 1980s a group in IBM (UK) developed an SQL front end to IBM’s ancient and still running hierarchical system IMS. There can be no objection to such products; on the contrary, if the front end were fully relational (as opposed to SQL), we would encourage them to be provided where the need arises.

Parting Advice

Thank you for spending so much time with us today. Most NoCOUG Journal readers have long careers ahead of them in the relational field. Do you have any parting advice for them?

Chris: As for the time, you’re more than welcome. I’m always happy to do anything I can to help dispel all the myths there are out there regarding relational systems. Parting advice? Well, if you’re right that they have long careers ahead of them in the *re*-lational field—my italics—then I congratulate them; the subject is intellectually stimulating, and pragmatically important, and it

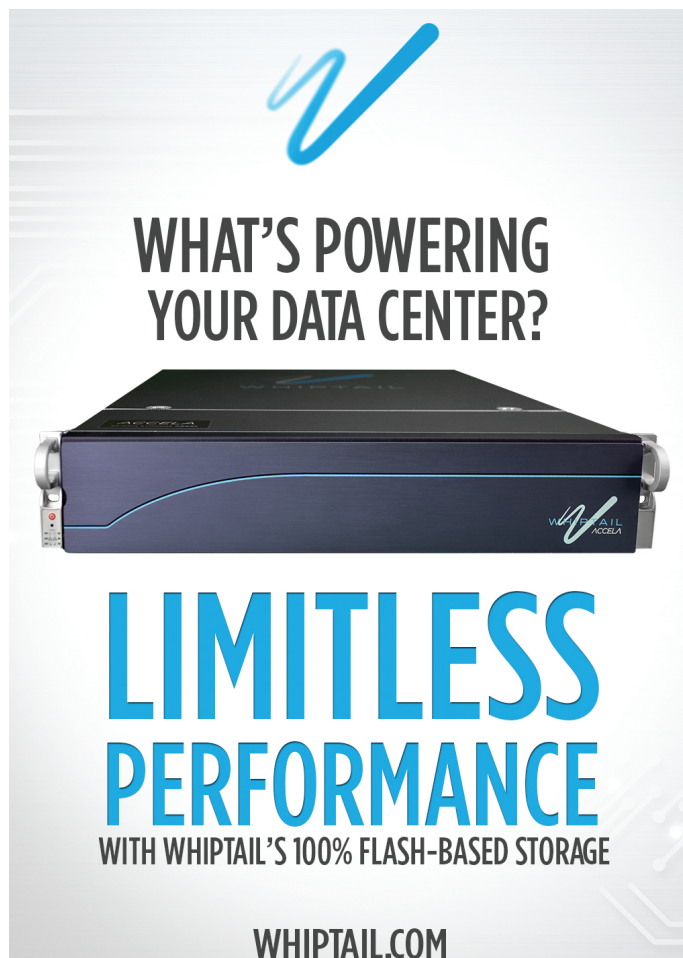
“If the NoSQL movement were to advocate relational as one of its proposed alternatives to SQL, then we would sign up ourselves!”

can be *fun*. One piece of advice I can offer is this (and I’m sorry if this sounds a little self-serving—I don’t mean it to be): 1. Learn the relational model, by reading the right books and/or attending the right courses. 2. Go out and get your hands dirty working on a real project for a year or three. 3. Come back and read those books and/or attend those courses again. A related piece of advice is: Learn the relational model first, SQL second (doing it the other way around is *hard*). Finally: Read either or both of Ted Codd’s first two papers every year.⁵

Hugh: Yes, please do your best to become properly informed about what “relational” really means. In our experience, those who disparage relational are almost invariably very far from being properly informed and almost invariably equate “relational” with SQL and its implementations. If the NoSQL movement were to advocate relational as one of its proposed alternatives to SQL, then we would sign up ourselves! ▲

Interview conducted by Iggy Fernandez

Picture of Chris Date by Douglas Robertson (Edinburgh)



WHAT'S POWERING
YOUR DATA CENTER?

LIMITLESS
PERFORMANCE

WITH WHIPTAIL'S 100% FLASH-BASED STORAGE

WHIPTAIL.COM

⁵ *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599 (August 19th, 1969), reprinted in ACM SIGMOD Record 38, No. 1 (March 2009); *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6 (June 1970), reprinted in the 100th issue of the NoCOUG Journal (www.nocoug.org/Journal/NoCOUG_Journal_201111.pdf).

CREATE ASSERTION: The Impossible Dream?

by Toon Koppelaars



Toon Koppelaars

One of the promises of relational database management systems (RDBMSs) was that they would provide us with full support for declarative data integrity constraints. Unfortunately, more than 25 years after their appearance into the marketplace, they still fail to do so. Although a few good mechanisms are offered to declaratively implement constraints, for the vast majority of constraints we still have to develop (i.e., code) our own mechanisms to implement them. Just to be sure we are all on the same page here, let me briefly clarify this and take a structured (class-by-class) approach in doing so.

A common classification scheme for data integrity constraints is to look at the scope of data that is being constrained. We'll briefly introduce this classification here. For the interested reader (with some mathematics education), there's a book on data integrity constraints and other related topics that is worthwhile reading: *Applied Mathematics for Database Professionals* (AM4DP), Apress 2007.

- *Attribute constraints* deal with a single column value inside a single row. For instance: salary must be between 1500 and 9500.
- *Tuple constraints* deal with two or more column values, still inside a single row. For instance: trainers must earn more than 5000.
- *Table constraints* deal with column values across multiple rows inside a single table. For instance: departments with trainers must also employ at least one clerk.
- *Database constraints* deal with column values that span multiple tables. For instance: trainers cannot be employed in LA.

We all know the mechanism to implement attribute and tuple constraints: the CHECK clause, which is part of the CREATE (and ALTER) table statement.

```
alter table EMP add constraint SAL_CHK
check (SALARY between 1500 and 9500);
```

```
alter table EMP add constraint TRN_SAL_CHK
check (JOB != 'TRAINER' or SALARY > 5000);
```

These are *declarative*: the expressions specify *what* needs to be implemented, not *how*. Fortunately for a DBMS vendor, the *how* in the cases of attribute and tuple constraints is not that difficult to implement given the specification of the *what*. We'll talk about how this is no longer the case for table and database constraints.

But first we demonstrate how the ANSI ISO committee that developed the SQL standard envisioned full declarative constraint support in RDBMSs. They introduced a language construct for this called a "SQL assertion." Using assertions, all table and database constraints can be declaratively specified. Here's an assertion for the table constraint involving only EMP:

```
create assertion TRN_CLRK as
CHECK (
  not exists (
    select 'department in violation'
    from (select distinct DEPTNO from EMP) e1
    where exists (
      select 'a trainer'
      from EMP e2
      where e2.DEPTNO = e1.DEPTNO
      and e2.JOB = 'TRAINER'
    )
    and not exists (
      select 'a clerk'
      from EMP e3
      where e3.DEPTNO = e1.DEPTNO
      and e3.JOB = 'CLERK'
    )
  )
);
```

And an assertion for the database constraint, involving both EMP and DEPT, could be:

```
create assertion TRN_CLRK as
CHECK (
  not exists (
    select 'an LA department'
    from DEPT d
    where d.location = 'LA'
    and exists (
      select 'a trainer in it'
      from EMP e
      where e.DEPTNO = d.DEPTNO
      and e.JOB = 'TRAINER'
    )
  )
);
```

One important point to make right now is that primary, unique, and foreign keys are just special cases of assertions. For instance, here's the assertion for a primary key in EMP:

```
create assertion EMP_PK as
CHECK (
  not exists (
    select 'two EMPs with same EMPNO'
    from EMP e1, EMP e2
    where e1.EMPNO = e2.EMPNO
  )
);
```

```

    and e1.rowid != e2.rowid
  )
);

```

The assertion for a foreign key is left to the reader. Since primary, unique, and foreign keys are so ubiquitously present in (and fundamental to) database designs, the SQL standard committee introduced special shorthand syntax for these constraints. This, in turn, opened up the opportunity for DBMS vendors to implement dedicated constraint validation code for these three subclasses of constraints. The fact that a constraint can be specified via shorthand immediately enables a DBMS vendor to come up with an efficient implementation for it. This is, in general, not true for constraints specified through assertions. But why isn't it true? After all, aren't we supplying the DBMS with a syntactically correct and executable expression that it could easily use for validating the constraint?

Well, yes, but . . . (and this is a big “but”). Let's take a closer look at the table constraint (trainers requiring a clerk in the same department). Would you accept an evaluation of the expression as specified in that create assertion command *every time a transaction changes something in the EMP table*? This is an execution strategy that could easily be implemented by the DBMS vendor. But it would be one that you would not accept. Why not? Because this implementation strategy is a very inefficient one. You would have at least two major objections to it.

1. Clearly not every transaction that changes something in the EMP table requires validation of the constraint. Take, for example, a transaction that increases the salary of an employee. We should hope that in this case the DBMS would not evaluate the table constraint, for it would be completely superfluous.
2. The expression given in the assertion validates all data (i.e., all departments) in the EMP table, whereas the transaction at hand might only be inserting a trainer in department 42 and therefore necessitate validation of the constraint for department 42 only.

It is here where the complexity surfaces that we ask the DBMS vendor to solve for us. The vendor is tasked with developing an algorithm that takes as its input the assertion expression and produces as its output an implementation strategy for the assertion that is efficient in that it at least addresses the above two major objections. Note: the algorithm should work for *every arbitrarily complex assertion*, which makes it orders of magnitude more complex than developing an algorithm taking (predefined) shorthand as its input. To illustrate the kind of output that the algorithm needs to produce, let us investigate an implementation strategy for the example table constraint and make sure objection 1 is dealt with. The first challenge is to identify the kinds of transactions that would need validation of the constraint. A way to do this is to investigate insert, delete, and update statements separately. Thinking about this for a little bit, you can come up with the following scenarios:

- For inserts: only inserts of a trainer require validation of the constraint. The DBMS should verify whether a clerk is present in the same department.
- For deletes: only deletes of a clerk require validation of the constraint. The DBMS should verify whether, if this is the last clerk being deleted from that department, then no trainer is employed anymore in the department.

- For updates: only updates on the involved columns *could* require validation of the constraint. Note that the involved columns in this constraint are DEPTNO and JOB. Updates of all other EMP columns will not affect the validity of the constraint. Now what kinds of updates on DEPTNO and JOB actually do require validation? Well, if we promote a salesman to become a manager, the constraint is obviously unaffected. Only if we update the JOB column to the trainer value (this mimics a new trainer insert) or update it away from the clerk value (this mimics a clerk delete) does the constraint require validation. Thinking about the DEPTNO column, we can come up with following scenarios: only if we update the DEPTNO value of a trainer or a clerk does the constraint require validation. This kind of update represents a move of the employee to a new department. When moving a trainer, the DBMS will have to validate the constraint for the new department. And when moving a clerk, the DBMS will have to validate the constraint for the old department.

Following from the second objection, the next challenge for the DBMS is to compute the minimal expression to validate in all of the above cases. The assertion expression validating all departments was:

```

not exists (
  select 'department in violation'
  from (select distinct DEPTNO from EMP) e1
  where exists (
    select 'a trainer'
    from EMP e2
    where e2.DEPTNO = e1.DEPTNO
    and e2.JOB = 'TRAINER'
  )
  and not exists (
    select 'a clerk'
    from EMP e3
    where e3.DEPTNO = e1.DEPTNO
    and e3.JOB = 'CLERK'
  )
)

```

It is now the task of the algorithm to identify the first (and only) inline view in the expression above as the lever to only validate a subset of the departments. So, for instance, when a transaction inserts a trainer in department 42, the DBMS should only execute the following (or some equivalent) expression to determine the continued validity of the constraint (note the change of the inline view):

```

not exists (
  select 'department in violation'
  from (select 42 as DEPTNO from DUAL) e1
  where exists (
    select 'a trainer'
    from EMP e2
    where e2.DEPTNO = e1.DEPTNO
    and e2.JOB = 'TRAINER'
  )
  and not exists (
    select 'a clerk'
    from EMP e3
    where e3.DEPTNO = e1.DEPTNO
    and e3.JOB = 'CLERK'
  )
)

```

Somehow we humans seem to have little trouble in coming up with all of the above: when to validate the constraint and how to (minimally) validate the constraint in those cases. It does require some thinking on our part. But all in all it's not too difficult for

us. Nevertheless, writing an algorithm that computes all of this given some arbitrarily complex assertion is something completely different. Neither Oracle nor any other vendor has been able to produce this yet. In Chapter 11 of the AM4DP book, you can find a much more detailed treatment of this topic.

So where does this all leave us now? As stated earlier, for the vast majority of (table and database) constraints, we still have to develop (i.e., code) our own mechanisms to implement them. One vehicle of choice would be to employ database triggers for this purpose. This is, in fact, no different from what the database vendor already does for the constraints that we have shorthand for. Let's take a look at the foreign key for example. We specify it declaratively as follows:

```
alter table EMP add constraint emp_dept_fk
foreign key (DEPTNO) references DEPT(DEPTNO);
```

The vendor has a straightforward algorithm embedded in its kernel that takes the above shorthand specification and injects constraint validating code right after:

- Inserts of rows into the EMP table, to verify if the referencing department exists in the DEPT table.
- Updates of the DEPTNO column of rows in EMP, to verify whether the new value exists in the DEPT table.
- Deletes of rows from the DEPT table, to verify whether the department being deleted is no longer being referenced by any row in the EMP table.
- Updates of the DEPTNO column of rows in DEPT, to verify whether the old DEPTNO value is no longer being referenced by any row in the EMP table.

And, of course, the algorithm only validates minimally—per department in this case. Conceptually this validating code executes (“fires”) at the same moment that after-statement table triggers would fire. In the next few paragraphs we'll sketch the trigger design to implement the other table and database constraints ourselves.

Step 1: Design after insert, delete, and update *row* triggers to capture what is being changed (inserted, deleted, or updated) in a table. We can use a global temporary table to track the changes and build three views on top of it called *inserted_rows*, *deleted_rows*, and *updated_rows*. These views will hold details of the rows that have been inserted, deleted, or updated respectively by the statement that caused the row triggers to fire. In scientific papers this information is often referred to as the “Transition Effect” of a (dml) statement. Chapter 11 of AM4DP has detailed examples of building the row triggers for maintaining the Transition Effect.

Step 2: For the given assertion, write queries on top of these three views that reflect when the assertion is in need of validation. The semantics of these queries should be such that if they return no rows, then the assertion does not require validation. And if they do, then the assertion does require validation. We'll call these queries Transition Effect (TE) queries. These TE queries will “guard” the execution of code that performs the actual validation of the assertion: they prevent this code from firing when it's not necessary and, vice versa, will ensure that it fires when it's necessary. For the example table constraint, this results in the following three queries:

```
select deptno
from inserted_rows
```

```
where job = 'TRAINER'
--
select deptno
from deleted_rows
where job = 'CLERK'
--
select old_deptno as deptno
from updated_rows
where((old_deptno != new_deptno) or (old_job != new_job))
and old_job = 'CLERK'
union
select new_deptno as deptno
from updated_rows
where((old_deptno != new_deptno) or (old_job != new_job))
and new_job = 'TRAINER'
```

The last TE query above is a bit tricky, but it reflects exactly the kind of updates that require validation of our table constraint.

Step 3: Rewrite the assertion expression into a minimal query that uses information generated by the TE queries (a DEPTNO value in this case) and tries to detect a violation of the assertion. We'll use this query in the after insert, delete, and update *statement* triggers to perform the actual constraint validation. Here is the after insert trigger code, which combines the TE query with this validation query. The delete and update triggers are coded similarly.

```
create or replace trigger EMP_AIS_TRN_CLRK
after insert on EMP
declare pl_message varchar2(80);
begin
--
for r in (select deptno from inserted_rows where job = 'TRAINER')
loop
--
begin
--
select 'Department '||to_char(deptno)||' requires a clerk.'
into pl_message
from (select r.deptno as DEPTNO from DUAL) e1
where exists (
select 'a trainer'
from EMP e2
where e2.DEPTNO = e1.DEPTNO
and e2.JOB = 'TRAINER'
)
and not exists (
select 'a clerk'
from EMP e3
where e3.DEPTNO = e1.DEPTNO
and e3.JOB = 'CLERK'
);
--
raise_application_error(-20000,pl_message);
--
exception when no_data_found then null;
end;
--
end loop;
--
end;
/
```

We are now almost done. There is one final step to perform.

Step 4: Add serialization code to the statement triggers. The scope of this short paper prevents a detailed treatment of this topic. Without the necessary serialization code, you can end up having two concurrently executing transactions both successfully validating the same assertion, but when they're committed, they leave the database in an incorrect state. This is due to the fact that Oracle's snapshot isolation (MVCC) does not guarantee serializability. You can find more on this topic in the AM4DP book, Chapter 11.

(continued on page 26)

Oracle Identity and Access Manager 11g for Administrators

A Book Review by Brian Hitchcock

Details

Author: Atul Kumar

ISBN: 978-1-84968-268-8

Pages: 280

Year of Publication: 2011

Edition: 1

List Price: \$59.99

Publisher: Packt Publishing

Overall Review: This book offers some good information, but it contains too many screenshots and typos. I'm not sure this book is worth its \$60 price.

Target Audience: Administrators working with Fusion Middleware and/or Fusion Applications.

Would you recommend this book to others: I would recommend this book conditionally, but please read the whole review.

Who will get the most from this book? Administrators supporting Oracle Fusion Middleware, specifically OID, OAM, and OIM.

Is this book platform specific: No.

Why did I obtain this book? I wanted to learn more about these components of Oracle Fusion Middleware, and NoCOUG purchased this book for me to review.

Overall Review

I am currently supporting Oracle Fusion Middleware (FMW) for multiple customers. Since Oracle Identity Manager (OIM), Oracle Access Manager (OAM), and Oracle Internet Directory (OID) are components of Fusion Middleware, I was looking for books that would help me understand how they operate and integrate with the applications they support. I was looking for specific details of how they work and how to troubleshoot them when they don't.

The many pieces of FMW came from multiple companies acquired by Oracle. I would have liked an overview showing where OIM, OAM, and OID came from, and how Oracle has, over time, integrated them—as well as how this integration will likely continue.

I did learn a lot about OAM: specifically, where things get installed and how to find various parts of OAM in the screens of the GUI.

As we use more and more GUIs to manage software, screenshots are necessary to describe how to do things. At the same



time, it is easy to fill many pages with screenshots, instead of writing. This book explores the limit of how many screenshots are too many. I would have preferred more writing to go with the large supply of screenshots.

Normally I don't find typos to be an issue. In this book, however, there were so many that it distracted me from the material I was trying to read.

Preface

The author tells us what this book covers and describes what you will need if you want to actually install these Oracle products on a machine of your own. I did not attempt to install anything; I see enough of these installations every day! In the five pages of this Preface, I found two typos. Not a good start.

We are told that this book will cover various aspects of IDAM, which is a new term to me (it stands for Oracle Identity and Access Management). I'm used to seeing this group of products identified as Oracle Identity Management (OIM), which refers to the group of products that include Oracle Identity Manager (OIM). Confused? It is confusing. Anyway, IDAM was new to me.

Chapter 1—Oracle Identity Management: Overview and Architecture

This chapter provides an overview of Oracle Identity Management—specifically, Oracle Access Manager (OAM) and Oracle Identity Manager (OIM). Both of these, and many other parts of FMW, rely on WebLogic Server (WLS), and an overview of WLS is offered as well. In the section on WLS, we are told about clusters being a group of WebLogic Servers and that these servers can be Admin Servers or Managed Servers. Really? I thought only Managed Servers could be clustered. It would have been good to have this clarified. The section explaining Node Manager is okay, but I would stress its role in starting and stopping managed servers on remote machines.

I learned several things from this chapter, including the fact that the OAM policy store must be in an RDBMS and that OIM uses database schema MDS for storing configuration information and schema OIM for user information.

I found two typos in this chapter.

Chapter 2—Installing Oracle Identity and Access Manager

This chapter covers the steps of installing both OIM and OAM. There are lots of screenshots, which are good, but it would be better to have more insight as to what is happening, why it is happening, and what can go wrong.

The need to install Coherence is mentioned, but no explanation is given as to what it is, nor are there details of what the JMS

persistence store is or does. This is mainly a checklist for installation steps. This is okay, but you don't learn much about how things work.

I did learn that SOA suite is required for OIM. I wanted some troubleshooting tips, since installations don't always go according to plan. Similarly, what if the newly installed components don't start? The utilities "pack" and "unpack" are mentioned but not explained. What are they and what do they do?

Only one typo in this chapter, but it is pretty bad. On page 28 we have this section heading: "Run Repository Creaion . . ." I would think a spell check would have caught this.

Chapter 3—IDAM Directory Structure and Files

I was surprised by this chapter. It is a whole chapter of screenshots of files in directories. These screenshots were taken on a terminal window that had a dark background, and the text is dark. It is almost impossible to read most of the text in these screenshots. I think this would have been noticed by anyone else that read this chapter. There is a table that tells us what each item in the DOMAIN_HOME directory is. While this is very thorough, I don't think most readers need to have all of these entries explained.

If you weren't familiar (already) with WLS, OIM, and OAM, I don't think you could get much from this chapter. So many acronyms fly by so fast!

I found three typos in this chapter.

Chapter 4—Start-up Shutdown IDAM

The steps needed to start up and shut down the components of IDAM are explained. But I think there is too much space taken up with listing commands. I don't think we need to be shown the full syntax of a command with "start" replaced by "stop." The concept of a Node Manager being associated with a "machine" is brought up, but it needs to be explained more fully. It isn't made clear that you can stop (but not start) the Admin Server from the WLS Admin Console.

Chapter 5—OAM Administration and Navigation

I found a lot of good information here. The description of the Identity store was helpful. The Oracle Secure Token Service is mentioned but not explained. I'm not sure why it was introduced. The description of how to create a new User Identity Store is fine, but I don't understand when I would need to do this. LDAP is mentioned but not explained.

Some of the screenshots have sections that appear to have been redacted, i.e., covered up in a sloppy manner. I understand that you need to obscure some of the fields in screenshots, but you could edit in some fake values.

But back to the good parts. I liked the description of OAM policy and session data store, and learning that these two stores can be in a single database or two separate databases. The information about the OAM config data store was good, and I didn't know that the OAM Key Store is not available through any of the GUI consoles and can't be viewed or modified. The process of OAM server registration was explained well. For the first time I see where the OAM agent gets set up and what it is protecting. The process of registering OAM agents using the Admin console was good. I would have liked to have seen more discussion of when to use OAM WebAgent versus OSSO.

I found two typos in this chapter.

Chapter 6—OAM Policy Component and Single Sign-On

This chapter opens with a description of the primary function of OAM. This is good, but I really want more explanation. An example of how OAM fits into a real-world application would be a big help. However, there is a good discussion of the definition of application domain and resource type. The host identifier is described as a list of all the URLs that can be used to access an application. I'm not clear what happens if I fail to include all the combinations: does that mean the URL I left out is not protected?

The explanation of OAM's SSO capability was useful. Next is a discussion of the success and failure URLs, referring to where users are sent if their attempt to log in is successful or fails. I'm confused about the success URL. Usually I get sent to the protected resource after I authenticate. In the screenshot this field is blank. Is this a real-world example or do we normally have to supply a success URL? An example would have helped me understand how these URLs work.

The SSO login request flow diagram is confusing. Some of the items numbered in the diagram don't have corresponding sections in the text on the next page.

The next diagram, showing SSO login request flow with OSSO agents, is also confusing. It is hard to see which numbers correspond to which parts of the diagram.

The chapter ends with good information about managing an application domain. The example is good, and this time the screenshots presented really help explain what the text is discussing.

I found two typos in this chapter.

Chapter 7—OAM Session Management

The definition of session was simple and to the point. The flow of OAM session management is shown in a diagram. Again we have a diagram with numbers that aren't linked with the corresponding sections in the text. The description of each step in the diagram was good. I learned new things from the description of OAM Session Management, Session Management Engine (SME), and Oracle Coherence. How Coherence is used to share session state among multiple OAM servers was shown in the next diagram; I didn't know this much about Coherence.

I also liked the description of the OAM clustered environment as well as the discussion of user session lifecycle and the settings that affect this lifecycle.

Chapter 8—Installing and Configuring OAM Agents

OAM agents are a topic that I am very interested in. I knew they were part of OAM, but I didn't know much else about them. This chapter helped me a lot. For example, I didn't know that by default, all resources (URLs) are protected by OAM. You have to specifically set up a resource that has a separate authentication scheme to allow access to users that are not authenticated.

Chapter 9—OIM Navigation: Administration and Design Console

Reading this chapter I learned that the screens of the OIM administration console are different depending on how you access it. If you are an authenticated user, you see more options than the self-service screens that are shown to nonauthenticated users. I would have liked to learn when I need to use each of these OIM interfaces. An example of the workflows requiring these different interfaces would also be good.

At one point, the “attestation process” is mentioned but not explained. I assume this refers to some process in the OIM workflow, but I don’t know this for sure. I tried the index but did not find an entry for this topic.

The advanced administration console screens have various tabs that are highlighted in the screenshot and listed in the text, but when do we need to use each tab?

The section on the OIM Design Console doesn’t make clear if this utility is only available on Windows or not. I have run this utility on Linux systems. It would be good to include some information about which releases run on which platforms. The sections covering how to navigate through the screens of this utility were helpful, since I am using this on my current job.

I had not heard of SPML web services or Service Provisioning Markup Language (SPML) before. They provide OIM services to client applications.

I found two typos in this chapter.

Chapter 10—OIM Connectors—Installation and Configuration

A good explanation of the OIM connector opens this chapter. The components that make up a connector are listed, although not in the same order that they are discussed in the text that follows. We are told that the process form captures the data needed to provision a user; if we had ten users, we would have ten instances of this process form. I wanted to learn more about how this process scales. What happens if we have 1,000 users?

We learn next about the reconciliation process. This occurs when user data in OIM needs to be reconciled with user data in a target system. The target system might be OID, Oracle E-Business Suite, or Microsoft Active Directory. However, there wasn’t any information about how this process actually works. What happens when there is a conflict between the data in these two systems? Which system wins? What happens to data in OIM that is not in the target system? Perhaps this process only works in one direction?

Starting with the section covering deploying the OIM connector for Oracle Internet Directory and continuing to the end of this chapter, far too much time is spent on the details of installation. The installation details are presented for each combination of OIM to target system. I doubt most readers will need more than one type of connector. I think the time should have been spent discussing operational issues, such as what goes wrong and how to fix it.

The last topic covers transferring connectors from test to production using Deployment Manager. The coverage here was good. I did not know about this process.

I found six typos in this chapter.

Chapter 11—OIM Configuration and Tasks

I had not seen the WebLogic Full Client before, so this section was useful to me. This is a jar file needed by various OIM utilities. The MDS repository is referenced, but I don’t know what this is. I wanted more coverage of this topic and details of the database schema that is used.

I found one typo in this chapter.

Chapter 12—OAM Integration with Fusion Middleware and EBS R12

This was, for me, the best chapter of the entire book. I learned many new things that will help me on the job. The summary of the changes that are done when a Fusion Middleware application

is integrated with OAM to provide SSO was very good.

The six pages that discuss Fusion Middleware security concepts are also very good. I didn’t know that the credential and policy stores are pointed to the same type of store.

However, in this chapter I also found the worst typo seen so far. I present this section title just the way it appears in the book: “Sentence case and hyphenate High-level for OAM with FMW.” This is a really bad typo. I assume the author intended this to be “High-level steps to integrate OAM with FMW.” The diagram describing the request flow for EBS integrated with OAM is missing some arrowheads, which makes it difficult to follow the flow.

I found six typos in this chapter.

Chapter 13—Logging and Auditing for OIM/OAM

This chapter covers the different available logging methods. The Oracle Diagnostic Logging (ODL) framework is explained, including the various loggers and log handlers available. Changing log level is covered using two different utilities. First is the WebLogic Scripting Tool (WLST), which is a command-line utility. Second is the Fusion Middleware Control, a GUI interface featuring Oracle Enterprise Manager that has been extended to manage FMW. The WebLogic logging service and OAM auditing are also covered.

The Remote Diagnostic Agent (RDA) is explained. This utility gathers all kinds of information and generates a zip file to be uploaded to Oracle Support. I know from personal experience that RDA can save significant time when working with Oracle Support. The section covering configuring RDA was good, as it explained that RDA is only available for OAM and not OIM—or, at least, not yet.

I found three typos in this chapter.

Appendix

The appendix covers a number of FAQs and then has a section on common issues that covers starting and stopping various components and several OIM issues.

Conclusion

While I did learn valuable things from this book, there are too many typos and too much space is used on screenshots. I expected more insights into how these Oracle products work and how to fix them when they break. I don’t usually comment on typos because they don’t usually interfere with reading a book. In this case, however, the typos were so frequent that they did indeed interfere with my reading. If I can find 30 typos by reading this book once—and I wasn’t looking for typos; I was just trying to read the text—why weren’t most, if not all, of these found before the book got to me?

Inside the back cover I see that my copy of this book was printed 10 May 2013. The copyright page shows 2001. I don’t understand how, between 2001 and May 2013, these typos were not found and corrected. Further, as it becomes normal to print books “on demand,” i.e., as they are ordered, instead of in one huge print run, I would expect the number of simple mistakes to go down, not up.

I also want to add what I would do if I were asked to make this book better. I know what I wanted from this title and what I need to help me at work. I would discuss the complete flow of user access without OIM, OAM, and OID, including the log files in

(continued on page 26)

RuleGen 3.0: Data Quality Assurance

by Mark Rooijakkers



Mark Rooijakkers

CB is the logistics service provider for the book, media, fashion, and healthcare industries. With no fewer than 65,000 square meters of warehousing and 120 trucks, CB-Logistics operates an extensive distribution network in the Netherlands and Belgium, and offers wide-ranging and integral logistics services. As an example, we distribute on average 70,000 books on a daily basis from our central warehouses to bookstores within our network.

This, of course, would be impossible without a very high degree of IT to automate and support our many business processes. The 60 people working in our IT department maintain a very large, custom-made, back-office application landscape that supports the complete order value stream: from order entry all the way to product delivery and associated financial handling. Main components within our IT landscape are the Oracle 11g DBMS, (traditional) WebForms, and an increasing Apex footprint. All of our applications have been developed in a *database-centric* manner: our Forms and Apex pages are *thin*, and our database is *fat*. All business logic resides and runs inside the database tier as PL/SQL code.

Data quality is of vital importance for us. In our 24/7 operation, we cannot allow the garbage-in, garbage-out syndrome. The specification and rigorous implementation of business rules (data integrity constraints) using table triggers that disallow garbage to enter in our databases is a continuous high priority in our software development practice. Some eight years ago we decided to investigate new ways to implement our business rules, as different developers approached this in different ways (just think about the different ways to circumvent the infamous mutating table error, or using materialized views and function-based indexes for special cases), taking different amounts of effort and resulting in different levels of maintainability of the code involved. It was back then that we discovered, investigated, and adopted the RuleGen framework (www.rulegen.com).

In this article we will give a bird's-eye overview of how RuleGen enables us to implement (or change) our business rules inside the database quickly and confidently. RuleGen is a PL/SQL generator and is itself built in PL/SQL. Using an Apex front-end application, a database developer can quickly specify new business rules. The way this is done is that the developer needs to specify *when* a business rule needs to be validated and *how* it should then be validated. This is done through specifying a few queries and is really quite simple. What's better than to just provide you with an example?

Let's assume that we have the following business rule: "The sum net value of the books ordered by a bookstore cannot exceed this bookstore's daily net-value threshold." Data-model-wise, it would look like this BOOKSTORE table that holds the daily threshold:

```
create table book_store
(store_id number not null -- id for this bookstore
,...other columns...
,daily_net_value_threshold number not null
,primary key (store_id));
```

And we have an ORDER table that holds the book orders for bookstores:

```
create table book_order
(order_id number not null -- id for this order
,order_date date not null
,store_id number not null -- reference to bookstore
,book_id number not null -- reference to book
,net_price number not null -- negotiated net price
,...other columns...
,primary key (order_id)
,foreign key (store_id) references book_store(store_id));
```

So the rule basically dictates that the result of the following query must at all times be empty:

```
select *
from (select s.store_id
      ,trunc(o.order_date)
      ,sum(o.net_price) as sum_price
      ,s.daily_net_value_threshold
      from book_store s
      ,book_order o
      where s.store_id = o.store_id
      group by s.store_id, trunc(o.order_date))
where sum_price > daily_net_value_threshold;
```

It's always good to come up with a query like this first. It will show you what the involved columns for this business rule are (which helps you in the first step below); moreover, such a query is a nonambiguous way of specifying the constraint, next to the (potentially ambiguous) English sentence explaining the rule.

What we need to do now is tell RuleGen when this business rule needs to be validated. You need to specify what type of transaction could violate this rule, which is not too difficult for this rule: we will have to investigate inserts, deletes, and updates (on the involved columns) of the involved tables:

1. Whenever the threshold value for a **bookstore** is decreased, we'll have to check the rule for the bookstore (and all dates at which orders exist for this bookstore);
2. Whenever a **book order** is entered, we'll have to check the rule for the bookstore and the date for which the order was entered;
3. Whenever the net price for an existing **book order** is increased, we'll have to check the rule for the bookstore and the date of the order;
4. And finally, whenever the store_id or order_date columns of an existing **book order** are updated, we'll have to check the rule for the new store_id / order_date values.

We now have to transform the above into four queries: one for the bookstore table and three for the book order table. Here's the first one identifying the update to bookstore:

```
select store_id -- rule needs to be revalidated for this store
from updated_rows -- This view is provided by RuleGen
where new_daily_net_value_threshold < old_daily_net_value_threshold
```

And here are numbers 2, 3, and 4 identifying the type of changes to the book order table that require rule validation:

```
select store_id
, trunc(order_date) as order_date -- Revalidation for this store + date
from inserted_rows -- View provided by RuleGen
```

As events 3 and 4 both concern updating the book_order table, we must combine this into one query specification for RuleGen as follows:

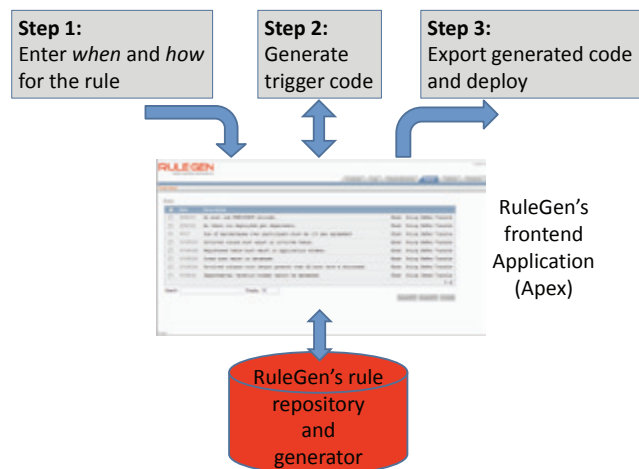
```
select store_id, trunc(order_date) as order_date
-- Revalidation for this store + date combination
from updated_rows
where new_net_price > old_net_price
union
select new_store_id as store_id
, trunc(new_order_date) as order_date
from updated_rows
where new_store_id != old_store_id
or trunc(new_order_date) != trunc(old_order_date)
```

Note that in the first case the query only produces a store_id value, whereas in cases 2, 3, and 4, a store_id and an order_date (trunc'd) are produced. Our final step now is to provide RuleGen with two revalidation queries: one that accepts a store_id bind variable and one that accepts both a store_id and order_date bind variables. These queries should be such that they produce the error messages that should be raised in case our business rule gets violated. Here they are:

```
select 'Cannot decrease threshold. Violation found for store '||
p_store_id||' at date'||order_date||'.' as msg
from (select s.store_id
, trunc(o.order_date) as order_date
, sum(o.net_price) as sum_price
, s.daily_net_value_threshold
from book_store s
, book_order o
where s.store_id = o.store_id
group by s.store_id, trunc(o.order_date))
where sum_price > daily_net_value_threshold
and store_id = p_store_id -- Bind variable here
select 'Cannot decrease threshold. Violation found for store '||
p_store_id||' at date'||p_order_date||'.' as msg
from (select s.store_id
, trunc(o.order_date) as order_date
```

```
, sum(o.net_price) as sum_price
, s.daily_net_value_threshold
from book_store s
, book_order o
where s.store_id = o.store_id
group by s.store_id, trunc(o.order_date))
where sum_price > daily_net_value_threshold
and store_id = p_store_id -- Bind variable here.
and trunc(order_date) = p_order_date -- Bind variable here.
```

With the above five queries, we are done specifying this rule for RuleGen. All we need to do now is press a button to generate all necessary trigger code. These queries and the generated code for our business rules are maintained in RuleGen's rule repository. The generated code can simply be exported into a script file that we can then keep in our source code control system and deploy on the target database environment.



For more examples, check out www.rulegen.com.

Conclusion

With RuleGen, the task of managing hundreds of rules in our application's database design becomes straightforward. Rules can easily be added, dropped, or changed and then redeployed. RuleGen not only generates efficient code, the code generated also caters to various manipulations of rules, just as can be done with declarative constraints: you can enable/disable rules, set rules to be deferred (!), log errors that rules generate, and even control the order in which they should "fire." And on top of that, the code also uses the DBMS_LOCK package to acquire locks for correctly serializing the code.

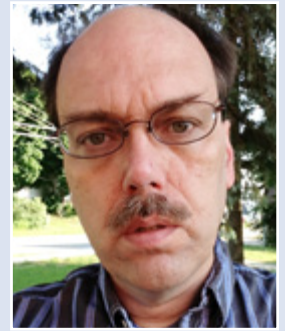
RuleGen abstracts technology (complex database trigger and package code, circumventing mutating table issues, and serializing code execution) for us from functional requirements: we simply specify our business rules through a few queries, and at the push of a button RuleGen generates all necessary PL/SQL code to enforce them. It is easy to learn and very intuitive to use. ▲

Mark Rooijakkers has been developing applications since 1993 using Oracle Forms, Oracle Designer, APEX, PL/SQL, and JDeveloper. He is currently a technical architect at CB-Logistics, Netherlands.

Copyright © 2013, Mark Rooijakkers

Wielding the Sword of Analytics

by Jonathan Gennick



Jonathan Gennick

Our brethren in SQL Server land have just gained a full implementation of window functions, termed *analytic functions* in Oracle Database, with the recent release of SQL Server 2012. Having a new constituency brought to mind the challenges involved in wielding these useful functions. Learning the syntax and mechanics of how they operate is the easy problem. The greater challenge in my mind lies in helping developers recognize when these wonderfully expressive functions can simplify what otherwise is a tough query problem.

Summary and Detail Together

The classic problem that analytic functions solve extremely well is the need to mix and compare summary and detail results in the same query. Imagine that you're an analyst for a department store chain. The sample data can be found at gennick.com/WindowExampleData.sql. Here's an example of a question that you might be faced with:

In what quarters were appliance sales below average for the calendar year?

Answering this question calls for quarterly sales data, so begin by writing a simple query to list appliance sales by quarter:

```
SELECT year, quarter, sales
FROM quarterly_sales
WHERE department = 'Appliances'
ORDER BY year, quarter;
```

YEAR	QUARTER	SALES
2000	1	235956.23
2000	2	129871.95
2000	3	398213.39
2000	4	54111.05
2001	1	87230.33
2001	2	105231.94
...		

Next you need to compare each quarterly sales result against an average value for the year. For that, you need a fourth column in the query output to reference in a `WHERE` clause. Begin by writing an aggregate function to generate an average sales amount:

```
SELECT year, quarter, sales,
       AVG(sales)
...
```

Next is where the magic begins. Add the keyword `OVER` to indicate an analytic function:

```
SELECT year, quarter, sales,
       AVG(sales) OVER
...
```

For each detail row, you want the average sales for that row's department and year. The `PARTITION BY` clause added next gives you that result:

```
SELECT year, quarter, sales,
       AVG(sales) OVER (
         PARTITION BY department, year)
...
```

Two housekeeping details remain: 1) Round average sales to the nearest cent, and 2) Add a column alias. Here's the final version:

```
SELECT year, quarter, sales,
       ROUND(AVG(sales) OVER (
         PARTITION BY department, year),2) avg_sales
...
```

Following is an execution of the new query and some of the results:

```
SELECT year, quarter, sales,
       ROUND(AVG(sales) OVER (
         PARTITION BY department, year),2) avg_sales
FROM quarterly_sales
WHERE department = 'Appliances'
ORDER BY year, quarter;
```

YEAR	QUARTER	SALES	AVG_SALES
2000	1	235956.23	204538.16
2000	2	129871.95	204538.16
2000	3	398213.39	204538.16
2000	4	54111.05	204538.16
...			

Each quarterly sales result includes the average quarterly sales for the year. You have all the data needed—summary and detail—to answer the business question. Just feed that data into an enclosing query having an appropriate `WHERE` clause. Here's the final query and some of the results:

```

SELECT *
FROM
(
  SELECT year, quarter, sales,
         ROUND(AVG(sales) OVER (
           PARTITION BY department, year)2) avg_sales
  FROM quarterly_sales
  WHERE department = 'Appliances'
) x
WHERE sales < avg_sales
ORDER BY year, quarter;

```

YEAR	QUARTER	SALES	AVG_SALES
2000	2	129871.95	204538.16
2000	4	54111.05	204538.16
2001	1	87230.33	114948.91
...			

The inner query generates the raw data, and the outer query filters that data to return the rows answering the original business question. You need this two-step process because analytic functions are evaluated late in a query's execution, after any filtering by the `WHERE` and `HAVING` clauses.

Values from Other Rows

Functions such as `LAG` and `LEAD` enable you to treat a row set as a two-dimensional entity. Imagine that you have quarterly sales by department in your data mart. You decide to compare quarterly sales year-on-year to answer the following question:

In what quarters did a department's sales drop below those of the same quarter one year previously?

In other words, if a department sold \$100,000 in Quarter 2 of 2005 and then only \$95,000 in Quarter 2 of 2006, you want to know about that, because it represents a year-on-year drop that bears further investigation.

Following is a simple query showing the available data:

```

SELECT department, year, quarter, sales
FROM quarterly_sales
ORDER BY department, year, quarter;

```

DEPARTMENT	YEAR	QUARTER	SALES
Appliances	2000	1	235956.23
Appliances	2000	2	129871.95
Appliances	2000	3	398213.39
...			

`LAG` provides easy access to a value from a prior row. It's perfect for working with time-series data in which the time interval from row to row is consistent. There are four quarters in a year, so there are four rows per combination of year and department. Write a `LAG` function to retrieve the fourth sales amount back:

```

SELECT department, year, quarter, sales,
       LAG(sales, 4) year_ago_sales
...

```

For `LAG` to be useful, you need a data-cleansing step to eliminate any gaps. I've already ensured that the example data represents a continuous run of quarterly data, with no missing rows.

Going back four rows means nothing without some sort of ordering. Add an `OVER` clause to the function and specify an ordering that follows the calendar:

```

SELECT department, year, quarter, sales,
       LAG(sales, 4) OVER (
         ORDER BY year, quarter) year_ago_sales
...

```

Then partition by department so as to avoid commingling results from different departments:

```

SELECT department, year, quarter, sales,
       LAG(sales, 4) OVER (
         PARTITION BY department
         ORDER BY year, quarter) year_ago_sales
...

```

Now execute the query. Each row of output contains the current quarterly sales along with the corresponding value from four quarters—i.e., one year—before. For example:

```

SELECT department, year, quarter, sales,
       LAG(sales, 4) OVER (PARTITION BY department
         ORDER BY year, quarter) year_ago_sales
FROM quarterly_sales
ORDER BY department, year, quarter;

```

DEPARTMENT	YEAR	QUARTER	SALES	YEAR_AGO_SALES
Appliances	2000	1	235956.23	
Appliances	2000	2	129871.95	
Appliances	2000	3	398213.39	
Appliances	2000	4	54111.05	
Appliances	2001	1	87230.33	235956.23
Appliances	2001	2	105231.94	129871.95
...				

There's no history prior to 2000, so rows from that year receive a null in the `YEAR_AGO_SALES` column. Subsequent rows pick up the sales amount from the same quarter and department one year earlier. Feed these rows into a parent query that answers the business question:

```

SELECT *
FROM
(
  SELECT department, year, quarter, sales,
         LAG(sales, 4) OVER (
           PARTITION BY department
           ORDER BY year, quarter) year_ago_sales
  FROM quarterly_sales
) x
WHERE sales < year_ago_sales
ORDER BY department, year, quarter;

```

DEPARTMENT	YEAR	QUARTER	SALES	YEAR_AGO_SALES
Appliances	2001	1	87230.33	235956.23
Appliances	2001	2	105231.94	129871.95
...				
Electronics	2003	2	624831.23	673839.83
Electronics	2003	3	634839.23	665839.73
...				
Jewelry	2002	2	323939.83	364983.54
Jewelry	2003	3	383542.32	387213.39
...				

The `WHERE` clause predicate in the outer query—`sales < year_ago_sales`—should be clear enough as written. The presence of nulls when no prior history is available is enough on its own to prevent any year-2000 data from cluttering the results. The `ORDER BY` clause is now in the outer query and serves to present the results in calendar order by department. You can readily see when a given department has suffered a drop in year-on-year quarterly sales.

Ranking and Numbering

Functions such as `ROW_NUMBER`, `RANK`, and `DENSE_RANK` assign rankings to rows in a result set. Look to them anytime you are faced with a business question involving words or phrases such as “topmost” or “bottommost,” “top N” or “bottom N,” or that is otherwise answerable by ranking the rows in a result set according to some criteria you can apply to one or more columns of data.

For example, you might be asked the following:

What are three examples of poor quarterly performance for each department?

Using `ROW_NUMBER`, you can rank quarterly sales records in order from poorest to best performance. Then you can peel off three of the worst examples by department. Here’s a `ROW_NUMBER` invocation to do that numbering:

```
ROW_NUMBER() OVER
(PARTITION BY department
ORDER BY SALES) rank
```

Here’s an explanation of what this function is doing:

1. Rows for each department are treated separately. (the `PARTITION BY department` clause)
2. Each row for a department is given a number, beginning at 1. (the `ROW_NUMBER()` invocation)
3. Row number 1 is the row with lowest sales for a given department. Row numbers increase as sales increase. (the `ORDER BY sales` clause)

Here’s an example showing some of the results:

```
SELECT department, year, quarter, sales,
ROW_NUMBER() OVER
(PARTITION BY department
ORDER BY sales) rank
FROM quarterly_sales;
```

DEPARTMENT	YEAR	QUARTER	SALES	RANK
Appliances	2000	4	54111.05	1
Appliances	2001	1	87230.33	2
Appliances	2002	1	95678.13	3
Appliances	2005	3	96078.73	4
Appliances	2001	4	100010.13	5
...				
Electronics	2000	1	345873.14	1
Electronics	2000	2	401144.56	2
Electronics	2000	3	454987.33	3
Electronics	2001	1	467043.14	4
Electronics	2000	4	476539.23	5
...				

Ties in the data call for a more sophisticated approach than just numbering the rows in sequence. You can generate some ties by amending the business question as follows:

Round all sales numbers to the nearest \$10,000. What are the three worst quarters for each department, ties included?

`RANK` is your secret weapon here. `RANK` respects ties by assigning them the same rank number. Substitute `RANK` in place of `ROW_NUMBER`. Invoke `ROUND` as required. Done! Here is the new query and results:

```
SELECT * FROM
(
SELECT department, year, quarter,
ROUND(sales,-4) sales,
RANK() OVER
(PARTITION BY department
ORDER BY round(sales,-4)) rank
FROM quarterly_sales
) bottom_three
WHERE rank <= 3;
```

DEPARTMENT	YEAR	QUARTER	SALES	RANK
Appliances	2000	4	50000	1
Appliances	2001	1	90000	2
Appliances	2001	4	100000	3
...				
Mattresses	2000	1	30000	1
Mattresses	2000	4	30000	1
Mattresses	2000	3	40000	3
...				

There is still a quirk to consider. Look at Mattresses. Why the jump from rank number 1 to rank number 3? What happened to rank number 2?

The two rows ranked as 1 are properly indicated as tying for first place. The third row is properly indicated as being third. It is third because it is preceded by two other rows.

Sometimes you prefer a different point of view. You might not want to derail a presentation to management with an explanation of how the `RANK` function works. `DENSE_RANK` is your friend now. You can use `DENSE_RANK` in the same manner as `RANK`, but the difference is that `DENSE_RANK` eliminates gaps in the numbering. That pesky jump from 1 to 3 will be gone, and that’s one less detail for you to have to explain when presenting the data.

The following query uses `RANK` just as before, but this time only in the selection criteria. A new column created by `DENSE_RANK` is added for display purposes.

```
SELECT department, year, quarter, sales, drank FROM
(
SELECT department, year, quarter,
ROUND(sales,-4) sales,
RANK() OVER
(PARTITION BY department
ORDER BY ROUND(sales,-4)) rank,
DENSE_RANK() OVER
(PARTITION BY department
ORDER BY ROUND(sales,-4)) drank
FROM quarterly_sales
) bottom_three
WHERE rank <= 3;
```

Here are some results:

DEPARTMENT	YEAR	QUARTER	SALES	DRANK
Appliances	2000	4	50000	1
Appliances	2001	1	90000	2
Appliances	2001	4	100000	3
...				
Mattresses	2000	1	30000	1
Mattresses	2000	4	30000	1
Mattresses	2000	3	40000	2
...				

Rows are the same as before due to the continued use of `RANK` in the selection criteria. You still get the bottom three performing quarters by department, including ties. However, the `SELECT` list is changed to show the results from `DENSE_RANK`, so you won’t

see any gaps in the rankings when you view the results. Rankings now move smoothly from 1 to 2 to 3, with no gaps.

Moving Frames of Reference

The *framing clause* is ideal when you can arrange a business question such that an answer comes from applying an aggregate function to a range of rows sliding or stretching smoothly as focus moves from one row to the next. Imagine that you wish to produce a running sum of quarterly sales by department. Following is a query and some results. Glance over the query; then read the detailed explanation following the output.

```
SELECT department, year, quarter, sales,
       SUM(sales) OVER (
         PARTITION BY department
         ORDER BY year, quarter) running
FROM quarterly_sales
ORDER BY year, quarter, department;
```

DEPARTMENT	YEAR	QUARTER	SALES	RUNNING
Appliances	2000	1	235956.23	235956.23
Electronics	2000	1	345873.14	345873.14
Jewelry	2000	1	240939.33	240939.33
Mattresses	2000	1	34873.33	34873.33
Vacuums	2000	1	65873.32	65873.32
Appliances	2000	2	129871.95	365828.18
Electronics	2000	2	401144.56	747017.7
Jewelry	2000	2	304984.38	545923.71
Mattresses	2000	2	47383.93	82257.26
Vacuums	2000	2	56983.43	122856.75
Appliances	2000	3	398213.39	764041.57
Electronics	2000	3	454987.33	1202005.03
...				

Look at the three rows for Appliances. Notice that the running sum progresses from 235,956.23 to 365,828.18, to 764,041.57. Each subsequent Appliances row adds that row's sales amount to the running sum. What's the magic? How does the query operate? Let's walk through it:

1. We want a running sum, so we begin by invoking the aggregate function to sum the sales amounts:

```
SUM(sales)
```

2. The running sum should be computed separately for each department:

```
SUM(sales) OVER (
  PARTITION BY department
```

3. And of course, the running sum should be computed in order by year and quarter:

```
SUM(sales) OVER (
  PARTITION BY department
  ORDER BY year, quarter
```

4. Now for a tricky bit. The frame of reference for each row should begin at the beginning:

```
SUM(sales) OVER (
  PARTITION BY department
  ORDER BY year, quarter
  RANGE BETWEEN
    UNBOUNDED PRECEDING
```

5. ... and end at the current row:

```
SUM(sales) OVER (
  PARTITION BY department
  ORDER BY year, quarter
  RANGE BETWEEN
    UNBOUNDED PRECEDING AND CURRENT ROW) running
```

6. But! It turns out that `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` is default behavior triggered by the `ORDER BY` clause. It is common to omit the phrasing and simplify the function call to just:

```
sum(sales) over (
  partition by department
  order by year, quarter) running
```

You can execute the preceding example query using the invocation of `SUM` from either Step 5 or Step 6. You'll get the same result both ways.

The *moving average* is another widely used tool in business, providing a way to smooth out and get a better sense of a trend over time. You might decide to compute a moving average involving five quarters: the current quarter of focus plus two quarters in either direction. Thus, the following five rows would determine the moving average reported for year 2000, quarter 3:

2000	1	235956.23
2000	2	129871.95
2000	3	398213.39
2000	4	54111.05
2001	1	87230.33

$(235956.23 + 129871.95 + 398213.39 + 54111.05 + 87230.33) / 5 = 181076.59$ = Moving Average for year 2000, quarter 3, centered over five quarters.

Following is the window function invocation to compute such an average:

```
AVG(sales) OVER (
  PARTITION BY department
  ORDER BY year, quarter
  ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
```

The framing clause here is `ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING`. For each *current row*, the average sales amount is computed using the sales values from the two prior and two subsequent rows. The result is a five-value moving average centered on the row for which the function is being evaluated.

You've seen that a framing clause can be written with `RANGE BETWEEN` or `ROWS BETWEEN`. What's the difference? Here it is in a nutshell:

- Use **ROWS BETWEEN** to specify an exact number of rows preceding or rows following. Take care to sort input values deterministically. Endpoints such as `UNBOUNDED PRECEDING`, `UNBOUNDED FOLLOWING`, and `CURRENT ROW` refer to specific *single* rows.
- Use **RANGE BETWEEN** when values are sorted ambiguously. Endpoints such as `UNBOUNDED PRECEDING`, `UNBOUNDED FOLLOWING`, and `CURRENT ROW` refer to *groups* of rows that all sort to the same position because they share the same sort key values.

Don't allow your thinking to stop at running sums and moving averages. Explore what the other functions have to offer. You

can get creative with `MAX` and `MIN`, for example, to identify new highs and lows. Imagine the following question:

In what quarters do the sales represent a new high for the given department?

To identify a new high, you must compare the current quarter's sales amount against the highest amount from all past quarters. As the current quarter advances, the list of past quarters stretches and grows longer. This shifting frame of reference is your clue to think about the framing clause.

Following is a query invoking `MAX` with a framing clause to answer the preceding question:

```
SELECT department, year, quarter, sales,
       MAX(sales) OVER (
         PARTITION BY department
         ORDER BY year, quarter
         ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING
       ) prev_max
FROM quarterly_sales
ORDER BY department, year, quarter;
```

DEPARTMENT	YEAR	QUARTER	SALES	PREV_MAX
Appliances	2000	1	235956.23	
Appliances	2000	2	129871.95	235956.23
Appliances	2000	3	398213.39	235956.23
Appliances	2000	4	54111.05	398213.39
Appliances	2001	1	87230.33	398213.39
...				
Electronics	2000	1	345873.14	
Electronics	2000	2	401144.56	345873.14
Electronics	2000	3	454987.33	401144.56
Electronics	2000	4	476539.23	454987.33
Electronics	2001	1	467043.14	476539.23
...				

It's now a simple matter to wrap this query in an outer query that filters on the condition `(SALES > prev_max)` or `(prev_max IS NULL)`. If you want to look at new highs over, say, a five-year period rather than over the entire history, you can rewrite the framing clause as `ROWS BETWEEN 60 PRECEDING AND 1 PRECEDING`. Replace `MAX` with `MIN`, and you can search out new lows. The framing clause is a great tool to have in your kit.

Summary

Analytic functions are a wonderful part of SQL. Their expressive power makes many previously difficult problems trivial and even fun to solve. Recognizing when to use them is sometimes the challenge. Keep in mind the different use cases described in this article, and you'll be well on your way to wielding these functions appropriately and with great success. ▲

Jonathan Gennick is an Assistant Editorial Director at Apress with responsibility for database topics.

Copyright © 2013, Jonathan Gennick



Oracle Database Administration, Development, and Performance Tuning... Only Faster.

Taking care of your company's data is an important job. Making it easier and faster is our's.

Introducing DB PowerStudio for Oracle. It provides proven, highly-visual tools that save time and reduce errors by simplifying and automating many of the complex things you need to do to take care of your data, and your customers.

Whether you already use OEM or some other third-party tool, you'll find you can do many things faster with DB PowerStudio for Oracle.

- > Easier administration with DBArtisan™
- > Faster development with Rapid SQL™
- > Faster performance with DB Optimizer™
- > Simplified change management with DB Change Manager™

Go Faster Now. >>> Get Free Trials and More at www.embarcadero.com

Don't forget Data Modeling! Embarcadero ER/Studio®, the industry's best tool for collaborative data modeling.

© 2011 Embarcadero Technologies, Inc.
All trademarks are the property of their respective owners.

embarcadero

Oracle Professional Consulting and Training Services

Certified training and professional consulting when you need it, where you need it.



www.quilogyservices.com
education@aspect.com
 866.784.5649

SPECIAL FEATURE (continued from page 15)

As you can probably understand, the task of managing all required trigger code in a typical application with dozens, if not more than 100, assertions can become daunting. You should realize, though, that the vast majority of the required trigger code can be generated. Once you have figured out the TE queries and the validation query, all code required to implement an assertion can be generated.


That then leaves us—or, rather, the DBMS vendor—with some final questions: Is it possible to “compute” the TE queries and the validation query, given only the assertion text? Can an algorithm be developed that parses an assertion and then figures out only when (i.e., during what type of dml changes) and how to most efficiently validate the assertion? On researching this topic you’ll find that this challenge is, at its core, the very same challenge that a DBMS vendor has when it tries to figure out how to “fast refresh” any given materialized view. So on the one hand, a lot of code should already be available to help implementing assertions. On the other hand, though, we are still far away from support of fast refresh for any given materialized view. A lot of restrictions still apply. So keep an eye on these restrictions becoming fewer and fewer. ▲

Toon Koppelaars has more than a quarter century of relational database experience and is the creator of RuleGen, a utility for automating the generation of assertion code in a relational database. He is currently a software development manager for Oracle Netherlands working on Oracle Health Insurance products. The views expressed in this article are his own and do not necessarily reflect the views of Oracle Corporation. © 2013, Toon Koppelaars



Database Virtualization Software

- Consolidate Infrastructure.
- Instantly Provision and Refresh.
- Maximize Performance.



www.delphix.com

BOOK REVIEW (continued from page 18)


which we can see a user accessing WLS, for example, directly. I would then cover all the pieces installed for OIM, OAM, and OID, and then show the complete flow of user access with OIM, OAM, and OID in place. Showing (again) the log files, we would learn the access flow and where to see it, so we can troubleshoot issues when they occur.

Implicit in this is a lot more coverage of Oracle Internet Directory (OID), which is Oracle’s implementation of LDAP. I think some amount of LDAP information is needed to explain how OIM and OAM work. I would also provide a title for each figure in this book. The text associated with each figure needs to refer to the figure explicitly to help the reader understand what is being discussed.

Overall this book has value, but I don’t think it is worth the \$60 list price. A significant fraction of the pages are filled with screenshots, without enough accompanying text to provide the insights I would expect from a book on this topic. ▲

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.

Copyright © 2013, Brian Hitchcock




Nothing hunts down Oracle performance issues like **Confio Ignite™**

Over 50% of DBAs who try Ignite resolve a performance problem on the first day.

Start your **free trial** at **Confio.com**

(303) 938-8282
© 2013 Confio Software



Database Specialists: DBA Pro Service



DBA PRO BENEFITS

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

CUSTOMIZABLE SERVICE PLANS FOR ORACLE SYSTEMS

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

DBA Pro's mix and match service components

Access to experienced senior Oracle expertise when you need it

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

24 x 7 availability with guaranteed response time

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

Daily review and recommendations for database care

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

Monthly review and report

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

Proactive maintenance

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

Onsite and offsite flexibility

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



CALL 1 - 8 8 8 - 6 4 8 - 0 5 0 0 TO DISCUSS A SERVICE PLAN

NoCOUG Summer Conference Schedule

Thursday, August 15, 2013—Chevron, 6101 Bollinger Canyon Road, San Ramon, CA

Please visit <http://www.nocoug.org> for updates and directions, and to submit your RSVP.

Cost: \$50 admission fee for non-members. Members free. Includes lunch voucher.

8:00–9:00 a.m.	Registration and Continental Breakfast—Refreshments served
9:00–9:30	Welcome: Naren Nagtode, NoCOUG president
9:30–10:30	Keynote: <i>Soul-Searching for the Relational Movement: Why NoSQL and Big Data Have Momentum</i> —Iggy Fernandez
10:30–11:00	Break
11:00–12:00	Parallel Sessions #1 Room 1220: <i>Analyze This! Analytical Power in SQL—More Than You Ever Dreamed Of</i> —Hermann Baer, Oracle Corp. Room 1240: <i>Creative Wait Interface Maneuvers: Fast Performance Problem Resolution</i> —Craig Shallahamer, OraPub Room 1150: TBD
12:00–1:00 p.m.	Lunch
1:00–2:00	Parallel Sessions #2 Room 1220: <i>Oracle Partitioning in Oracle Database 12c: It's Getting Even Better</i> —Hermann Baer, Oracle Corp. Room 1240: <i>Introduction to Time-Based Performance Analysis: Stop the Guessing!</i> —Craig Shallahamer, OraPub Room 1150: <i>MySQL Technology Update for Oracle DBAs and Developers</i> —Lynn Ferrante Howells, Oracle Corp.
2:00–2:30	Break and Refreshments
2:30–3:30	Parallel Sessions #3 Room 1220: <i>Using Resource Manager to Consolidate Databases in Oracle Database 12c</i> —Sue Lee, Oracle Corp. Room 1240: <i>ASH: Architecture and Advanced Usage</i> —John Beresiewicz, Oracle Corp. Room 1150: <i>HBase and Lewis Carroll</i> —Jeff Bean, Cloudera
3:30–4:00	Raffle
4:00–5:00	Parallel Sessions #4 Room 1220: <i>Heat Map and Automatic Data Optimization in Oracle Database 12c</i> —Gregg Christman, Oracle Corp. Room 1240: <i>Hey Oracle Optimizer! Don't Mess with My Plans</i> —Janis Griffin, Confio Software Room 1150: <i>NoSQL Drill-Down: So What's a Graph Database?</i> —Philip Rathle, Neo Technology
5:00–	NoCOUG Networking and No-Host Happy Hour

RSVP *required* at <http://www.nocoug.org>