Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 23, No. 3 · AUGUST 2009 $15

Let Knowledge Ripen at NoCOUG

Much more inside . . .

Let Knowledge Ripen at NoCOUG

The *First International NoCOUG SQL Challenge* was a great success; nine solutions were found by participants in seven countries and three continents. Alberto Dell'Era wins the contest for his wonderful solution using Discrete Fourier Transforms; the runner-up is André Araujo from Australia, who used binary arithmetic and common table expressions in his solution. The *August Order of the Wooden Pretzel* will be bestowed on Alberto but the real prize is six books of his choice from the Apress catalog. André will receive a prize of six e-books of his choice. Thanks to Chen Shapira for publicizing the event in her blog, Dan Tow for helping to judge the contest, and Apress for donating the books.

The complete results and a detailed analysis of the winning solution can be found on page 15. I hope you enjoy this edition of the *NoCOUG Journal*. Join me to hear Donald Burleson, Dan Morgan, Daniel Liu, Tim Gorman, and other great speakers at the summer conference on August 20. ▲

—Iggy Fernandez, *NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

2009 NoCOUG BOARD

Publication Notices and Submission Format

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

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

The submission deadline for the upcoming November 2009 issue is August 31, 2009. Article submissions should be made in Microsoft Word format via email.

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

Human Factors Affect Successful Change

Hanan Hit

by Hanan Hit

"The productivity of knowledge work—still abysmally low—will become the economic challenge of the knowledge society. On it will depend the competitive position of every single country, every single industry, every single institution within society."
—Peter Drucker

The economic recession creates greater opportunities for Oracle professionals as companies strive to improve their operational processes and raise their efficiencies by combining newer technology with greater staff expertise. Stephen Elliot, IT senior analyst with IDC, showed that on average 80% of IT-system outages are caused by operator and application errors. With the tolerance of such errors diminishing, there is a growing demand for those who can implement the most recent Oracle technologies to reduce operational cost and improve business application availability.

Lately I have been studying the concepts and policies recommended by the Information Technology Infrastructure Library (ITIL), and I was amazed that many organizations are still inefficient and ineffective. Rather than analyzing risks, organizations tend to avoid them altogether based on technical assumptions that have not been validated, preferring intuitions over methodical research. Planning is often inadequate due to lack of knowledge—and the outcome is IT projects that cost more and deliver less than expected.

Successful projects result from application of methodical problem-solving skills based on deep understanding of how the systems being deployed function. Such problem-solving skills include, among others, the ability to identify and eliminate database bottlenecks and the ability to develop efficient SQL statements. Adequately preparing for an Oracle project greatly increases the chance of success and reflects positively on the organization as well as our career. Preparing and planning will help. Successful projects boost the contributor's career—both directly and by contributing to the organization's revenue and improving its competitive advantage. NoCOUG can assist in all these areas. Use NoCOUG as a resource in your research and information-gathering stage. At NoCOUG conferences you can network with others who may have already completed a similar task. The *NoCOUG Journal* and website are also valuable sources of information.

The NoCOUG Summer Conference will be held on August 20 in San Ramon. Donald Burleson, a world-renowned presenter will kick off the day with a keynote presentation, "Oracle 2020—A Look at How Oracle Will Change in the Next Decade," followed by a presentation about self-tuning databases. The day will be filled with technical presentations by staff from Oracle Corporation as well as other renowned presenters such as Tim Gorman, Daniel Morgan, and Iggy Fernandez, and real Oracle users like you and me. The day will be rounded out with the usual networking opportunities, book raffles, and plenty of food and drink. Get all of the conference details and submit your registration on the NoCOUG website at **www.nocoug.org**. I hope to see you there. ▲

Great Expectations

An Interview with Tanel Poder

Tanel Poder

Tanel Poder is an experienced consultant with deep expertise in Oracle database internals, advanced performance tuning, and end-to-end troubleshooting. He specializes in solving complex problems spanning multiple infrastructure layers such as Unix, Oracle, application servers, and storage. He hosts an Oracle performance tuning blog at **blog.tanelpoder.com** *and has published useful Oracle troubleshooting tools like PerfSheet, LatchProfX, and Session Snapper. In addition to consulting and training, Tanel frequently speaks at major conferences such as OracleWorld, UKOUG, Hotsos Symposium, IOUG-A Live, and many regional ones. Tanel is one of the first Oracle Certified Masters in the world, an Oracle ACE Director, and an OakTable Network member.*

Tell us something about yourself.

I am an IT enthusiast, fascinated by the opportunities of using technology to improve our productivity and our lives. I'm focused on getting the most out of information technology, both for my clients and for me.

I started my professional career at age 16 as a part-time C developer (I was not too good at it back then, because I lacked software engineering experience). I got introduced to Unix, though, which gave me a good foundation for understanding Oracle and enterprise-class applications when I started working for a consulting company at age 19.

By now I have over 12 years of experience managing and improving Oracle-based database and application environments, plus troubleshooting; tuning; and delivering training classes for user groups, Oracle Education, and partners. Currently I'm particularly focused on researching Oracle 11*g* internals and its touch point with Unix/Windows for gaining fast and accurate troubleshooting skills—and, of course, for getting the most out of its new features.

I took the first public Oracle Certified Master DBA exam in the Oracle Education Center in Chicago (2002) and became the 25th–27th OCM in the world. The range of numbers is because previously there had been two internal beta exams for Oracle employees and 24 people were certified already; also a few people (like Dave Ensor) had received an honorary OCM certificate. There were four people taking the exam, but one of them, unfortunately, lost some data during a simulated crash, and a DBA cannot be called a master if they fail to deliver the most fundamental part of their work—retaining the data. So, 25th–27th it is (smile).

In 2004 I was invited to join the OakTable Network during the Miracle Database Forum. I think this is the greatest professional honor and recognition I've ever received, and I proudly accepted (after having to justify why MS-SQL Server is much better than Oracle to 10+ OakTable members in a public debate).

In April 2003 I also got my first international conference speaking experience from IOUG-A Live in Orlando, FL. I spoke about how to do major Oracle E-Business Suite upgrades with minimum downtime for the business. I had about 15 people attending my session in a big 300+ capacity room. But it looked like they liked it and I sure liked it. Ever since I've been speaking at conferences three to five times per year, depending on how busy (and lazy) I have been during that particular year.

And now I'm blogging. I hope to give something back to the community, as I have gained so much from it. But also I hope this blog becomes a good way to keep in touch with friends around the world in today's extremely busy life— which we'll definitely improve, eventually, as we throw even more technology at it (smile).

I'm now also an Oracle ACE Director.

What's an Oracle ACE director?

Oracle ACE director status is Oracle Corporation's official recognition for Oracle community members who have a track record of being community advocates on Oracle-related topics and who have advanced the Oracle technology field with their blogs, publications, and conference presentations. One difference between Oracle ACE and ACE director is that Oracle actively involves ACE directors in getting feedback and ideas about product usage and development directions.

You started your IT career as a programmer and went on to become a database administrator. Here is part of an email message that I received from a reader: **"I purchased your book in order to embark on a self-study of Oracle in order to pursue a career as a DBA. My education background is of a Bachelor Degree in Information Systems (Honors) from UK. The problem is I have no background in programming and my professional experience [is] as a business analyst."** ***Should lack of programming experience stop someone from pursuing a career as a database administrator?***

There are many DBAs out there without programming background. Understanding programming concepts and the

ability to write database maintenance scripts will definitely help a lot, but nowadays this is not a prerequisite for starting as a junior DBA. If you want to end up a senior DBA, then be prepared to learn some scripting language such as some Unix shell or Perl. The good news is that the database maintenance scripts usually don't require advanced programming skills such as multithreading, objects, etc.

Also, you could use your background as a business analyst to your advantage. If you aim toward being a *development DBA* or database designer, business analysis and knowledge of your company's business can be helpful as you can communicate with end users more easily and add more value.

Certification can be a costly pursuit. In addition to the work absences, there's the cost of instructor-led training courses; exam fees; and travel, board, and lodging. How much did you spend in acquiring the OCM credential? Was it a good investment? Would you recommend certification to other Oracle professionals? Should employers favor professionals with certification?

The total cost was around $6000. One-third of it was the exam fee, another third went for the two advanced courses I had to take as a prerequisite for the exam, and rest went on the trip from Europe to Chicago, as back in 2002 that was the only exam center giving OCM exams. This number excludes the costs involved taking the prerequisite OCP exams and, of course, all the learning time.

But yes, I think OCM was a good investment, especially as I could keep mentioning that I was one of the first OCMs out there (smile).

When looking for contract roles in the past, I had to explain to all recruiters what the heck OCM means, but once my resume passed them, I think the credential made a difference. I was definitely invited to the interview and the techies were eager to have a conversation with me.

Of course, at the interview table, you must back up what your credentials say—answer all the questions, even if some answers are simply, "I don't know, but I know how to find out." It doesn't matter how many certificates or diplomas you have: if you can't answer your job-related questions at the interview, you're out.

You won membership in the OakTable network by arguing that SQL Server might sometimes be a better choice than Oracle. Some of those arguments might come as a surprise to those who have never worked with a database technology other than Oracle. Would you care to share some of them with us?

Heh heh, well I think it was already decided by the OakTable *junta* that I'd be in, but they just wanted to give me a hard time and have some good laughs. This was five years ago and I didn't know many of the OakTable members personally yet, so it was a situation to remember. I was standing with another new Oakie-to-be on one side of the room across from 10 existing OakTable members, whom I already knew from their work and had high respect for, answering a long series of questions that they bombarded us with. I was actually nervous but it was also fun at the same time, since I knew they were just teasing us. This was an exam of personality, not technology. I don't remember exact questions anymore, but none of the questions were serious ones (smile).

You've published some free Oracle tools. What do they do?

I think my Oracle Session Snapper should be the first thing to mention. This is a single SQL script containing an anonymous PL/SQL block, which allows you to take snapshots and report *session-level* performance statistics in Oracle. It's very convenient to query where one or more sessions' response time is spent with Snapper plus other statistics like number of executions, logical IOs, and more. The key design principle behind Snapper is that it does not require any change to the

database whatsoever—no temporary tables, no PL/SQL packages, nothing—it's just an anonymous PL/SQL block parsed and executed on the fly. This is especially useful for troubleshooting databases under tight change control.

Another tool of mine is LatchProfX. This is a script that allows you to be more systematic when troubleshooting latch contention issues. Whenever a session is waiting for a latch, it's because some other session is holding it. LatchProfX allows you to easily identify who the latch holders are and why they are holding it. Again, this is an Oracle session-level troubleshooting approach; this is a more universal and reliable way for latch contention troubleshooting than the old system-wide latch sleep-based technique.

Finally, I have written an Excel-based tool called PerfSheet that I use for easy visualization of performance and other data. Basically the idea is that you can define a number of SQL queries in an Excel sheet; from there it takes just a couple of mouse clicks to run them against a database, and the result set is automatically fetched into Excel and visualized for you. There is no need to dump data to CSV and load to Excel; all data transport and visualization is done for you. This reduces many time-consuming (and boring) steps from data analysis and visualization. I have used this tool successfully in multiple troubleshooting and capacity-planning cases and sometimes use it for monitoring as well.

All the Oracle scripts and tools I've written have come out of necessity. I have had some problem I've needed to fix or I've realized I'm wasting too much time doing something manually over and over again. I don't like wasting time on doing things manually, so I've automated some tasks with my scripts or tools. I hope they are useful for you too.

You travel more than 100,000 miles every year to speak at conferences and deliver training. How do you cope? Do you have any travel tips to share? Favorite airports?

As with my work, I optimize my travel too (smile).

For example, when I started taking (too) frequent long-haul flights, I bought good noise-canceling headphones. I can both sleep and concentrate better with these on, as it's much quieter for me. Regarding jet lag, I've realized that it's more important to have a good rest before the flight instead of trying to get myself into the right time zone by skipping a night of sleep before the flight or forcing myself to wake up really early. Also, I try to take long-haul flights that depart in the morning so I can work the whole day on the plane instead of trying to try to get some sleep in flight. Obviously, sleeping in a hotel bed works much better than sleeping on a plane.

My favorite airport is Singapore's Changi airport. It actually feels good to be there. They have won numerous awards for their quality. They even have live piano music and occasional singers there.

Is Unix a better choice than Windows for an Oracle database and—if so—why? In your experience, is Oracle on Windows any less stable or reliable than Oracle on Unix?

By now I have realized that when running on comparable platforms (and Unix, Linux, and Windows are comparable nowadays), the quality of people involved in building and maintaining the system is much more important than the individual details of the underlying platform. With an experienced and motivated team you can build a scalable and working solution on Windows/MSSQL, just like with Unix/Oracle. With an inexperienced or unmotivated team, it doesn't matter how good the infrastructure you use is—you will have problems. The design decisions and coding quality can hurt or benefit the end result more than change from one major platform to another.

From a technical perspective I still like Unix/Linux platforms more than Windows for running Oracle databases, as they tend to have better diagnosability infrastructure available than when running on Windows. Of course you can always download and install additional tools and debuggers, but on Unixes they usually tend to be there. On the other hand, if your company is a 100% Windows shop, go with Windows as you already have Windows experience in-house.

My personal favorite is Solaris due to its extremely good instrumentation and diagnosability tools such as DTrace. This allows me to systematically go deeper in troubleshooting without having to resort to guessing or luck.

You'll be delivering your Advanced Oracle Troubleshooting Seminar at NoCOUG in November. Why should I come to your seminar?

The main reason is that I will provide a systematic approach to troubleshooting, along with the scripts and tools required for it. And the word *Advanced* in the seminar title actually *means* advanced too! I will show you how to drill down extremely deep in case of Oracle instrumentation bugs and complex issues such latch and mutex contention. Another reason is that I won't cover just Oracle database in isolation, I will also talk a lot about how Oracle interacts with operating systems and hardware and how to troubleshoot Oracle from the OS side too. Finally, you will see demos during the *majority* of the seminar time in SQL*Plus, Unix shell, or with few GUI tools of mine. This allows showing how Oracle really works and the troubleshooting techniques in action.

This seminar is concentrated entirely on Oracle troubleshooting—understanding what exactly Oracle is doing right now or what it was doing when the problem occurred. You will gain the skill to systematically work out the reasons for crashes, hangs, bad performance, or other misbehavior.

The seminar takes a holistic approach to end-to-end troubleshooting. It will explain the full lifecycle of a database request, from database client libraries and networks to the Oracle database kernel and the underlying OS. For each layer, a troubleshooting technique is provided along with advice on using the right tool for the right problem at the right time.

The two days are full of intensive learning, reading dumps, stack traces, network packet captures, and Oracle SGA directly. You'll be using debuggers and custom tools provided to you for real-time and post-mortem diagnosis. The emphasis is on practical troubleshooting; safety comes first and many techniques are designed to require no change to database schemas or instance parameters. I hope to see you there (smile). ▲

Interview conducted by Iggy Fernandez

Blind Variable Peeking: Bane or Boon?

Ask the Oracles!

Wolfgang Breitling: Bind variable peeking was introduced by Oracle with 9*i*. Quoting from the *Oracle 9*i *Database Performance Tuning Guide and Reference*:

"The CBO peeks at the values of user-defined bind variables on the first invocation of a cursor. This feature lets the optimizer determine the selectivity of any WHERE clause condition, as well as if literals have been used instead of bind variables. On subsequent invocations of the cursor, no peeking takes place, and the cursor is shared, based on the standard cursor-sharing criteria, even if subsequent invocations use different bind values. When bind variables are used in a statement, it is assumed that cursor sharing is intended and that different invocations are supposed to use the same execution plan. If different invocations of the cursor would significantly benefit from different execution plans, then bind variables may have been used inappropriately in the SQL statement."

To stake my position up front I think for the most part it is a bane. It appears to me that Oracle is trying to fix from the database side sins committed by the application programmer, much like the "cursor_sharing=force" (or, even worse, = similar) parameter setting. In that case Oracle is trying to "posthumously" use bind variable because the programmer neglected to do so.

The central purpose of this feature is to "let the optimizer determine the selectivity of any WHERE clause condition, as well as if literals have been used. . . ."

Let's take a look at this. Where does the difference between using a bind variable or a literal make a difference in the selectivity of a predicate? In this context we exclude predicates on columns with histograms. If the distribution of your column values is skewed such that a histogram is warranted, then you should not use bind variables. That is one of the lemmas derived from that disclaimer in the second paragraph in the quote from the *Performance Tuning Guide*.

We'll look at these types of predicates:

1. Equality predicate: Column = value
2. Like predicate: Column like value
3. Range predicate: Column between low-value and high-value; or, as the optimizer transforms it: "column >= low-value and column <= high-value" where, optionally, either side of the "and" may be omitted (unbounded range) or the ">=" or "<=" operands be just ">" or "<".

We will look at the resulting selectivity by examining the row source cardinality (cardinality = selectivity * num_rows) of three scenarios:

a) Using a literal
b) Using a bind variable with bind variable peeking disabled ("_optim_peek_user_binds" = false)
c) Using a bind variable with bind variable peeking enabled ("_optim_peek_user_binds" = true)

For the tests I use a table created "as select from dba_objects where rownum <= 10000" with a unique index on object_id and an index on owner,object_name.

a) Select count(*) from test where owner = 'SCOTT'

Num_distinct of owner was 9, and in all three cases the estimated cardinality turned out to be 1111 = 10000 / 9. So here bind variable peeking does not make a difference in the cardinality estimate and, therefore, also not for the access plan.

Not for Oracle 9*i*, that is. If you do the test in 10*g* or 11*g* and give a value that is not within the range set by LOW_VALUE <= predicate <= HIGH_VALUE, then the optimizer does not use a selectivity of 1/num_distinct, i.e., .1111 in this case, but "Using prorated density: 0.047437 of col #1 as selectivity of out-of-range value pred." The prorated density value depends on how far away the predicate value is from the low or high value known to the optimizer, but it is always less than the column density and, therefore, the estimated cardinality is less if a literal or bind peeking is used. In the latter case, that can have devastating effects if it happens to be the hard parse that determines the access plan for all. So the developer had done the right thing using a bind variable and "assumed that cursor sharing is intended and that different invocations are supposed to use the same execution plan." But he did not anticipate that the plan would turn out to be a stupid plan for the majority of "within-range" predicates based on an unfortunate and unusual value at hard parse time. I certainly consider this a bane of BV peeking, especially since Oracle changed the rules of the game with the upgrade to 10*g*.

b) Select count(*) from test where object_name like 'A%'
Select count(*) from test where object_name like 'AL%'
Select count(*) from test where object_name like 'ALL%'

These three tests use different lengths for the like string. The cardinality of object_name was 8556 and the estimated cardinalities 245, 2, and 1 with bind peeking and 500 (5% of 10,000) in all three cases with bind peeking turned off.

So here, clearly, bind peeking does make a difference, both for the estimated cardinality and also probably for the result-

ing access plan. However, is it correct to expect that "cursor sharing is intended and that different invocations are supposed to use the same execution plan"? I think this is a clear case where a bind variable should not be used, and thus the question "to peek or not to peek" becomes moot.

c) Select count(*) from test where object_id between 100000 and 200000;
Select count(*) from test where object_id between 10000 and 20000;
Select count(*) from test where object_id between 100 and 200;

Again, three tests that use different range spans. The cardinality of object_id, being the unique key column, is obviously 10,000 and the range is 2 to 232,471, so all the above ranges are within the low-high range, avoiding any complications from selectivity adjustments due to "out-of-bounds" ranges.

The estimated cardinalities with BV peeking are 4304, 432, and 6—exactly following the formula for bounded range predicates (high – low) / (HI – LO) * 10,000—while all three estimates without BV peeking are 25—5% of 5% of 10,000. The upper-case range is that of the column statistics; the lower-case range is that of the predicate. There is also an additional estimate of 2 added to account for the CPU cost part and rounding.

Again bind peeking provides more accurate estimates based on the actual range. The difference between the 0.25% selectivity without and the selectivity with BV peeking is most pronounced, of course, if the predicate range covers a large portion of the column range.

So back to the question "Is bind variable peeking good or bad"? As we have seen, it does not make a difference for equality predicates except in the out-of-bound predicate case in 10*g* and later, and then it is more of a curse than a blessing. Where it makes a difference is with like and range predicates. However, the sharing of the cursor because of the bind variables can easily backfire if the values for different executions are not similar. Remember the disclaimer warning. That, of course, was not the case before BV peeking because of the fixed 5% selectivity per predicate (ranges are two predicates: >= and <= , thus the 5% * 5% = 0.25% selectivity).

In Oracle 9*i*, where BV peeking was introduced, the overall effect was more or less neutral as long as bind variables were used as intended. But in my opinion Oracle 10*g* tipped the scale decidedly to "bane" due mainly to two new features:

a) The already-mentioned special selectivity for "out-of-bound pred," which can suddenly wreck the access plan for all "normal" predicate values because the plan was optimized for a rogue bind value—or because the data values change faster than statistics gathering can keep up with, and you legitimately query for new values that are outside the existing statistics.
b) The change of the default of method_opt for statistics gathering from "for all columns size 1" in 9*i* to "for all columns size auto" in 10*g*, which unpredictably generates histograms for many more columns—too many, in my opinion—than before. Even if, or especially if, the histogram may be justified because the distribution of the column values is highly skewed, one "untypical" bind value can again ruin the access plan for all the other executions of the shared cursor.

No wonder the number of horror stories reported in forums and on Oracle-L has increased since 10*g*, or 11*g*, has widely replaced 9i.

To reiterate my position, I think Oracle tried to fix an application problem, using bind variables where not appropriate, by peeking at bind values (much like it tried to fix another application problem, not using bind variables where appropriate, with cursor_sharing={force|similar}). Of course, the ultimate insanity is to use bind variable peeking together with cursor sharing. ▲

Wolfgang Breitling was born in Stuttgart, Germany, and studied mathematics, physics, and computer science at the University of Stuttgart. Following several years as a systems programmer for IMS and later DB2 databases on IBM mainframes, he got on the project to implement Peoplesoft on Oracle. In 1996 he became an independent consultant specializing in administering and tuning Peoplesoft on Oracle. The particular challenges in tuning Peoplesoft, often with no access to the SQL, motivated him to explore Oracle's cost-based optimizer in an effort to gain a better understanding of how it works and use that knowledge in tuning. He has shared the findings from this research in papers and presentations at IOUG, UKOUG, local Oracle user groups, and other conferences and newsgroups dedicated to Oracle performance topics.

Christian Antognini: Since its introduction in Oracle 9*i*, many people have argued over the pros and cons of bind variable peeking. Not surprisingly, however, most of the contributors to this debate have mostly pointed out the disadvantages of the feature. This is probably because it's much easier to recognize when something goes wrong because of a specific feature than the opposite. So before sharing my answer with you on the question of "bane or boon," I would like to briefly review what the pros and cons of bind variable peeking are. To do so, however, I have to begin with a discussion of the pros and cons of bind variables. This is necessary because too often I hear people criticize bind variable peeking when, in fact, it's the utilization of bind variables that should be questioned.

From a performance point of view, bind variables introduce both an advantage and a disadvantage. The advantage of bind variables is that they allow the sharing of cursors in the library cache and that way avoid hard parses and the associated overhead. The disadvantage of using bind variables in WHERE clauses, and only in WHERE clauses, is that crucial information is hidden from the query optimizer. For the query optimizer, it is in fact much better to have literals instead of bind variables. With literals, it is able to improve its estimations and, therefore, to choose optimal execution plans. This is especially true when it has to check whether a value is outside the range of available values (that is, lower than the minimum value or higher than the maximum value stored in the column), when a predicate in the WHERE clause is based on a range condition (e.g., HIREDATE>'2009-12-31'), and when it makes use of

histograms. As a result, for cursors that can be shared, you should always use bind variables if one of these three conditions is not met. The main exception is for SQL statements for which the parsing time is several orders of magnitude less than the execution time. In this kind of situation, using bind variables is not only irrelevant for the whole execution time, but it also increases the risk that the query optimizer will generate very inefficient execution plans.

To overcome the disadvantage due to bind variables, Oracle introduced bind variable peeking. The concept of bind variable peeking is simple. During the physical optimization phase, the query optimizer peeks at the values of bind variables and uses them as it would with literals. Hence, at first sight we have a win-win situation: the number of hard parses is reduced and the query optimizer makes better estimations. The problem with this approach, however, is that the generated execution plan depends on the values provided by the first execution. Consequently, as long as the cursor remains in the library cache and can be shared, it will be reused. This occurs regardless of the efficiency of the execution plan related to it. As a result, some executions might be efficient and some others might not. In addition, depending on how long a cursor remains in the library cache, unpredictable response times might be experienced. To solve (or at least diminish) the disadvantage due to bind variable peeking, as of Oracle Database 11*g*, a new feature called "extended cursor sharing" (also known as "adaptive cursor sharing") is available. Its purpose is to automatically recognize when the reutilization of an already available cursor leads to inefficient executions. This is a good thing. But be careful, the database engine is only able to recognize such a situation when a given cursor is executed several times with a suboptimal execution plan. In other words, the database engine tries to learn from its mistakes (but, of course, it has to make some mistakes to be able to recognize them).

In summary, to increase the likelihood that the query optimizer will generate efficient execution plans, you should not use bind variables when one of the conditions mentioned before is met. Even if bind variable peeking might help, it is sometimes a matter of luck whether or not an efficient execution plan is generated. The only exception is when the new extended cursor sharing of Oracle Database 11*g* automatically recognizes the problem.

So, bane or boon? As usual, it depends. There are applications that work very well with bind variable peeking. The reason is simple: they have been implemented carefully. They use bind variables in a sensible way. On the other hand, there are applications that experience unpredictable execution times for the opposite reason. For them, it is usually necessary to disable the feature through the undocumented initialization parameter _optim_peek_user_binds. Of course, it is never a good thing to set an undocumented parameter. It is my opinion that Oracle should not only promote this parameter as a regular one but also provide hints to control the utilization of bind variable peeking. In other words, they should give us the ability to choose whether it is beneficial to use bind variable peeking or not. ▲

Since 1995, Christian Antognini has focused on understanding how the Oracle database engine works. His main interests include logical and physical database design, the integration of databases with Java applications, the query optimizer, and basically everything else related to application performance management and optimization. Christian is the author of the book Troubleshooting Oracle Performance *(Apress, 2008). He is currently working as a principal consultant and trainer at Trivadis in Zürich, Switzerland.*

Dan Tow: Prior to 11*g* (where almost all of us are still working), bind-variable peeking was very definitely a two-edged sword, creating at least as many problems as it solved, in my opinion. Consider several possible cases:

1. The query does not execute often, less than once per minute. In this case, even parsing for every execution, with a different hard-coded constant each time it executes, is not that bad, with parse costs lost in the noise compared to overall load on the database. If the constant would almost always only have one of a few values (at most, often just one value), then it is useful to see those few values when tuning and maintaining SQL—hiding the one, or at most few, values behind a bind variable saves extremely few parses and makes the query harder to maintain and tune. In the case where a statement executes rarely and with at most just a few distinct values attached to the bind variable, we are better off with a hard-coded constant than with a bind variable, when possible, eliminating the need for bind variable peeking. Only if the bound values vary over a large number of values is there a maintenance benefit to a bind variable. If you are capturing SQL stats, it is easier to recognize that this is a single statement and a single tuning and maintenance problem if it really *is* a single statement with a bind variable, not a thousand statements with a thousand different hard-coded constants.

2. The query executes often, and the selectivity of the query conditions varies little regardless of the values used in the bind variables. Bind variables are great here, potentially saving lots of parses that would happen if we hard-coded lots of different constants instead, but bind variable peeking is unnecessary because we always end up wanting the same execution plans regardless of the peeked-at values, and using averaged selectivities would yield the same results.

3. The query executes often, and always with one—or at most a few—bind variable values that lead to the same best plan, while that is not the plan we would get if we didn't peek at those values. In this case, bind variable peeking is useful, but the same result can be had with hard-coded constants (with a few more parses, at most, to cover the few values), or with a hint-forced plan that is chosen with the expected selectivity in mind or with plan outlines.

4. The query executes often, and the selectivity of the query conditions varies widely, depending on the values used in the bind variables. This is the case that bind variable peeking was designed to address—we potentially need bind variables to avoid frequent parses, but the best execution plan depends on the values attached to those bind variables, and presumably we have any necessary histograms to help know how selectivity

would vary by bind variable assignments. However, the big problem here is that prior to 11*g*, bind variable peeking only happens at the first parse, and we're stuck with that execution plan until the SQL ages out of the cache (which won't happen for a long time in this case where the query executes often), so, since selectivities vary widely for values actually used, the chosen plan that we are stuck with, and that was correct for the first execution, will be wrong for many of the other executions. Furthermore, we have nasty inconsistency in performance, where one day the first execution leads to an execution plan that works terribly for one set of bind values, and the next day (when that first peek leads to a different plan), we get a plan that works terribly for the other set of bind values. Even if we usually get a plan that works well most of the time, we can unexpectedly get a plan that works really poorly on average, possibly at a very inopportune time, like the quarter close, just because we got unlucky and that first peek saw some uncommon case that leads to a plan that usually runs poorly. This behavior is very hard to anticipate or prevent as long as we have bind variable peeking. There are really two subcases here:

a) The most common subcase, by far, is that there are at most a few values for some unevenly distributed column matched to a bind variable. In this case, it's not a problem to replace the bind variable with hard-coded constants, since the result will be only a few parses for a few versions of the query, and we get different execution plans precisely when we want different plans.
b) A very rare subcase is that we have a large number of possible values—and some call for one plan while others call for another plan—and the query executes frequently enough for parse costs to matter. Here we have some trade-offs: We can live with whatever unpredictable result we get from the peeking for the first parse; we can force a single plan, chosen to be better on average, with hints that override bind-variable peeking; we can hard-code constants, tolerating significantly higher parse costs in order to get the best plans every time; or we can distinguish which plan we would prefer at the application layer and branch to different SQL with forcing hints for each of the cases.

*I can see no cases, prior to 11*g*, where bind variable peeking offers both significant parse savings over hard-coded constants and consistent good performance compared to results we get without bind variable peeking. In practice, I have seen many problems created by bind variable peeking, and the only problems I have seen prevented were problems that could also have been prevented by hard-coding a constant where a bind variable was always assigned the same value or by using dynamic hardcoding of values for SQL that executed very infrequently.* ▲

Dan Tow has 20 years of experience focused on performance and tuning, beginning at Oracle Corporation from 1989 to 1998, where he headed the performance and tuning group for all of Oracle applications and invented a systematic, patented method (U.S. Patent #5761654) to tune any query efficiently. This method is extended and elaborated in his book, SQL Tuning. Dan lives in Palo Alto, California, and is reachable at **dantow@singingsql.com**.

Jonathan Lewis: Looking at this question, I can't help imagining an editor writing an opinion piece shortly after the scythe started replacing the sickle as the latest technological aid for bringing in the harvest. *Scythes: bane or boon?* How many farmers would praise the implement for the extra efficiency, how many would curse the way it made it easy to take a chunk out of your own leg? The story of bind variable peeking is the same.

Developers and DBAs hadn't really got to grips with the cost-based optimizer and the impact of statistics on execution plans when Oracle Corp. dropped a bombshell in 9*i* that meant you couldn't figure out why the current execution plan had appeared for a query (using bind variables) unless you knew what values had been passed in when that plan was first generated. And this wasn't a feature you could choose to enable, it was a feature you could only choose to disable—if you discovered the correct hidden parameter!

If you have lots of users on an OLTP system who are constantly doing the same high-precision jobs, using SQL statements that can be separated into a small number of "class actions" where every statement in a given class is virtually identical and is expected to do the same amount of work as every statement in that class, then one "token" optimization for each class is a good idea because optimization is expensive and you don't want to keep doing it for every tiny little query. And that's why (a) bind variables are good—each class of statements turns into a single representative text with bind variables, and (b) bind variable peeking is good—you really do need to use genuine values when you do your one "token" optimization.

Note, however, the critical assumptions:

- Everyone is doing the same amount of work with any given (class of) SQL statement.
- Every (class of) SQL statement is used very frequently.
- Each (class of) SQL statement does a small amount of work—so the overhead of optimization is relatively large compared to the work done.

As soon as your application fails to meet this model, you have to ask yourself: Do you really need to adopt a bind variable approach? Could it even be counter-productive to adopt a bind variable approach? And if you do adopt a bind variable approach, how flexible and subtle does it have to be to avoid the side effects of not conforming to the model?

The biggest threat with bind variables comes when the first assumption doesn't apply. If different input values result in different amounts of work, a single execution plan from the "token" optimization may not be enough —but one plan is all you get and that one plan is dependent on the first set of peeked values. There are three main types of query most likely to cause problems. Queries involving:

- Ranges (often date ranges) that are allowed to vary enormously
- Columns such as "status" or "flag" that have a small number of distinct values with very different frequencies that have had (frequency) histograms built on them

(continued on page 17)

The Manga Guide To Databases

A Book Review by Brian Hitchcock

Details

Author: Mana Takahashi and Shoko Azuma

ISBN: 978-1-59327-190-9

Pages: 224

Year of Publication: 2009

Edition: 1

Price: $19.95

Publisher: No Starch Press

Summary

Overall review: A refreshingly entertaining overview of a topic that is all too often made boring by experts.

Target audience: Anyone new to databases.

Would you recommend to others: Yes.

Who will get the most from this book: Anyone who needs to understand the basic principles of relational databases.

Is this book platform specific: No.

Why did I obtain this book: While I was attending MySQL Bootcamp, one of the instructors brought this book to class.

Overall Review

First, what is Manga? I'd say it is a form of graphic novel or comic book from Japan. You may not be familiar with Manga, but you would probably recognize the style of illustration seen in Manga. It has become very popular recently, and now we have a whole series that sets out to teach subjects such as electricity, physics, and databases, and that is illustrated in this style. While it would be easy to dismiss the books in this series as being just for kids, that isn't fair. I got a chance to look at this book while attending a MySQL class and found it interesting. And after reading it, I found it worthwhile.

The material is presented cheerfully, at times bordering on the comical, but the information is accurate and easily understood. For people who aren't already familiar with how relational databases operate, this book is very good. I would recommend it to anyone who needs to understand, at a high level, the most important features of all modern relational database products.

The material is really very serious, but it is presented by means of a simple story, a very Manga-type story of a kingdom that sells fruit and all the problems that come up as their business grows and they try to keep track of everything with paper and pencil. You will not be surprised to find out that just when our heroes appear to have no hope, a database comes to the rescue.

Preface

Here the author presents the design and structure of the book. We are told that databases are crucial parts of almost all business computer systems and that the true nature of the database is hard to understand. I'm not sure I know the true nature of the database, but it is an interesting concept (something to think about while waiting for support to respond).

Chapter 1: What Is a Database?

This chapter actually starts with a more important question, namely, why do we need a database? To answer this question, a story is told. It's the story of a kingdom that sells fruit —lots of fruit, and more and more fruit all the time. A princess (Ruruna) is responsible for all aspects of the fruit business and she relies on a humble assistant (Cain). (Do you have any users who act like a prince or princess? I thought so . . .) The princess is struggling to keep up with all the paperwork for the business using her laptop—a laptop with a logo that looks like a piece of fruit, but definitely not an apple.

It is the busy season for the fruit business and the princess is frustrated. There are so many departments, and they sell fruit to many countries. Things are getting out of hand.

Enter the humble assistant with a magic book from the king, a book that describes a secret technology called a "database." Apparently the king found this book in a faraway land. (Someone went to Oracle World on a full pass and attended a lot of marketing presentations.)

Our heroes (the princess and her humble assistant) open the book and release a fairy, a database fairy named Tico. And the database fairy is only visible to those who opened the book. (DBAs appear to be talking to persons who aren't really there all the time . . .)

Tico offers to help our heroes with the supernatural powers of the book in a proper manner. Tico says the first step is to create a database, at which point the princess asks, what is a database? The database fairy is eager to explain, and then observes that our heroes are trying to manage values and numbers, customers and sales, files and departments, and all in an uncoordinated fashion. Data is duplicated in each department, for example. This can create problems. It seems the princess had a crisis just the other day. (This is sounding more and more familiar.) Specific examples of the daily crises are given: the price of apples went up and this required updating data in multiple places. One department didn't update their data, one department updated the price incorrectly, and there was lots of time spent finding and correcting all these errors. Tico offers insight: "You will be tormented by data management even if you do your best, won't you." This hits close to home.

Tormented. The database fairy speaks the truth.

But Tico is here to help, explaining that a database is a system where data can be shared by everyone (cue the SOX fairy . . .) and, listen carefully, if you use a database you "would not have to keep useless data." Unfortunately, Tico doesn't explain that last part. Upon hearing this, the princess wants a database very much and now wants to create one and asks Tico, "Can't you just do it?" Sounds like a consulting gig to me, but Tico replies that the princess and her assistant have to create the database themselves. Tico can only offer guidance. Sounds like a consultant to me.

At this point the Manga story pauses and we have several pages where the ideas introduced so far are described in a more classical style without comic book artwork. This section covers how data is duplicated among departments, how data conflicts arise, update difficulties, and how a database handles all of these issues.

Chapter 2: What Is a Relational Database?

This chapter starts, after a brief bit of drama with a prince from a neighboring country (who has no interest in databases, only the princess), by defining some database terms. Tico, the database fairy, explains that this knowledge is necessary before designing a database. Terms such as "record," "field," and "product code" are discussed. You can see how this is leading to normalization. We also see that duplicated records are not good. The need for NULLs is brought up and covered very briefly—if only it were this simple.

Next we are told about different data models used in databases. Tico, with a BOINK and a SHAZAM (no, I'm not making this up!) causes a hierarchical data model to appear, followed, with similar sound effects, by a network data model and finally a relational model. The first two models are left simply as diagrams with little explanation, while the relational model is explained in detail. Tables with keys and relational operations are all covered. And then it is time to leave the story to give more detailed explanations and examples of all that was covered so far. The types of data models are further explained. The way the various operations work is illustrated with tables and data for the kingdom's fruit business. At the end of this section we are told that "The Relational Database Prevails!"—and indeed it does!

Chapter 3: Let's Design a Database!

Now that the princess Ruruna and her assistant Cain have the necessary knowledge, it is time to design the database for the kingdom. Tico appears to present an ER diagram. Entities and relationships are discussed. A many-to-many (way too many?) relationship is illustrated between fruit and export destinations. I like the map showing a horse-drawn cart overloaded with fruit exiting the kingdom. My databases are never this interesting. Tico explains all this by using a chalkboard, a nice retro touch.

Next up is normalization, which requires making more tables. The princess is not happy; she had just one table for all the product sales and was satisfied. But Tico shows us why more tables are better with the example of updating the price of a product. Better to have the price of each product in a separate table and link that to a table of sales.

Tico is shown leading our heroes up "Mt. Relational Database" as the levels of normalization are explained. Everyone is very tired when this is completed. Having arrived at the top of the mountain, with third normal form, it is again time for a more detailed explanation of the various concepts covered. More traditional examples are given to further illustrate relationships and normalization.

Chapter 4: Let's Learn About SQL!

This chapter opens with more Manga-style drama. The princess and her assistant are strolling through the town. The relationship between them isn't clear, but the princess obviously perceives that they have some sort of connection. The prince appears and suddenly the princess wants to go into a café with Cain (and not the prince) to "study databases for a while." I guess this is one way to deal with your relationship issues. And Cain is clueless. But so it goes. Anyway, with this drama out of the way we can now focus on SQL. Why the prince isn't bright enough to pursue the princess into the café is not explained.

Tico re-emerges from the database book and explains that we need SQL to use the database. Both the princess and Cain look at Tico and in unison ask "Squeal?" Indeed, there is lots of squealing when users start using SQL. Tico explains that when you go to the Swimmy region "across the sea" you need to speak the Swimmy language, and therefore, when you want to have a conversation with a database you need SQL. Ah, sure, the Swimmy Region and Squeal. It's all coming into focus, isn't it?

But seriously, we learn how to use SQL to enter data into and retrieve data from the tables we created earlier to model the entities and relationships for the kingdom's fruit business. When Tico directs our heroes to retrieve data from the product table, both Princess Ruruna and Cain begin to pray for the data to be retrieved. Squealing and praying— clearly Tico has worked with users for a long time! After explaining how to get data from a single table using wildcards and how to sort the data that is returned, joins are discussed. Tico also tells us that a table must have a primary key and that this will prevent the entry of incorrect data. I would offer that incorrect data can be entered at any time, the definition of what would make data incorrect is complicated, and that a primary key really only prevents entering duplicate key values, but Tico seems pretty definite about this so I'll keep my opinions to myself. Again it is time for a more detailed explanation of SQL, comparison operators, searches, functions, subqueries, and how to create tables. All the basics are covered. With a description of views and several examples for us to study, we are done.

Chapter 5: Let's Operate a Database!

Suddenly, the quiet kingdom has an IT department! We see nameless drones pounding at keyboards (click, click, clack, click) surrounded by lots of screens. While our heroes have learned a lot about databases, they have "some worries." Cain has done some research and wonders how a database can let a large number of users access data at the same time. But not to worry, he has a set of charts that he will present to us, with a wonderful title of "Database Theater." My Oracle classes were

never this entertaining. There is much excitement (clapping and a "Whoopee!") as the show begins. The first chart is presented: "One day, Andy and Becky accessed the database at the same time"—the chart shows two users (stick figures), each with a pointed hat, one labeled "A" and the other "B." Do your users wear silly hats? Do you see them as cartoonish stick figures?

It turns out that UserA and UserB both read from the database that a total of 30 apples were in stock, but UserA added 10 to that amount while UserB did the same thing. We can see that 50 apples total are now in stock, but the database only reports 40. The princess is quick to accuse Cain of eating the 10 missing apples (Cain is a DBA, he just doesn't realize it yet).

Cain has more charts of course, and he is quick to offer that a database is designed to process data operations in "lumps." Indeed, this is the way I've always thought of it. Lumps, swirling silently in space. And these "lumpy" operations are called "transactions." I will assume that a follow-up book will be needed to explain what happens when your database has too many lumps at one time. A lumpy Method-R perhaps? The Automated Lump Repository can't be far behind, but it will probably require an additional license fee. Cain's charts provide a very clear and detailed explanation of how transactions prevent UserA and UserB from having a conflict, and locks come into the picture. The princess is impressed with Cain. There are different kinds of locks. Tico whispers in the princess's ear that Cain is "dependable." Deadlocks are covered in more charts, and a cancelled transaction is rolled back while completed transactions are committed. But wait, Database Theater is about to provide more relationship drama as the prince appears with a relevant comment about concurrency. Who knew? It seems the prince has appeared because he has invoices from the kingdom that are a real mess, and also because "a database is a nasty thing." Cain (ever more the hapless DBA) opines that "someone with malicious intent might have performed an unauthorized data overwrite." What? There's no chance that a user with a silly hat put in some incorrect data? My money is on UserZ—I never trusted that user. But back to the relationship drama. The prince thinks that as compensation for this inconvenience, the princess should marry him and leave the kingdom to live in his country. Cain isn't sure why he is upset by this (clueless) but he keeps quiet. Cain proposes that they set up usernames and passwords to control user access. And they will grant specific rights to access specific data to specific users. The princess thinks this is clever. The prince tries to steer the princess back to the subject of marriage, but Cain isn't done. He also wonders what will happen as more and more users access the growing database. Searches could become slow. The prince asks if it is "safe to trust a database," foreshadowing future conflicts between his country and the kingdom over things like medical records privacy, et al, but that is getting ahead of the current drama. But Cain (the once and future DBA) is on a roll. He offers that indexing seems to be promising. The prince wants to discuss marriage, but Cain rolls on, explaining all about indexes. The prince and Cain then discuss disaster recovery and logs that keep track of all data operations performed in the database. It isn't clear why the prince is suddenly able to discuss database issues. Perhaps his country needs a DBA as well? We are told about rolling forward (and back) transactions as part of disaster recovery. And now, the prince returns to the marriage issue, but the princess suddenly announces her "passion" for Cain and their database.

This is followed by detailed descriptions of transactions, atomicity, concurrency, and query optimization.

Chapter 6: Databases Are Everywhere!

The Manga drama is pretty much over. Now that the princess has a DBA under her control (marriage is a low-cost way to ensure in-house DBA services) she can focus on running her empire. Tico tells our heroes that databases are used in other countries for things like banks, railways and even the Web! It seems that, in other countries, you can by any book you want from the Web! Imagine that! It seems that you can enter search criteria into a web page and that information is used by a database on the Web to find and return data using SQL. Princess Ruruna is shocked to find that her father has a book for sale online titled *Fruit Love*. But we move on. This leads to a discussion of multiple databases to share the load (distributed database, failover) and hints at the issues of keeping multiple databases in sync so that any one of them can take over if the others fail. Stored procedures are described as a way to reduce the burden on the network by doing lots of the processing in the server instead of sending SQL and results back and forth. Triggers are next. Then princess Ruruna concludes that she will, by using databases, build a wonderful country. Wow! These databases can work miracles!

The final section provides more details about databases on the Web, stored procedures, partitioning data, two-phase commit, replication, and more.

Conclusion

Despite what you might think, the material is very accurate and does bring out the important issues. The style is very different from all other database books I've seen, and that is very refreshing.

Perhaps I will be replaced by the database fairy—oh wait, that's right, the fairy can't do the actual work. Perhaps this fantasy kingdom looks a lot more like my workplace than at first glance?

I think this book has another message, one that is very important for all of us in the database management world. This series of books (search Amazon.com for *Manga Guide*) covers very basic topics such as electricity, physics, statistics, and molecular biology (and Sudoku!). While Manga is enjoyed by many people of all ages, it is primarily aimed at young people. Note the range of subjects that young people are interested in these days, and then realize that databases are right in there with molecular biology (and Sudoku!). What we do all day has become relevant to a much wider audience than we realize. Many high school students have had exposure to MySQL databases as part of websites they frequent. These days, databases really are a part of everyday life and are becoming more important all the time. The fact that a publisher wants a series of Manga titles to cover databases is something worth knowing. The impact of the systems we create and support is huge and growing.

(continued on page 17)

First International NoCOUG SQL Challenge

We Have a Winner!

A News Report by Iggy Fernandez

The *First International NoCOUG SQL Challenge* was a great success; nine solutions were found by participants in seven countries and three continents. Alberto Dell'Era wins the contest for his wonderful solution using Discrete Fourier Transforms; the runner-up is André Araujo from Australia, who used binary arithmetic and common table expressions in his solution. The *August Order of the Wooden Pretzel*[1] will be bestowed on Alberto but the real prize is six books of his choice from the Apress catalog. André will receive a prize of six e-books of his choice. Thanks to Chen Shapira for publicizing the event in her blog, Dan Tow for helping to judge the contest, and Apress for donating the books.

The Challenge

An ancient 20-sided die (icosahedron) was discovered in the secret chamber of mystery at Hogwash School of Es-Cue-El. A mysterious symbol was inscribed on each face of the die. The great Wizard of Odds discovered that each symbol represents a number. The great wizard discovered that the die was biased: that is, it was more probable that certain numbers would be displayed than others if the die were used in a game of chance. The great wizard recorded this information in tabular fashion as described below.

Name	Null?	Type
FACE_ID	NOT NULL	INT
FACE_VALUE	NOT NULL	INT
PROBABILITY	NOT NULL	REAL

The great wizard then invited all practitioners of the ancient arts of Es-Cue-El to create an Es-Cue-El spell that displays the probabilities of obtaining various sums when the die is thrown N times in succession in a game of chance, N being a *substitution variable* or *bind variable*. The rules of the competition can be found in the May 2009 issue of the *NoCOUG Journal* and on the Web at **www.nocoug.org/SQLchallenge/FirstSQLchallenge.pdf**.

[1] Steven Feuerstein was asked the following question in an interview published in the May 2006 issue of the *NoCOUG Journal*: *"SQL is a set-oriented non-procedural language; i.e., it works on sets and does not specify access paths. PL/SQL on the other hand is a record-oriented procedural language, as is very clear from the name. What is the place of a record-oriented procedural language in the relational world?"* Steven replied: *"Its place is proven: SQL is not a complete language. Some people can perform seeming miracles with straight SQL, but the statements can end up looking like pretzels created by someone who is experimenting with hallucinogens. We need more than SQL to build our applications, whether it is the implementation of business rules or application logic. PL/SQL remains the fastest and easiest way to access and manipulate data in an Oracle RDBMS, and I am certain it is going to stay that way for decades."*

The Solutions

The modus operandi was for each participant to post their solutions on their blog or website; you can find them all with a simple Google search. The first solution—by **Laurent Schneider** from Switzerland—used the *CONNECT BY* clause to join the table to itself N times and the *SYS_CONNECT_BY_PATH* and *XMLQUERY* functions to perform the necessary additions and multiplications. The number of records generated by CONNECT BY grows exponentially and hurts performance.

The second solution—by **Craig Martin** from the USA—used the CONNECT BY clause to join the table to itself N times and logarithms to perform the necessary additions and multiplications. The number of records grows exponentially in this case too.

The third solution—by **Rob van Wijk** from the Netherlands—used the *Model* clause to generate records. The number of records grows exponentially in this case too.

The fourth solution—by **Vadim Tropashko** from the USA—used *recursive common table expressions* to generate records. The number of records grows exponentially in this case too. Recursive common table expressions are available in Microsoft SQL Server and DB2 but are not presently available in Oracle 11*g*R1. Rumor has that they will be available in Oracle Database 11*g*R2.

The fifth and sixth solutions—by **Alberto Dell'Era** from Italy—used advanced mathematical techniques such as *convolutions*, *Discrete Fourier Transforms*, and *Fast Fourier Transforms*. The Fast Fourier Transform method is an efficient way of calculating Discrete Fourier Transforms and was implemented using the Model clause.

The seventh solution—by **Fabien Contaminard** from France—was based on the *multinomial* probability distribution, an extension of the binomial distribution.

The eighth solution—by a blogger named Cd-MaN from Romania—used *pipelined table functions* and recursion. The solution was demonstrated in a Postgres database but can easily be adapted for use in an Oracle Database. The use of recur-

sion means that this is not a pure SQL solution.

The ninth solution—by **André Araujo** from Australia—used binary arithmetic and common table expressions.

Judges' Decision

Dan Tow authored the following statement on behalf of the judging committee:

To begin, we'd like to congratulate the contestants on finding so many different and clever ways to solve this problem, and on making the problem of picking a winner so difficult for us, personally. Each of the solutions had major advantages to recommend it. We were also very pleasantly surprised at the global extent of the entries, with entries from seven nations and three continents!

The criteria for the judging were stated in advance at **www.nocoug.org/SQLchallenge/FirstSQLchallenge.pdf**. The main criteria that separated the top scores from the rest, given that they were all quite good as technical solutions from one perspective or another, were the inclusion of commentary and test results, which were minimal or altogether lacking from most entries. (Hey, we understand that you're busy, so we're not surprised to see these missing or minimal, but they were important to getting a win here!)

There were elegant solutions using the Model clause, including one by Alberto Dell'Era that implemented a solution using Fast Fourier Transforms that was technically amazing, well-documented and tested, and scaled better than any other solution, with order N * Log(N) scaling, almost linear up to enormous numbers of throws of the dice. However, these used "iterate" loops that we believe violated the contest requirement that "Solutions that use procedural loops to multiply probabilities are not eligible" stated at the top of the "Judges' Statement." (We know that there are wonderful, super-efficient ways to solve this problem in procedural code like C, but the point behind that unbendable requirement was to get contestants "thinking in SQL," doing the job in a set-wise manner, not just finding ways to bend SQL into doing what we'd be better off doing in C, procedurally.) The "procedural loop" component in these solutions was really minimal and easy to miss, even, in a casual examination of the code, but we think we have to stick with the pre-stated rules here and disqualify those solutions, even while we admire them.

Scaling almost as well (at order N * N), and also very well documented and tested, both from the perspective of performance and functionality, was the amazing Fourier-Transform-based solution, **www.adellera.it/investigations/nocoug_challenge/index.html** also by Alberto Dell'Era from Italy, that we think has to be declared the winner here, with no procedural loops and good-to-excellent scores in every stated judging criteria.

The runner-up choice is difficult, too, but we'd probably have to go with André Araujo of Australia, **www.pythian.com/news/2385/nocoug-sql-challenge-entry**. Of all the solutions, his probably best combined straightforward (very clever, but still straightforward to the reader!), portable SQL that could be easily understood and maintained by a developer without an advanced degree in mathematics, with fairly scalable and well-tested SQL that ran well up to quite high numbers of throws of the die ("N"). It is true that the SQL had a hard-coded limit of N = 511 (a limit that André documented well, to the credit of the solution), and that functional limit lost a few points, but we should keep in mind that this high value of N is one that most of the implementations (other than Alberto Dell'Era's) would never reach in our lifetime, anyway, owing to their comparative lack of scalability—being logically correct at high N is worth nothing if the program never finishes! If we had to actually maintain one of these in a production environment, and we didn't anticipate needing results at very high values of N, we'd probably go with André's solution, just because we'd be frightened of long-term maintenance on the high mathematics of Alberto Dell'Era's brilliant but more complex and technically harder-to-follow solution.

Analysis of the Winning Solution

Alberto recognized that the contents of the die table define a mathematical function and that the process of joining the table with itself and grouping the results is the so-called *convolution* of this function with itself. Throwing the die N times is therefore equivalent to performing N – 1 convolutions. For example, for N = 3, we have to perform two convolutions. This is best expressed using *common table expressions* as follows. Notice that the definition of the second convolution references the first convolution.

```
WITH

first_convolution AS
(
  SELECT face_value, SUM (probability) AS probability
  FROM
  (
    SELECT
      d1.face_value + d2.face_value AS face_value,
      d1.probability * d2.probability AS probability
    FROM die d1 CROSS JOIN die d2
  )
  GROUP BY face_value
),

second_convolution AS
(
  SELECT face_value, SUM (probability) AS probability
  FROM
  (
    SELECT
      d1.face_value + d2.face_value AS face_value,
      d1.probability * d2.probability AS probability
    FROM first_convolution d1 CROSS JOIN die d2
  )
  GROUP BY face_value
)

SELECT face_value, probability
FROM second_convolution
ORDER BY face_value;
```

The most common solution of the problem requires an N-way cross join. Convolutions have obvious advantages over an N-way cross join because they keep the size of intermediate results in check. The question is how to compute the required N – 1 convolutions with a single SQL statement if the value of N is not known in advance. One solution is to recursively invoke a *table function* as was done by the Romanian contestant; that is, we have to resort to procedural programming. Alberto was able to avoid procedural programming using a *Fourier transform;* that is, a certain function whose definition is derived from the original function. The Fourier transform has the interesting property that the transform of the convolution

of functions is the simple product of the individual transforms. Therefore, the Fourier transform of the N-way convolution of our function with itself is the Nth power of the Fourier transform of the function. The Fourier transform is straightforward to compute and—with a little mathematical trick involving a conversion from the Cartesian coordinate system to the polar coordinate system—the Nth power of the Fourier transform is also straightforward to compute. At this point, Alberto has the Fourier transform of the N-way convolution. To obtain the result that he really needs, all that is left for him to do is to compute the *Inverse Fourier Transform* of the Fourier transform. An explanation of Fourier Transforms can be found on the Web; efficient C-language implementations can also be readily found.

For readability and maintainability, Alberto uses a sequence of *common table expressions*. The following is a simplified version of his solution; the full solution handles more general cases and uses some tricks to reduce the number of scientific computations. Hints to guide the optimizer and improve efficiency are included in the version shown below.

First Alberto creates a one-column table of sequence numbers using the CONNECT BY method; this table is used several times in the rest of the solution. The number of elements in the table is one more than the product of the number of sides of the die and the number of throws.

```
sequence AS
(
  SELECT /*+ NO_MERGE */
    LEVEL - 1 AS n
  FROM dual
  CONNECT BY LEVEL <= (:N * :sides + 1)
)
```

Alberto then constructs a *discrete function* whose domain is the numbers in the sequence table. The function is called *discrete* because it is only defined for certain discrete values—not for all values in a range as in the case of a *continuous* function. Whenever possible, the function uses the values in the die table. If a value is not found, the value of the function is set to zero. This is done using an *outer join*.

```
function AS
(
  SELECT /*+ NO_MERGE */
    n,
    COALESCE(probability, 0) AS x
  FROM sequence LEFT OUTER JOIN die ON (n = face_value)
)
```

Alberto then computes the *Fourier transform* of the discrete function. There's some advanced math going on here but it's easy to take in small doses; you'll recognize Pi as the well-known mathematical constant. A *cross join* with the Sequence table is required to calculate sums.

```
transform AS
(
  SELECT /*+ NO_MERGE LEADING(function) */
    sequence.n,
    SUM(x * COS(-2 * :Pi * sequence.n * function.n / (:N * :sides + 1))) AS x,
    SUM(x * SIN(-2 * :Pi * sequence.n * function.n / (:N * :sides + 1))) AS y
  FROM function CROSS JOIN sequence
  GROUP BY sequence.n
)
```

In another little dose of math, Alberto switches to *polar form* for ease of further computation.

```
polar AS
(
  SELECT /*+ NO_MERGE */
    n,
    SQRT((x * x) + (y * y)) AS r,
    CASE
      WHEN ABS(y) < 0.000001 AND ABS(x) < 0.000001 THEN 0
      ELSE ATAN2(y, x)
    END AS theta
  FROM transform
)
```

Computing the Nth power of the Fourier transform is then very easy.

```
power AS
(
  SELECT /*+ NO_MERGE */
    n,
    POWER(r, :N) AS r,
    theta * :N AS theta
  FROM polar
)
```

Alberto has no more use for the polar form, so he converts back to Cartesian form.

```
cartesian AS
(
  SELECT /*+ NO_MERGE */
    n,
    r * COS(theta) AS x,
    r * SIN(theta) AS y
  FROM power
)
```

So far, Alberto has calculated the Fourier transform of the discrete function and computed its Nth power. As explained earlier, this is the Fourier transform of the N-way convolution of the discrete function. To obtain the convolution itself, Alberto computes the *Inverse Fourier Transform* using another *cross join* with the Sequence table.

```
convolution AS
(
  SELECT /*+ NO_MERGE LEADING(cartesian) */
    sequence.n,
    SUM
    (
      x * COS(+2 * :Pi * cartesian.n * sequence.n / (:N * :sides + 1)) -
      y * SIN(+2 * :Pi * cartesian.n * sequence.n / (:N * :sides + 1))
    ) / (:N * :sides + 1) AS x
  FROM cartesian CROSS JOIN sequence
  GROUP BY sequence.n
)
```

Finally, Alberto can display the results. The result has more than 30 digits of decimal precision; only 30 are displayed in the interests of accuracy.

```
SELECT
  n AS face_value,
  ROUND(x, 30) AS probability
FROM convolution
WHERE n >= :N
ORDER BY n;
```

As you can see, Alberto's solution used advanced mathematical techniques, but it is not very long and the use of *common table expressions* makes it quite readable. We have a winner! ▲

(continued from page 10)

- Partitioning keys—especially with list partitions of different sizes

If you have critical queries that match these characteristics, you may need to find a way to get multiple plans for the same SQL text—and that typically involves being selective in your use of bind variables or adding complexity to the code that drives the queries.

So bind peeking is generally a boon for OLTP systems where everyone does the same thing very frequently. But, almost invariably, bind peeking will cause problems in a few cases because queries that superficially look the same may need to collect dramatically different amounts of data using different execution paths. ▲

Jonathan Lewis is well known to the Oracle community as a consultant, author, and speaker, with more than 20 years of experience in designing, optimizing, and troubleshooting on Oracle database systems. His latest book is Cost Based Oracle Fundamentals, which is the first of three volumes on understanding and using the cost-based optimizer.

(continued from page 13)

Finally, if you should want to read some Manga, I recommend a series titled DeathNote. Full Metal Alchemist is also very good. Another Manga series called Ghost in the Shell is great, but is only available in Japanese. This series is available in the Anime (animation) form in English. You might want to read these, if only so that you can be slightly less un-hip than you currently are. ▲

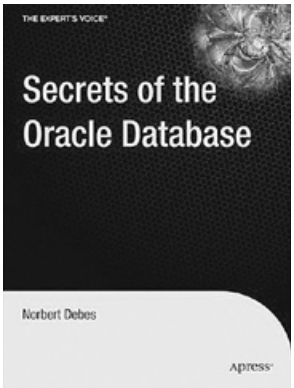
Brian Hitchcock has worked at Sun Microsystems in Newark, California, for the past 11 years. He is a member of a DBA team that supports 2400+ databases for many different applications at Sun. He frequently handles issues involving tuning, character sets, and Oracle applications. Other interests include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

(continued from page 26)

Developer supports unit testing and an integrated data modeler, and we'll review these new features. The session closes with a brief review of new functionality planned for the next release of SQL Developer.

Anatomy of a Database Attack
Dana Tamir, Imperva 4:00–5:00

Corporate databases are under siege. From outside the organization, criminals can exploit web applications to steal confidential information for financial gain. From the inside, databases can be compromised by employees and contractors with malicious intent. SQL Injection, platform vulnerabilities, buffer overflows: databases are vulnerable to a myriad of threats and attack vectors. This presentation will use live demonstrations to trace the steps involved in breaking into a database and present a reference architecture and checklist for implementing iron-clad database security measures. ▲



More Oracle Secrets

by Norbert Debes

Norbert Debes

My name is Norbert Debes and I'm the author of Secrets of the Oracle Database*, an Apress book that unveils undocumented aspects of the Oracle database server. Starting with this edition I'm going to contribute regularly to the* NoCOUG Journal. *My contributions to the* Journal *will have the same theme as my book—useful, undocumented features that Oracle Corp. does not tell us about but nonetheless help us DBAs do our jobs better and more efficiently or, better yet, solve problems that would have been impossible to solve with documented features.*

Oracle Corp. has always emphasized the fact that their database server is very scalable. The reuse of cursors in the shared pool that is enabled by using bind variables in recurring statements is an important facet in attaining scalability. Frequently executed SQL statements without bind variables have caused headaches to numerous DBAs and resulted in far less than optimal performance of quite a few applications. In this article, I'm going to talk about assessing the performance impact of such statements based on an undocumented feature that was introduced with Oracle10*g*.

Terminology

According to the *Oracle Concepts* manual, "a *cursor* is a handle or name for a private SQL area—an area in memory in which a parsed statement and other information for processing the statement are kept."[1] Elsewhere the manual states: "Oracle represents each SQL statement it runs with a shared SQL area and a private SQL area." *Bind variables*, i.e., placeholders in SQL statements, are key to reusing the information in the shared SQL area. Depending on the programming language, bind variables may appear as "*:n*" (n≥0) or ":<id>", where id as an identifier, in the SQL statement text. The former format is used in binding by position, the latter in binding by name.[2]

A *literal* is a numeric or character constant in a SQL statement. Literals defeat the reuse of cursors. In the following code section, the first `SELECT` statement is reusable since it contains a bind variable; the second is not, since it contains a literal in the `WHERE` clause.

```
variable department_id number
exec :department_id:=200

SELECT department_name FROM departments
WHERE department_id=:department_id;

-- not reusable for departments other than 200
SELECT department_name FROM departments
WHERE department_id=200;
```

I'm going to call a cursor associated with a statement that fails to use bind variables a *non-reusable cursor*. Likewise the text of the underlying SQL statement is called non-reusable. DDL statements are non-reusable by their very nature, but `INSERT`, `SELECT`, `UPDATE`, `DELETE`, and `MERGE` are all reusable with bind variables.

Cursors and the Shared Pool

Let's quickly review the infrastructure for reusing cursors offered by the Oracle DBMS. Any SQL statement executed by a database client is represented by a cursor in the shared pool of the database engine. Essentially a cursor is opened, the statement text associated with a cursor is parsed, values are supplied for bind variables (optional), and the cursor is executed. If the statement in question is a `SELECT` statement, fetch may be called on the cursor to retrieve rows. When the application is done, it should close the cursor to release memory and avoid exceeding the maximum number of open cursors (initialization parameter `OPEN_CURSORS`).

Reusing cursors helps scalabilty since it avoids the overhead of repeatedly checking statements for correct syntax and for access privileges to the referenced database objects. It also makes execution plans reusable; hence the DBMS does not repeatedly have to spend CPU cycles on trying to find the best execution plan with cost-based optimization.

V$ Views

The cursors opened by a session are in `V$OPEN_CURSOR`. Each cursor has a certain amount of memory that is private to the session using the cursor. Another memory region associated with a cursor is reusable by other sessions. Among other things, the reusable part comprises the execution plan (`V$SQL_PLAN`). The size of the reusable memory region is available by querying the column SHARABLE_MEM of the fixed view `V$SQL`.

[1] Oracle Database Concepts 10*g* Release 2 (10.2) B14220-02, page 24-4.

[2] Binding by name is a new feature in JDBC 3.0 (Oracle10*g* and newer drivers).

By default, only statements with identical SQL statement texts are reusable. The DBMS calculates a hash value on the statement text (V$SQL.HASH_VALUE) and uses it to search for an already parsed cursor. If a match is found, information on the cursor is reused and the statement executes faster. Since Oracle10*g*, a new approach to uniquely identifying a SQL statement was added—the SQL ID (V$SQL.SQL_ID). The SQL ID is relied upon by the active workload repository (AWR). The old hash value (V$SQL.OLD_HASH_VALUE) that retains the algorithm used in Oracle9*i* is available too. The old hash value is still used by STATSPACK.

Child Cursors

Matters are more intricate than I have described so far. There may be multiple child cursors (V$SQL_PLAN.CHILD_NUMBER) per cursor. Each time a statement is parsed with a different optimizer environment (i.e., different CBO parameter settings that result in a different value for V$SQL.OPTIMIZER_ENV_HASH_VALUE), a new child cursor is created. The new child cursor may be associated with a different execution plan (V$SQL.PLAN_HASH_VALUE).

When a database client performs a parse call on a cursor, the DBMS calculates a hash value on the statement text and tries to locate existing information based on previously executed statements with the same hash value. If the optimizer environment matches an existing one, the plan can be reused in addition to the other information.

CURSOR_SHARING

Oracle development recognized the need to make some of the benefits of reusable cursors available to applications that do not use bind variables. Hence they introduced the parameter CURSOR_SHARING with Oracle8*i*. The default value of this parameter is EXACT, meaning that only cursors for statements with identical text can be reused. The other two allowed values are SIMILAR and FORCE. SIMILAR is the recommended setting to alleviate the damage done by statements without bind variables since—according to documentation—literals will not be replaced by bind variables if "the degree to which the plan is optimized" is affected.[3] What happens when CURSOR_SHARING has a non-default value is that the DBMS rewrites statements with literals to statements with system-generated bind variables. If you see a SQL statement with a bind variable named SYS_B*n* (n≥0) you'll know that the statement was rewritten.

Cursor sharing may be used to improve the performance of non-sharable SQL statements. However, even with cursor sharing, an application will not perform as well as with proper use of bind variables. The highest scalability is attained by enabling server-side (SESSION_CACHED_CURSORS[4]) and client-side (API dependent) cursor caching along with bind variables. This allows statements to be parsed only once as long as the cache is large enough.

FORCE_MATCHING_SIGNATURE

Now that we have laid a foundation for cursor handling and reuse and have a common terminology, it's time to address the core subject of this article—aggregating the resource consumption of non-reusable cursors. In other words, measuring the resources used by SQL statements that would be reusable if the statements contained bind variables.

The undocumented tidbit is that the DBMS engine calculates yet another hash value on a SQL statement's text that is independent of literals. I imagine that this is done by using the tokens recognized by the lexer as part of the syntactic analysis of a statement. The tokens are then fed to a hash function, I presume. Hence, a literal such as "Operations" would be replaced by a token, say STRING_LITERAL, whereas a numeric constant such as 3000 would be matched by the token INTEGER_LITERAL. Of course any other such literals would be represented by the same lexer tokens. A *lexer* is software that converts a sequence of characters into a sequence of tokens.[5] Recognizing the reserved words (keywords) of the SQL language is also accomplished by a lexer.

FORCE_MATCHING_SIGNATURE is a new column in V$SQL and V$SQLAREA that was added in Oracle10*g* Release 2. This new column has data type NUMBER and is also present in DBA_HIST_SQLSTAT and DBA_HIST_ACTIVE_SESS_HISTORY, allowing customers that have a license for the Diagnostics Pack to analyze statements that have been captured by AWR. However, non-reusable statements are unlikely to appear in any AWR or STATSPACK report unless they have caused a significant load during the snapshot interval. A typical case consists of thousands of SELECT, INSERT, or UPDATE statements that execute quickly and never appear in any reports because each statement is handled as separate from the other semantically identical statements.

What the Documentation Says

According to the Oracle10*g* Release 2 and Oracle11g Release 1 database reference manuals, the force matching signature is the "signature used when the CURSOR_SHARING parameter is set to FORCE." Not very enlightening. Unfortunately this is the only bit of information available.

What about CURSOR_SHARING=SIMILAR? And EXACT? Does this mean the force matching signature is available only when the parameter CURSOR_SHARING=FORCE? What statement types have a force matching signature? Are there any useful applications of the force matching signature for us DBAs? Here we have another instance of mythmaking caused by poor documentation from Oracle Corp.

The Facts

Testing reveals that the force matching signature is calculated irrespective of the parameter CURSOR_SHARING. The value zero indicates that no force matching signature has been calculated. I assume that the DBMS uses the signature to match statements that have a different hash value or SQL ID when CURSOR_SHARING is set to SIMILAR or FORCE. When a statement with the same FORCE_MATCHING_SIGNATURE is found, its cursor can presumably be reused.

[3] Oracle Database Reference 10*g* Release 2 (10.2) B14237-03, pp. 1–33.

[4] Since Oracle11g, the default server-side cursor cache size is 50. It was 20 in Oracle10*g*.

[5] See **en.wikipedia.org/wiki/Lexical_analysis**.

Statement Types

The statements we are interested in are INSERT, SELECT, UPDATE, MERGE, and DELETE, since these would harm performance if they were executed frequently without bind variables. The following query segregates statements into two sets—one set with a valid (non-zero) signature and another with a signature value zero. This intermediate result is then grouped by the availability of a signature and the command type. The query answers the question of what statements may have a valid force matching signature.

```
SQL> WITH s AS (
SELECT decode(command_type,
  1, 'CREATE TABLE',
  2, 'INSERT',
  3, 'SELECT',
  6, 'UPDATE',
  7, 'DELETE',
  26, 'LOCK TABLE',
  47, 'PL/SQL Block',
  48, 'SET TRANSACTION',
  170, 'CALL',
  189, 'MERGE',
  to_char(command_type)
  ) AS command_name,
  decode(force_matching_signature,
  0, 'unavailable', 'available') AS availability
  FROM v$sql
)
SELECT availability, command_name, count(*) AS count
FROM s
GROUP BY availability, command_name
ORDER BY availability, command_name;

AVAILABILITY    COMMAND_NAME       COUNT
------------    ---------------    -----
Available       CREATE TABLE           2
Available       DELETE               173
Available       INSERT                59
Available       MERGE                  1
Available       SELECT              4140
Available       UPDATE              2637
Unavailable     CALL                   2
Unavailable     DELETE                14
Unavailable     INSERT               119
Unavailable     LOCK TABLE             5
Unavailable     PL/SQL Block         218
Unavailable     SELECT                84
Unavailable     SET TRANSACTION        1
Unavailable     UPDATE                14
```

The query result above shows that most of the interesting statements do have a force matching signature most of the time. In fact, when taking a closer look at statements that have a signature value of zero, it turns out that most of these do have bind variables and have been reused multiple times. Unfortunately, the signature of INSERT statements without bind variables is always zero. The next query counts statements without a valid force matching signature that have been reused.

```
WITH s AS (
SELECT hash_value, sum(executions) AS count
FROM v$sql
WHERE force_matching_signature=0
AND command_type in (2, 3, 6, 7, 189)
GROUP BY hash_value
)
SELECT
reuse, count(*)
FROM (SELECT decode(count, 1, 'not reused', 'reused') AS reuse FROM s
)
GROUP BY reuse;

REUSE         COUNT(*)
----------    --------
not reused          26
reused             150
```

The above result shows that most statements without a valid signature have been reused multiple times.

Semantically Identical Statements

The next example proves that semantically identical statements with different literals differ in their hash values but match on the force matching signature.

```
SQL> SELECT force_matching_signature AS force_match_sig,
to_char(hash_value, '9.9EEEE') AS HV,
replace(sql_text, '* FROM departments WHERE', '...')
AS sql_text
FROM v$sql
WHERE force_matching_signature=15970513317740138009;

FORCE_MATCH_SIG       HV       SQL_TEXT
--------------------  -------  -------------------------
15970513317740138009  3.7E+08  SELECT ... department_id=4
15970513317740138009  2.4E+08  SELECT ... department_id=2
15970513317740138009  1.6E+09  SELECT ... department_id=1
```

Note that the SQL ID, which is not shown above, is also different for statements that contain different literals. Next up is a query that identifies non-sharable SQL statements in the shared pool based on their signature. The query also reports how many statements that differ merely by literals are in the shared pool. The query may easily be extended to show how much memory is used by those statements. Such a query is a good starting point to persuade a software manufacturer to use bind variables.

```
WITH f AS (
SELECT force_matching_signature, count(*) AS count
FROM v$sql
WHERE force_matching_signature>0
GROUP BY force_matching_signature
HAVING count(*) > 1000
)
SELECT force_matching_signature AS force_match_sig,
replace(sql_text, ' departments SET department_id=department_id WHERE
department_', '...') AS sql_text,
count FROM (
  SELECT f.force_matching_signature, s.sql_text, count, RANK() OVER
(PARTITION BY s.force_matching_signature ORDER BY s.hash_value) AS
rank
  FROM v$sql s, f
  WHERE s.force_matching_signature=f.force_matching_signature
)
WHERE rank <=3;

FORCE_MATCH_SIG          SQL_TEXT               COUNT
---------------------    ----------------       -----
14706988439514424098     UPDATE...id=9455        7630
14706988439514424098     UPDATE...id=7361        7630
14706988439514424098     UPDATE...id=6348        7630
```

The use of the analytic function RANK makes it possible to show three example statements per each non-reusable statement with a certain force matching signature.

Aggregation

When aggregating the resource consumption of non-reusable statements it is important to remember that most reusable statements also have a force matching signature greater than zero. In order to look at non-reusable statements only, I have used the predicate V$SQL.EXECUTIONS=1 in the query below. Thus the query only considers statements that have been executed once and then aggregates such statements over their force matching signature. Due to space constraints I have commented out several performance metrics such as buf-

fer gets and disk reads that are available from V$SQL in the query below.

```
SQL> WITH f AS (
SELECT command_type, force_matching_signature, sum(executions) AS
exec_count FROM v$sql
WHERE force_matching_signature>0
AND executions=1
GROUP BY command_type, force_matching_signature
HAVING sum(executions) > 1000
)
SELECT decode(f.command_type,
  1, 'CREATE TABLE',
  3, 'SELECT',
  6, 'UPDATE',
  7, 'DELETE',
  189, 'MERGE',
  to_char(f.command_type)
  ) AS command_name,
f.force_matching_signature, sum(executions) exec_calls,
/*sum(parse_calls) parse_calls, sum(fetches) fetches, sum(disk_reads)
disk_reads,
sum(buffer_gets) buffer_gets, sum(rows_processed) "ROWS",
round(sum(elapsed_time/1000000),3) elapsed_secs, */
round(sum(cpu_time/1000000),3) cpu_secs
FROM v$sql s, f
WHERE s.force_matching_signature=f.force_matching_signature
GROUP BY f.command_type, f.force_matching_signature
ORDER BY cpu_secs DESC

COMMAND_NAME  FORCE_MATCHING_SIGNATURE  EXEC_CALLS  CPU_SECS
------------  ------------------------  ----------  --------
DELETE            342201937948462289        4899     3.572
UPDATE          14706988439514424098        3474       2.6
```

Limitations

Thousands of SQL statements that are never reused will quickly age out of the shared pool. This is an inherent limitation of the method presented herein. Ideally, the forced matching signature would be part of the SQL trace file format. Then SQL trace profilers like TKPROF might easily aggregate the resources consumed by non-sharable statements. Oracle has added the SQL ID (sql_id) to the SQL trace file format in Oracle11*g* version 11.1.0.6 and the plan hash value (plh) in 11.1.0.7. Why not add the force matching signature? Of course it should also be made available for non-sharable INSERT statements.

Alternative Solutions

As with other types of performance problems, the use of SQL trace instead of V$ views may prove to be superior. As stated in the preceding sections there is currently no solution for the following two issues:

1. Non-sharable statements quickly age out of the shared pool and hence will be removed from V$SQL.
2. There is no V$ view that has a valid force matching signature for non-sharable INSERT statements.

SQL trace solves problem 1, since it captures all statements of a session or multiple sessions that meet certain criteria such as a common service name and module.[6]

The solution for item 2 requires third-party software that is capable of parsing the undocumented SQL trace file format as well as the relevant SQL statements (SELECT, INSERT, etc). According to the company websites, Hotsos and Method-R Corporation offer this capability in their Profiler product (**www.hotsos.com; www.method-r.com**).

I'm currently in the process of re-implementing my Perl-based SQL trace profiler, which ships with *Secrets of the Oracle Database* in Java, and have recently implemented the aggregation of non-sharable SQL statements as a licensable feature. This tool—I'm calling it the MERITS Profiler since it supports the MERITS performance optimization method that I describe in the book—is capable of reporting the force matching signature. Additionally it uses its own lexer, parser, and hash value to aggregate non-sharable statements, including INSERT statements. Additional information on the MERITS Profiler will soon become available on my website, **www.oradbpro.com**.

Summary

SQL statements without bind variables have a negative impact on performance and scalability. The force matching signature, a new column in V$SQL, is presumably a hash value that is calculated while ignoring literals. Hence statements that are semantically identical have the same force matching signature. This article discusses how to identify such statements and how to aggregate the resources consumed by them. In the past, the identification of non-shareable statement was difficult to automate. I have seen recommendations to group statements by the first 80 characters and the like. Nothing that worked was available out of the box. Except for INSERT, the problem is now solved. Third-party SQL trace profiler tools that are capable of parsing SQL trace files and the SQL statements contained therein provide an all-encompassing solution to the problem. ▲

*Norbert Debes has more than 13 years experience as an Oracle database administrator. He holds a master's degree in computer science from the University of Erlangen, Germany, and is an Oracle8, Oracle8*i*, and Oracle9*i *certified professional Oracle database administrator. For over 6 years, he held different positions and technical roles at Oracle Germany. He was a team leader in Oracle Support Services, and a technical account manager in Strategic Alliances. In his last role at Oracle, Norbert was responsible for promoting Real Application Clusters on a technical level. During his tenure, he contributed to the Oracle 9*i *SQL Reference, the Real Application Clusters manual set, and various Real Application Clusters training materials.*

In his spare time, Norbert likes to hike, snowboard, play basketball, and read nonfiction on topics such as the emotional brain. Furthermore, he is a passionate analog and digital photographer. Having been intrigued by the vibrancy of stereoscopic (i.e., three-dimensional) capture for 20 years, he rejoices in his recent acquisition of a stereo camera.

[6] Please refer to the documentation on the package DBMS_MONITOR.

Many Thanks to Our Sponsors

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

Long-term event sponsorship:

Chevron

Oracle Corp.

Thank you! Year 2009 Gold Vendors:

- Burleson Consulting
- Confio Software
- Database Specialists, Inc.
- Enteros
- Precise Software Solutions

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

TREASURER'S REPORT

Naren Nagtode, *Treasurer*

Beginning Balance		
April 1, 2009		**$ 45,469.53**
Revenue		
Membership Dues	1,411.00	
Meeting Fees	800.00	
Vendor Receipts	250.00	
Advertising Fee	–	
Training Day	–	
Sponsorship	–	
Interest	4.80	
Paypal balance	–	
Total Revenue		**$ 2,465.80**
Expenses		
Regional Meeting	7,664.83	
Journal	5,505.33	
Membership	–	
Administration	1,840.00	
Website	–	
Board Meeting	617.66	
Marketing	100.00	
Insurance	–	
Vendors	–	
Tax	800.00	
Training Day	–	
Accounting	–	
Miscellaneous	–	
Total Expenses		**$ 16,527.82**
Ending Balance		
June 30, 2009		**$ 31,407.51**

NoCOUG Spring Conference

Session Descriptions

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

Keynote

Oracle 2020: A Look at How Oracle Will Change in the Next Decade
Donald Burleson, Burleson Consulting.....9:30–10:30

The advances in hardware and Oracle automation features are going to have a huge impact on the job duties of the Oracle professional. This presentation explores industry trends to show how the job of the DBA will move beyond compartmentalized duties and into a broader spectrum. The Oracle Professional of the 21st century will be relieved of the tedium of monitoring and tuning and be free to concentrate on other important database administration activities. This fun and interesting presentation will give the attendees a look at how their jobs are going to change, sooner than they think.

*Donald K. Burleson is one of the world's best-known Oracle authors. A full-time DBA for more than 25 years and a retired adjunct professor emeritus, he has authored more than 30 books on Oracle database management, published hundreds of articles in national magazines, and is a popular lecturer at international database conferences. As a corporate database consultant, Don has worked with numerous Fortune 500 corporations creating robust database architectures for mission-critical systems. Don serves as CTO of Burleson Consulting (***www.dba-oracle.com**) and offers a popular remote DBA service (**www.remote-dba.net**).

Room 1220

Creating a Self-Tuning Database
Donald Burleson, Burleson Consulting..... 11:00–12:00

With the release of Oracle9*i*, Oracle started to create the foundation for a self-tuning database, and Oracle11*g* has further enhanced the automation of many tuning actions. Using the existing data from the Automated Workload Repository and Automatic Session History tables, this presentation will show you how to create sophisticated scripts to detect anomalies and how to dynamically invoke the dbms_scheduler utility to automatically repair the problem before it cripples the database instance. This presentation is for Oracle professionals who want to know how to automate their manual decision rules within the automation framework of Oracle 11*g*. This presentation will show working code from real-world Oracle 11*g* databases.

What to Expect from the Oracle Optimizer When Upgrading to Oracle Database 11*g* Release 2
Maria Colgan, Oracle Corp. 1:00–2:00

One of the most daunting tasks for a DBA is to upgrade the database to a new version. Having to comprehend all of the new features and deal with potential plan changes can be overwhelming. The purpose of this session is to dispel some of the mysteries surrounding the query optimizer by explaining in detail the new optimizer features, including SQL Plan Management, and what you can expect when you upgrade to Oracle Database 11*g*. It will also include step-by-step instructions to help you prepare for the upgrade.

Things You Always Wanted to Know About Oracle Partitioning
Hermann Baer, Oracle Corp......2:30–3:30

Partitioning is a key technology for addressing the requirements of large data volumes, for data warehouse as well as OLTP environments. Benefits are not only for performance but also increasingly for manageability and Information Lifecycle Management. This session will reveal best practices and designs used by successful customers. Furthermore, it will provide insight into less-known details of how to get the best leverage out of Oracle Partitioning.

DBA 101: Interpreting SQL Query Execution Plans
Iggy Fernandez, Database Specialists.....4:00–5:00

SQL efficiency is central to database efficiency, and the ability to interpret SQL query execution plans is a critical skill of the database administrator. In this session, we review the process of generating and interpreting query execution plans; the meaning of operations such as "Merge Join," "Hash Join," "Hash Group By," and "Index Fast Full Scan"; and how to monitor changes in query execution plans using Statspack and AWR data. We also discuss how to generate graphical versions of query plans, which are much easier to read than their more common tabular counterparts.

Iggy Fernandez is an Oracle DBA with Database Specialists and has more than ten years of experience in Oracle database administration. He is the editor of the quarterly journal of the Northern California Oracle Users Group (NoCOUG) and the author of Beginning Oracle Database 11*g* Administration*, published by Apress.*

Room 1240

Running Oracle in EC2 **EDITOR'S PICK**
Ahbaid Gaffoor, Amazon.com 11:00–12:00

In this session you'll learn how to set up an Oracle database on an EC2 instance, configure access, and have it persist across reboots. We'll also look at S3 (Simple Storage Service) for RMAN-based backups in the cloud. Expect to leave this session with the tools to deploy your next Oracle instance in the cloud. We'll talk a bit about Oracle licensing in the cloud, then look at Amazon Web Services' cloud offerings, including EC2 (Elastic Cloud Compute), Elastic IP, Elastic Cloud Front, and S3 (Simple Storage Service).

Tuning a Multi-Terabyte Database for High Performance: An Architectural Approach
Daniel Liu, Oracle Corp. 1:00–2:00

The size of database systems has grown exponentially in the past few decades. Do you want to know how to design and tune a multi-terabyte database for high performance? How to manage a hybrid database with both OLTP and OLAP data? This session takes an architectural approach to examining the following areas: storage layout, network pipeline, server and system setup (memory and CPU), physical database setup, logical database design, and application tuning. It provides tips and tricks on tuning a database for better performance. It also shows how to take advantage of Oracle products and features (Enterprise Manager, Exadata, Real Application Clusters, Real Application Testing, Partitioning, Advanced Compression, etc.) to deliver high performance.

Daniel Liu is a principal solution architect at Oracle Corporation and co-author of Oracle Database 10g New Features by Rampant TechPress. A recognized Oracle expert and a frequent speaker at various Oracle conferences, Daniel has published articles with DBAzine, Oracle Internals, Oracle Technology Network, *and* SELECT. *Daniel received the* SELECT *Editorial Award for Best Article in 2001 and was named Architect of the Week by the OTN in 2004. Prior to joining Oracle Corporation, he worked as a senior technical manager at First American, managing one of the largest and most complex database environments in the world.*

The Latest Oracle 11*g* Gems
Daniel Morgan, University of Washington 2:30–3:30

Oracle ACE Director Daniel Morgan will dispense with the PowerPoint slides and give a live demo of new and valuable capabilities in the latest release of the Oracle database.

Daniel Morgan is the Morgan of Morgan's Library on the Web, an Oracle Ace Director, and the education chairman of PSOUG. Daniel develops curricula and teaches the Oracle Basics and Advanced Oracle Application Development programs at the University of Washington. He is a member of UKOUG, the British American Chamber of Commerce, and former leader of the Washington Software Alliance's Database Special Interest Group. A regular contributor at monthly PSOUG meetings, Daniel has spoken at Open World, UKOUG's annual conference, and at user group events in Canada, California, Oregon, and Minnesota.

Closing the Privacy Gap: How Safe Is Your Data?
David Alexander, IBM Optim.....4:00–5:00

Data protection and privacy continue to be a tremendous focus and risk for the IT community today. While companies are making great strides to protect data privacy in production application environments, the story of implementing similar strategies in non-production (testing, development and training) environments is often overlooked. Bridging this "privacy gap" helps companies protect the most exploited areas of an organization's IT infrastructure—non-production application environments. In this session, attendees will learn strategies that can be deployed in the testing environment to support compliance initiatives and how to leverage data-masking techniques as part of a data management strategy.

Room 1140

Tuning PL/SQL Using DBMS_PROFILER
Tim Gorman, Evergreen Database Technologies 11:00–12:00

Beginning in Oracle 8 v8.0, the DBMS_PROFILER package has offered the ability to tune the performance of PL/SQL programs themselves, outside of the SQL statements they call (which are best tuned with SQL tracing).

*Tim Gorman has worked in IT on relational databases since 1984, as an Oracle application developer since 1990, and as an Oracle DBA since 1993. Tim is an independent consultant (***www.EvDBT.com***) specializing in performance tuning, database administration (particularly availability), PL/SQL development, and data warehousing. He has been an active member of RMOUG since 1992 and has been a board member since 1995, holding most of the positions, including president. He has co-authored three books,* Oracle8*i* Data Warehousing, Essential Oracle8*i* Data Warehousing *(both from John Wiley & Sons) and* Oracle Insights: Tales of the OakTable *(from Apress). Tim has presented at Oracle Open World, Collaborate, UKOUG, Miracle Database Forum, and Master Classes, as well as local Oracle user groups in North America and the Caribbean.*

Introducing Database Modeling and Design with Oracle SQL Developer Data Modeler
Kris Rice, Oracle Corp. 1:00–2:00

Oracle SQL Developer Data Modeler supports logical and physical data modeling for Oracle, Microsoft SQL Server, and IBM DB2. This addition to the Oracle SQL Developer family of tools provides forward and reverse engineering of database structures for all who work with graphical data models. In this session, see how to create a logical entity relationship diagram, with a choice of Barker or Bachman notations, and forward engineer the design to one or more relational schema diagrams. The session reviews various diagramming options and the set of Design Rules provided to help ensure that your models comply with a set of standards. You hear about the implementation-specific physical models and review the DDL generated for the models designed.

Kris Rice is the architect and director for Oracle SQL Developer. He joined Oracle Corporation in 1998 and has worked in various groups, including consulting, Oracle Applications development, and Application Express development. He has been using Oracle database since 7.1.3 and has been a Linux user as long, starting with SuSE 4.0.

Everyday Tasks with Oracle SQL Developer
John McGinnis, Oracle Corp......2:30–3:30

Oracle SQL Developer provides database developers with a powerful tool for database tasks. With too many features to demonstrate, this session demonstrates one scenario that database developers might encounter, touching many areas of the tool to illustrate the diversity and features it offers. The highlights include the SQL Worksheet, with its code insight, snippets, and templates; Reports; Oracle APEX integration; general schema copy and compare; and the integrated file navigator and source code control support. The latest release of SQL

(continued on page 17)

NoCOUG
P.O. Box 3282
Danville, CA 94526

NoCOUG Spring Conference Schedule

August 20, 2009, at Chevron, San Ramon, CA

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

8:00 a.m.–9:00	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Hanan Hit, NoCOUG president
9:30–10:30	**Keynote:** *Oracle 2020: A Look at How Oracle Will Change in the Next Decade* —Donald Burleson, Burleson Consultation
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Room 1220: *Creating a Self-Tuning Database*—Donal Burleson, Burleson Consulting
	Room 1240: *Running Oracle in EC2—Ahbaid Gaffoor, Amazon.com* **EDITOR'S PICK**
	Room 1140: *Tuning PL/SQL Using DBMS_PROFILER*—Tim Gorman, Evergreen Database
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Room 1220: *What to Expect from the Oracle Optimizer When Upgrading to Oracle Database 11g Release 2* —Maria Colgan, Oracle Corp.
	Room 1240: *Tuning a Multi-Terabyte Database for High Performance: An Architectural Approach* —*Daniel Liu*
	Room 1140: *Introducing Database Modeling and Design with Oracle SQL Developer Data Modeler* —Kris Rice, Oracle Corp.
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Room 1220: *Things You Always Wanted to Know About Oracle Partitioning*—Hermann Baer, Oracle Corp.
	Room 1240: *The Latest Oracle 11*g *Gems—Daniel Morgan, University of Washington*
	Room 1140: *Everyday Tasks with Oracle SQL Developer*—John McGinnis, Oracle Corp.
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Room 1220: *DBA 101: Interpreting SQL Query Execution Plans*—Iggy Fernandez, Database Specialists
	Room 1240: *Closing the Privacy Gap: How Safe Is Your Data?*—David Alexander, IBM Optim
	Room 1140: *Anatomy of a Database Attack*—Dana Tamir, Imperva
5:00–	**NoCOUG Networking and No-Host Happy Hour at Izzy's Steaks and Chops, 200 Montgomery Street, San Ramon**

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

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo: Wheat field in Southern California. Photo by Tim McCabe, USDA Natural Resources Conservation Service.