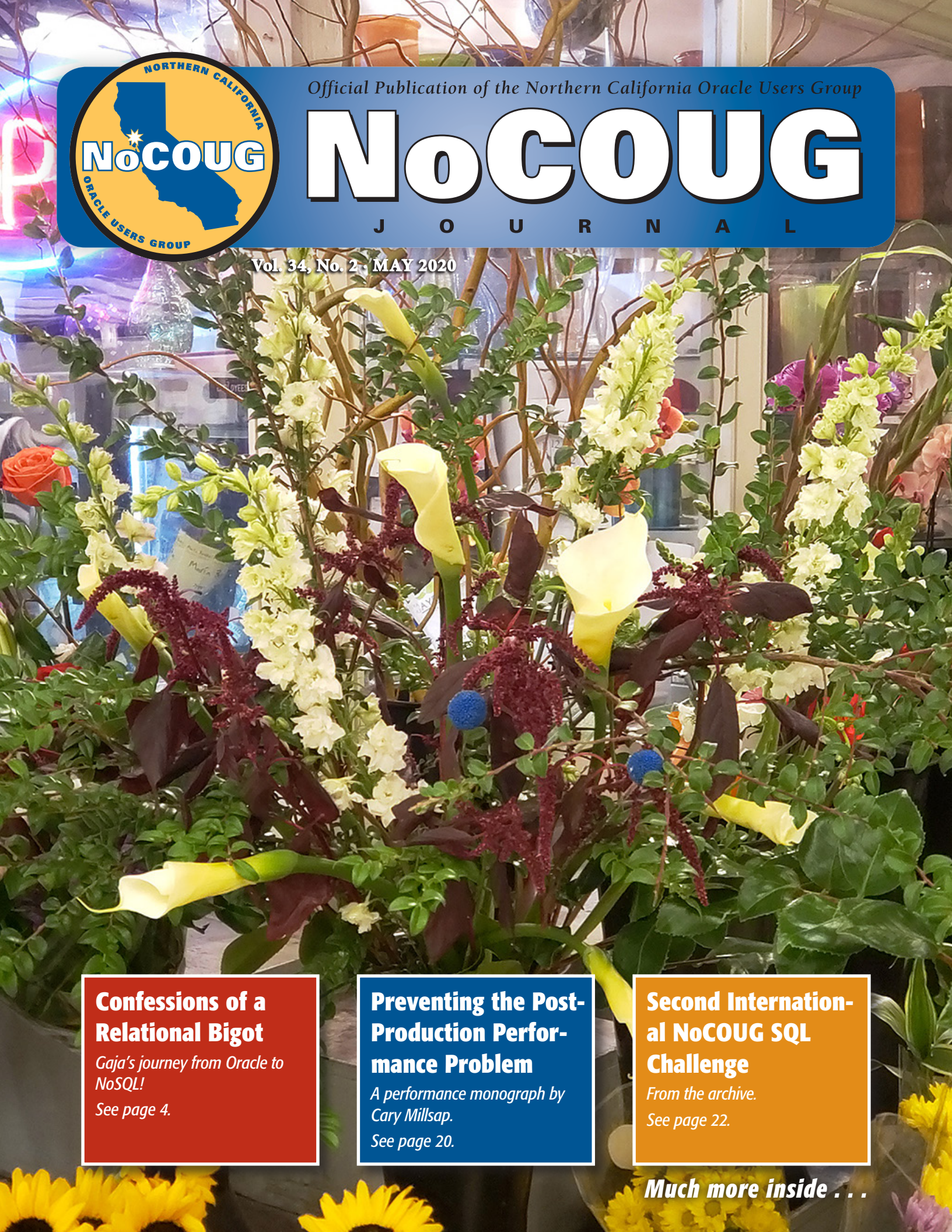
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 34, No. 2 · MAY 2020



Confessions of a Relational Bigot

Gaja's journey from Oracle to NoSQL!

See page 4.

Preventing the Post-Production Performance Problem

A performance monograph by Cary Millsap.

See page 20.

Second International NoCOUG SQL Challenge

From the archive.

See page 22.

Much more inside . . .

Amazon Aurora

Performance and availability of commercial-grade databases at 1/10th the cost.

Learn more: aws.amazon.com/aurora

aws

Professionals at Work

First there are the IT professionals who write for the *Journal*. A very special mention goes to Brian Hitchcock, who has written dozens of book reviews over a 12-year period.

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

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

And, finally, the *Journal* is printed and shipped to us. This is the 134th issue of the *NoCOUG Journal*. Enjoy! ▲

Table of Contents

Publication Notices and Submission Format

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

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

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

2020 NoCOUG Board

ADVERTISING RATES

Confessions of a Relational Bigot

My Journey from Oracle to NoSQL

by Gaja Krishna Vaidyanatha

Gaja Krishna Vaidyanatha

Greetings of love, peace, and harmony! Thank you for taking the time to read my professional musings. I will share thoughts and experiences relating to my work life and the two database technologies that I have had the pleasure of working with. I have no intention of starting a war of words or a technical debate. This is a platform where I wish to share my journey in the database world. I seek your patience and understanding if what I share is not congruent with your own thoughts and perceptions. I am a certified relational bigot and an evolved data practitioner. There, I have come out and said it! This technology-induced bias was formed early in my professional upbringing with the Oracle RDBMS. Despite this bias, one thing is still clear—change is the most permanent thing. We are, after all, living in a time-space continuum. Nothing stays constant, and we need to continually grow and evolve.

In my early work years you couldn't have convinced me to design or build anything without using an Oracle database. In my mind, nothing was beyond Oracle! And, for the most part during the first two decades of my professional life, that was true. I was blessed with the many learning opportunities that have been bestowed upon me in a technical career that has spanned over 27 years (25 years of relational: Oracle 6 to Oracle 12*c*). I am very thankful for that!

However, an important transformation occurred in 2017. I came face to face with a data integration problem that required a different approach. I knew that if I forced the issue and traveled down my usual relational route, there was a high probability of project failure. For the first time, I was overwhelmed by a huge sense of discomfort. And that was a sign of the need to grow and get out of my comfort zone. I pondered the problem and spent a great deal of time in research. The solution to the problem that surfaced (time and again) was a NoSQL database. For a relational bigot like me, entering the NoSQL world was analogous to changing my religion. But I had to open myself, as I had no plans of intentionally failing. At least I could not fail due to my own denial of the facts from the technical research I had undertaken. The problem warranted a change in my approach and thought process.

The data landscape in the past 25 years has undergone many a transformation. This metamorphosis requires relevant data management techniques to change as well. When you know exactly what you want to store (numbers, character strings, and dates, for the most part), how you want to store it, and how much you want to store—and there is a relatively static data model that supports the data persistence method—there is no better option than a relational database. An RDBMS is all about table structure and organization (columns and rows). Yes, it has grown and matured to support other data aspects over the last few decades, but its foundation has not changed. Relational databases came into being from set theory, an integral component of structured mathematics. Here is some additional reading on the topic for you: **en.wikipedia.org/wiki/Structure_(mathematical_logic)**.

The lack of a continually evolving physical enterprise data model poses all sorts of challenges for data integration. Data models (more often than not) are not kept up to date with the business models. This is due to the fact that the rate of change in the business is difficult to match in the relational world. It makes it super hard to implement a single source of the data truth sourced from disparate data sources when business models continually change. To state the obvious: if the data model does not continually evolve, the benefits of your data integration efforts will be short lived. This is because the world we live and work in is a continuous flow of change. Data persistence (in the form of business objects delivered to the enterprise) needs to continually change with time and keep up with the business.

If the data model is frequently forced to change by business factors, there is a definite need for a different approach from a relational database. Further, the fragmentation of data stores across the enterprise today, caused by a historically undisciplined

"In my early work years you couldn't have convinced me to design or build anything without using an Oracle database. In my mind, nothing was beyond Oracle! And, for the most part during the first two decades of my professional life, that was true. However, an important transformation occurred in 2017. I came face to face with a data integration problem that required a different approach."

approach to data management (fueled either by lack of data governance and/or corporate politics), will push any relational data integration database to its breaking point. How is a relational database expected to deliver the goods in a consistent fashion without the required structure of a stable data model? Relational databases were neither designed for fluid data models nor for natively handling the transformation challenges of today's businesses and today's data.

When faced with the above challenge, a NoSQL database—with its inherent flexibility and evolutionary approach to harmonization (both of the data model and the data)—provides a reasonable solution to a complex technical problem. Having taken ACID transactions for granted for many years, I looked for a NoSQL database that supported ACID functionality across multiple rows or objects. The flexibility of a NoSQL database's persistence layer that would evolve with time is a game changer in this space. After over 25 years of data management in the relational realm, my next round of professional evolution had begun.

The Early Days (1992–1997)

Spring had just sprung in Toledo, Ohio, in the month of May. The year was 1992. Nature was working overtime to make itself seen, having spent many months under the cloak of a hard winter. The extent of the transformation that the spring of 1992 would bring upon me wouldn't be known to the fullest extent until many years later. I was pursuing my master's degree in computer science at Bowling Green State University (BGSU), Bowling Green, Ohio. It was (and still is) a fantastic school for computer science (among other majors), but the U.S. job market at the time was jittery, fueled by the 1991 recession and the Persian Gulf War. As a rite of passage, I applied and interviewed for a few internships at BGSU's on-campus job fairs. Unbeknownst to me, the company that chose me (and I chose) for my internship would play a pivotal role in transforming my professional life.

Transitioning from C to Relational Data Management

On my first day at Owens-Corning Fiberglas (OC) in Toledo (Fiberglas Tower, 200 North Saint Clair St.), I was informed that my assigned manager was Mark Amos. I was hired as an intern for my C programming skills and was originally told that I was going to be developing a system in C. I had not interviewed with Mark and thus was not sure what to expect. After exchanging pleasantries, Mark's first question to me was, "Do you know Oracle?" Sheepishly, I replied, "You mean, am I into fortune telling?" Until that day, I had never heard of relational databases, let alone Oracle. Mark read the humor of the situation, smiled ear to ear, reached under his desk, and handed me an Oracle Computer Based Training (CBT) package.

At the end of my first two weeks, my right thumb was sore from continually hitting the space bar on the keyboard (required to advance the CBT program). It was a Friday and I reported back to Mark that I had successfully completed the Oracle CBT. He said, "Good, now you can go and build us an Executive Information System (EIS). And oh, by the way, we will be using Lightship as the client. Let me know if you have any questions. Have fun!" I thus was thrown into my first technical deep-end-must-learn-to-swim experience. I was back in the office early on Monday morning, and I began to swim my first professional lap.

Projects, Oracle 6, Windows 3.0 Client Memory Management, and OSI Model

My relational journey as part of building an EIS started with Lightship (client-side BI software that no longer shows up on a Google search), which connected to an Oracle Database 6.0.31, housed in an HP 9000 Series 500 server running a proprietary version of Unix—HP-UX. Here is some fun reading of hardware specifications from way back when: **en.wikipedia.org/wiki/HP_9000**.

From the get-go, I was very comfortable with the server side of things. I loved *vi* as I saw the indomitable power of the Unix editor that could run shell commands from within—not to mention that its support for 26 clipboards (copy-paste buffers that can be named from A to Z) is unprecedented. After *vi*, came the editors *vim*, *emacs*, and *jove* into my work life. Unix became my happy place, despite the lack of a graphical user interface on my desktop. In those days the spiffy and fancy Unix workstations were meant only for the system administrators. "Ah, the life of a sys admin: she gets to play with Unix using a GUI!" I thought to myself.

After successfully developing and deploying the EIS on Lightship and Oracle in the first six months at OC, I went on to my next adventure. Mark asked me to build an email-based report delivery system using SMTP, KornShell scripts, and SQL. Basically, there was a need to publish nightly reports via email using the data that was collected during the day. Thus began the building of an SMTP-based reporting system. The AWK utility (more like a programming language) became an indispensable part of the code stack, with its incredible power and brevity for data manipulation and extraction. The server-side gravitational pull was stronger than ever, and the journey to become an Oracle database administrator continued.

Charlie Mather, a seasoned campaigner at OC and my mentor, taught me how to write complex SQL. He and I shared many a lunchtime meal discussing all sorts of technical topics. Charlie was a great guy to work with and was always supportive of my unending list of questions. He was also the one who introduced me to Forest and Trees (analytics client software that breathed its last in 1999).

The Oracle RDBMS 6.0.31 was quite primitive compared to the functionality that it boasts these days. Oracle 6 had only two types of indexes:unique and non-unique. As a matter of fact, the database did not even have constraints, which were later supported in Oracle 7. Thus, to implement a primary key constraint in Oracle 6, you had to create the relevant column(s) as NOT NULL and then create a UNIQUE index on the said column(s). This feels surreal when you compare the support for pluggable databases in Oracle 12*c* and beyond.

Those were also the days when Oracle Corporation used to ship its software on floppy disks via Federal Express. I still remember how the 47th diskette of a 48-disk pack would invariably be corrupt and unreadable. Once in a while a little wiggling of the diskette brought it back to life. You considered yourself fortunate if that happened. More often than not, one had to wait another three days for the replacement pack to show up, delete everything you did so far, and start over. If you could imagine a world without product-download websites, MetaLink, or OTN, that would be then. Dial-up modems had staggering download speeds of 19.2 Kbps—if you could download something overnight without the connection hanging up, your stars were defi-

"The fragmentation of data stores across the enterprise today, caused by a historically undisciplined approach to data management, will push any relational data integration database to its breaking point. Relational databases were neither designed for fluid data models nor for natively handling the transformation challenges of today's businesses and today's data."

nitely lined up. I am not sure whether the database technologists of today can even relate to this.

The Oracle 6 documentation was shipped as books (another FedEx package), and the entire Oracle 6 library fit on your desk. Today, if printed, the Administrator's Guide and Reference Manual will pretty much occupy the same amount of desk real estate. Those days, when you said you "knew" Oracle, you pretty much knew *all* of it, including but not limited to the database and its utilities (you may recall SQL*DBA), SQL*Net, the relevant Pro*Compilers, SQL, and PL/SQL. We had SQL*Net drivers/adapters for TCP/IP, SPX/IPX, IBM LU6.2, Novell, and DECnet. These days, Oracle is an ocean of applications, features, and functionality that make it impossible to know it comprehensively.

A fundamental requirement for maintaining your sanity while working with PCs those days was to proactively manage the 640K of base-addressable memory. We did this by using the Quarterdeck Extended Memory Manager (QEMM). QEMM assisted in the management of the order in which the device drivers were loaded from the Windows configuration files (AUTOEXEC.BAT and CONFIG.SYS). This was to ensure the diligent use of the precious 640K of core memory. The ultimate goal was to generate the fewest number of holes or fragments in core memory. We had to engage in this activity to get PCs to be useful, despite the prevailing urban legend at the time that 640K ought to be enough for anybody.

One networking issue that I'll never forget is when my Windows 3.0/3.1 client would render itself completely useless and display the infamous blue screen of death (BSOD). This occurred when an attempt was made to save a SQL statement within Lightship. Lightship was connected to an Oracle 6 database via SQL*Net for TCP/IP. The BSOD was inevitably followed by the three-finger salute (CTRL+ALT+DEL) and a coffee break, as it took about 10 minutes for the PC to reboot. The IBM OS/2 WARP machine that I also had the pleasure of keeping company with in my cube took even longer during reboots.

Without any idea why my computer was freezing and becoming inoperable so frequently, I approached Mark. I explained to him what was going on, and after patiently listening to my techno-rant, he thought about it for a bit and said, "Maybe you're losing your connection to the network." He had a theory that something in the SQL parser in Lightship was causing the problem. We then tested the theory on his machine. Sure enough, after 16 SQL syntax errors, Mark's PC froze just like mine.

After grabbing a cup of coffee and talking about his sailing pursuits, Mark quickly got down to troubleshoot the problem. He was able to unearth the TCP errors from the logs (my first real-world experience in troubleshooting), which then led him to proclaim that"16 SQL syntax errors cause your transport layer to become inoperable. You don't have any sockets left. There is probably a bug in the interface layer of Lightship and SQL*Net." When I requested a simple explanation in English, Mark drew the seven layers of the Open Systems Interconnection (OSI) model on a flipchart with two stacks on each end, one for the client and one for the server. He then went on to explain in great detail how each layer mapped to our installed technology stack. I carry that understanding of client-server, peer-to-peer computing, and networking with me to this very day. Mark was not only my manager but also my other mentor.

I am deeply indebted to Mark and Charlie for all of their contributions to my learning. I grew a lot technically during my 15 months at OC, and my hat is off to both of them. Much respect, gentlemen!

Oracle Education, Oracle 7, and Drinking from a Firehose

On May 23, 1994, a couple of weeks after completing my master's degree, I joined Oracle Corporation's Cleveland, Ohio, office in their consulting division. After completing some basic training, I was on the bench for a few weeks. During this time, I asked my manager whether I could teach some classes for Oracle Education (OE). This was supposed to be a stopgap arrangement before I was assigned to a suitable consulting project.

The first course I picked up to teach was Introduction to Oracle 7, and in a matter of few weeks I was able to gain a good level of mastery. After picking up Advanced SQL Queries, one thing led to another, and soon I started teaching Oracle Database Administration I and II, following that with Oracle 7 Performance Tuning. If my first rapid learning experience at OC was characterized as baptism by fire, the learning stint at OE could be characterized as baptism by fire while drinking from a firehose, but it was super gratifying. In December 1994, I officially transferred from Consulting to Education, after I had communicated to my respective managers on both sides my desire to learn, share, and teach Oracle for a living.

I loved reading the *Oracle 7 Concepts* manual and did so many times. I also enjoyed playing with the database more than ever before. I had access to training databases where I knew I could go all out. I attended database internals boot camps and got my hands on every shred of relevant information and documentation on the database. If I may say so myself, I trained like a relational marine. As time progressed, I got better in my understanding of the internal workings of Oracle. The Friday afternoon course evaluations were a constant reiteration of what I did well and where I had to improve.

During the latter half of 1996 through early 1997 I had the pleasure of being part of a team that designed, developed, and reviewed the courseware as part of the Oracle 8 Beta Program. This was a great opportunity to learn the new version in detail, with time to read the *Concepts* manual over again. My years at OE laid a solid foundation for my understanding of Oracle. I had to learn the database much more deeply than the level at which I taught the courses. After a great ride at OE, I joined Stonebridge Technologies at Austin, Texas, in October 1997.

I would be completely remiss if I did not express my gratitude to some of my colleagues at OE: Scott Gossett, Chuck Muehlbrad,

With the Texas Oracle Education team (1996)

Inderpal Tahim, Stephen Jackson, Susan Jang, David Austin, and Keith Roshto. These folks contributed to my learning of the internal workings of the Oracle database. I also thank Yolanda Salas and Nancy Hall for believing in me during those early days.

The Growth Years (1998–2003)

The 1990s were a transformational period for computing, as the glory days of the mainframe ended and client-server/open-systems took center stage in every IT organization. The PC market was going gangbusters. There were quite a few people who became Dellionaires (either working for Dell Corporation or playing their stock). Others made their riches on Oracle, Sun Microsystems, Cisco, Microsoft, and International Business Machines (IBM), to name just a few. The relational database market was defined and dominated by five players: Oracle, Sybase, Informix, DB2 UDB, and SQL Server.

Relational Databases and the Internet Bubble

To this day I believe that Informix had quite possibly the best database technology (based on version 7–9). Informix defined and drove the object-relational space (with the Illustra acquisition) and boasted groundbreaking functionality. It was a great database that provided time series and spatial extensions for the usual relational fare in high-transaction-rate applications. At the time it was very evident that Oracle and Microsoft were both playing catch-up to Informix's innovations. Emblematic of the time were the billboard wars of the 1990s on California's Highway 101, along the Marine World Parkway exit. If Informix Corporation had gotten its act together, the relational market would be quite different today. But they were plagued with a lot of internal issues—ineffective marketing, lack of corporate governance, and executive wrongdoing. Informix was unable to stay in business on its own and was acquired by IBM for USD $1 billion in July 2001. In April 2017, IBM outsourced the future development of Informix to the Indian services company HCL, through a 15-year partnership agreement.

Similarly, Sybase had one of the best database replication technologies and was deployed in many high-transaction-rate environments (some of the largest logistics companies in the world used Sybase for their package tracking systems). It had a very loyal customer base but lacked the staying power in an ever-changing market. It too could not hold it together on its own. SAP eventually acquired Sybase for USD $5.8 billion in 2010.

In comparison, Oracle did really well, continually delivering a solid database product to the market. It demonstrated a relentless quest to be a market leader. During my early years, Oracle had approximately 40–48% of the relational database market. Although it was initially perceived as weak in the applications segment, the relaunch of Oracle Apps as a 100% Java product and the M&A activities from 2002 changed that perception. The acquisition of PeopleSoft, Siebel, and a slew of other companies gave Oracle the foothold and dominant position in the applications space.

The mid-to-late 1990s was also the era when relational databases were perceived as one of the coolest technologies to work with. Oracle was considered the flagship database product and dominated the market during the internet boom. You would be aggressively headhunted with multiple offers in the job market if you could just spell Oracle (today's "talent acquisition" equals yesterday's "headhunting"). Oracle skills were in such high demand that people started to move into the space in droves. This high demand created a false sense of job security. The stock market saw excessive speculation on anything that had the word "Internet" or the "e/E/i/I" prefix or suffix. This hype caused stock prices to rise to dizzying but artificial heights. The fundamentals of finance and business went out the window. This false sense of prosperity made the future very ominous. The market eventually came to its senses and crashed in 2001–2002. The false sense of job security quickly vanished when the bubble burst and dot-com companies fell by the wayside. Unemployment suddenly skyrocketed, and that meant a very hard landing for many people.

"The Oracle 6 documentation was shipped as books, and the entire Oracle 6 library fit on your desk. Those days, when you said you "knew" Oracle, you pretty much knew all of it. These days, Oracle is an ocean of applications, features, and functionality that make it impossible to know it comprehensively."

Proprietary Hardware and Unix

The Unix server market was dominated by Sun Microsystems (Sun), Hewlett-Packard (HP), Digital Equipment Corporation (DEC), and IBM. They all had their own proprietary chip architectures and their own flavors of Unix (Solaris, HP-UX, Digital UNIX/Tru64, and AIX, respectively). This technically handcuffed a customer whenever a single-vendor decision was made. Companies that went the route of a single vendor called themselves an IBM shop, a Sun shop, an HP shop, and so on. The larger companies, whose pockets were deep, had the luxury of playing with multiple vendors and then deciding which one best suited their needs. Dell Corporation and COMPAQ Corporation dominated the Windows server market. Most corporate entities aligned themselves with a primary server vendor.

"You can procure and deploy an enterprise-class server (albeit a virtual one) on AWS, Azure, or GCP, installed with your choice of the OS and database, in a matter of minutes. You also get to shrink and expand your system, pretty much at will. It is amazing how cloud computing has truly transformed the way hardware procurement and systems deployment is done."

One other vendor of Unix-based servers requires an honorable mention here: Sequent Computer Systems. Sequent had a symmetric multiprocessing computing platform and a Non-Uniform Memory Architecture (NUMA-Q)–based computing paradigm. It was considered state of the art for the time. Sequent's version of Unix was called DYNIX. I had the pleasure of working on a Sequent system in 1998, during one of my consulting projects. But then again, with the market rapidly changing and commodity hardware becoming the norm, Sequent was forced to shut its doors in 1998 when IBM bought it for USD $810 million. I will write about Silicon Graphics Inc. (SGI), another hardware vendor, in the next chapter, as it played a bigger part in my journey.

Procurement—Then and Now

Given the focus on hardware and servers in this chapter, let's talk about the 1990s' method of hardware procurement and systems deployment. During this period, project planning for systems always involved an on-site visit from your favorite hardware sales team. They took your requirements during your planning phase and came back with a quote for a server. The entire process of hardware procurement could take up to 16–24 weeks. It all depended on the size of the company you worked for, its budget and purchasing power, its level of influence with the vendor, and other geo-political factors. Not to mention the many weeks that you spent after the system was delivered installing all of the software and making it ready for use. You were stuck with the system that you procured, regardless of how it worked or was sized.

Compare that to today's world, where you can procure and deploy an enterprise-class server (albeit a virtual one) on Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP), installed with your choice of the OS and database, in a matter of minutes. You also get to shrink and expand your system, pretty much at will. It is amazing how cloud computing has truly transformed the way hardware procurement and systems deployment is done. This is great progress! If hardware provisioning at your company still takes 16 weeks, it is my humble opinion that the clock in your company has stopped in the 1990s.

RAID—More Than an Insecticide

For the first seven years of my life in the U.S. (1990–1997), the only RAID I knew was the insecticide that's found in the aisles of the grocery store. The term "Redundant Array of Inexpensive/Independent Disks" (the other RAID) came into my vocabulary in early 1998. I was working for Stonebridge Technologies as an Oracle consultant in Austin, Texas, and the hardware vendor at our customer site introduced me to RAID and the different flavors—RAID 0–7 (not counting RAID-S and AutoRAID). The popular ones were 0, 1, 0+1, 1+0, and 5.

The distributed parity of RAID 5 (in lieu of mirroring) and the overhead it posed worried me from the first time I read about it. I just could not fathom how this significant overhead, both during writes and during "rebuilding from partial failures in degraded mode," would contribute positively to the performance of write-intensive applications. Yes, it worked well for a predominantly read workload, but it was marketed as less expensive when compared to RAID 10 (mostly without including the cost of the cache). Storage vendors at the time did everything to mask the performance problem of RAID 5. They allocated large amounts of cache to their storage systems to mask the problem. More often than not, the Oracle performance issues encountered in a RAID 5 storage environment stemmed from inadequate storage performance capacity (IOPS and transfer rate), which then resulted in very high service times. The parity overhead stuck out like a sore thumb.

My Love for DEC and SGI

During a project with Stonebridge Technologies in 1997, the customer had Oracle 8 running on a DEC Alpha 8500 series with Digital UNIX. In the 12+ months that I was involved in the project, neither the OS nor the server had a single hiccup. Oracle also seemed to be a lot more stable (fewer issues when compared to its predecessor on HP-UX). And it was fast, with a discernible difference in elapsed times across all of the SQL in our core workload. It was a fantastic OS housed inside a fabulous server. Digital UNIX/Tru64 was my first taste of a 64-bit operating system. I absolutely loved it!

In early 1998, while still with Stonebridge, an SGI consultant introduced me to the Origin 2000 server—**en.wikipedia.org/wiki/SGI_Origin_2000**—at one of our customer sites. I was in absolute awe as I listened to him raving about its capabilities as a true high-performance computing platform that scaled for the needs of the time.It had its own flavor of Unix called IRIX, and it was also a true 64-bit OS, based on a node-based architecture, with up to 512 processors and 512 GB of memory. The nodes were linked with a non-uniform memory architecture (NUMA)

With the MFADS team and the SGI Origin 2000 (1998)

interconnect, and it had a Crossbow (XBOW) architecture for its I/O sub-system that could support a ton of I/O. Remember the timeframe: 1998.This was a server and a half!

As part of the 1 TB data warehouse that we were deploying in 1998, we looked at the available RAID options that the Origin 2000 offered. RAID 3 (striped volume with a dedicated bit-level parity disk) caught our attention. In all of our comparison tests between RAID 5 and RAID 3, the latter won by a significant margin (measured in elapsed time of the jobs, IOPS, average service time, and transfer rate). Armed with the results and solid performance data, we made the decision to implement the storage layer in RAID 3. We then refactored the code to leverage the great throughput capacity of the Origin 2000. We parallelized relevant operations within the Oracle database and optimized the code. There were batch jobs that previously did not finish in three days but now completed in 45 minutes. This mega-project, which was originally in "red" status, went to "green" in 45 days. The project went live on time! Thank you, Paul Scott and Kevin Embree for giving me the freedom to express myself at work. I thoroughly enjoyed our time together in Austin, and it was a pleasure working for both of you.

SGI's incredible media-streaming server from the 1990s had all the traits of a supercomputer and could have made a killing as *the* preferred platform for very large databases (VLDBs). In my mind they had the perfect server for data warehousing. They missed that *huge* opportunity and did not evolve when the x86-based commoditization began. Instead, they tried launching Windows NT workstations with their expensive hardware. Beats me as to why that made any business sense to anyone when Dell was selling cheaper commodity Windows NT PCs like hotcakes. At the time of the writing of this article the remains of SGI is owned by Rackable Systems. SGI filed for bankruptcy in 2009. Sigh.

The year 1998 remained a terrible one with the fall of another computing giant. It was the beginning of the end for DEC, as they sold the bulk of their products, IP, and assets to COMPAQ. DEC was first sold to COMPAQ, followed by COMPAQ being acquired by HP. Digital UNIX and SGI IRIX were some of the best flavors of Unix that I have worked on. At the time, they were miles ahead of HP-UX, Solaris, AIX, and DYNIX. I will go on record and state that they were even better than any flavor of Linux I have worked on.

Breaking Free

The underlying reasons why so many well-established hardware companies fell one after another were the same:commoditization of hardware, the role of Intel in the chip market, and the surge of x86 as an industry standard. X86 and all of its successors became a viable alternative for expensive and proprietary chip architectures. It is human nature to seek freedom when we are shackled. It was only a matter of time before we broke free from proprietary architectures. This quest for freedom is the fundamental reason why the "hardware bloodbath" occurred in the first place. It also reinforces the importance of two elements that are still relevant:

1. The need for continuous technical evolution
2. A tendency to voluntarily embrace mandatory market factors

In hindsight, IBM had the foresight to play their cards right and embrace the x86 revolution and Linux. They still stand strong after all these years. To a large extent, HP also has stood the test of time with its dominance in the blade server market. Sun, the darling server company of the dot-com age, does not exist in its original form anymore. The hardware vendors who evolved and opened their doors to commodity hardware and open-source Unix (Linux) have thrived. As for the others, they have made the annals of computing with historians writing about them. In the software space, clear messaging and marketing that are congruent with the market forces is important. Of course, credibility and integrity are required elements in business. When legal boundaries are pushed in pursuit of profits or personal greed, the consequences will be pretty dire.

You've Got Mail

If you haven't done it already, go ahead and watch the 1998 movie "You've Got Mail," featuring Meg Ryan and Tom Hanks, which qualifies as the early internet days. Those three words are what people loved hearing. The internet service provider America Online (AOL) announced that you had email in your Inbox each time you logged in. You will get a kick out of this one and will also appreciate how far we have come in 20+ years.

"The underlying reasons why so many well-established hardware companies fell one after another were the same:commoditization of hardware, the role of Intel in the chip market, and the surge of x86 as an industry standard. It is human nature to seek freedom when we are shackled. It was only a matter of time before we broke free from proprietary architectures."

Getting on the Information Superhighway and accessing the World Wide Web was a fashionable thing to do in the late 1990s, in order to check email and browse the world of information and marketing (the Internet, with a capital "I"). Thus began the era of dial-up modems with speeds up to 56 Kbps. The crackling sound of a dial-up modem became part of every household. Check this out for old times' sake: **www.youtube.com/watch?v=abapFJN6glo**. Digital Subscriber Line (DSL) and its successors soon followed suit, supporting much higher bandwidth and speeds. I also remember my first mobile phone, a Nokia 5110 released in 1998, **parade.com/5457/iraphael/the-evolution-of-the-cell-phone/**, which now looks like an ancient space gadget.

We browsed the web using Yahoo Search, Excite, AltaVista (a DEC contribution), or AskJeeves. Google came into my life in 2000, when a coworker showed me the power of a simple search box without any unnecessary marketing content. I have never looked back. Netscape Navigator was the competitive offering to Internet Explorer (IE). It was first released in 1994 and was considered the cooler browser. There were browser wars between IE and Netscape, each competing to outdo the other in every new

release. Mozilla Firefox came much later in 2002 and Google Chrome even later in 2008.

Oracle Performance Management—Time for Some Oracle Geek Speak

In 1999 I had the pleasure of rereading the Yet Another Performance Profiler (YAPP) paper written by Anjo Kolk. It changed my perspective and that of many others who worked on Oracle Database performance. The paper talked about the importance of the mantra *Response Time = Service Time + Wait Time*. This made a huge impression on me, and I felt the need to re-engineer my own thought processes in the space of Oracle performance tuning.

I embraced the Oracle Wait Interface (*v$system_event, v$session_event, v$session_wait,* and the *10046* diagnostic event) with both arms and started approaching Oracle tuning engagements very differently. I started to focus more on the application problem (SQL) and the step-level execution plan inefficiencies (thanks to *ROWSTATS* in Oracle 9*i*) and less on instance configuration. Yes, when set incorrectly certain Oracle parameters can cause performance issues, but SQL is responsible for 90% or more of most application performance problems.

This was the inflection point when I totally moved into the area of Oracle performance. In hindsight, what I taught in the Oracle 7/8 Performance Tuning classes for Oracle Education (1995–1997) was unnecessarily focused on the various *init.ora* parameters and cache-hit ratios instead of asking, "What resources is the SQL consuming and waiting for?" We need to continually filter the noise and pay enough attention to the important aspects of our lives.

The huge potential of the Oracle Wait Interface got me to author a paper and presentation titled "Oracle Performance Management," which I then presented at the International Oracle Users Group - Americas (IOUG-A) in 2000. Along with my peers at the Oaktable Network, I embarked on a two-year journey, where we extensively spoke about this alternate method at every possible conference. I even got my audiences to chant "Cache-hit ratios are for losers" at the beginning of the talk. Jokes aside, I wanted them to focus on SQL waits and logical I/O (Oracle's foreground CPU time is mostly spent here). Everything else was secondary. Those were exciting times. We made an impact and people saw the benefit of this approach. This laid the foundation for my first literary project: *Oracle Performance Tuning 101*, published by Oracle Press in 2001.

Storage Management for Oracle—The Israel Connection

My interest in the performance of Oracle combined with its interplay with the OS and storage continued as I delved deeper in the area of systems performance management. It was during one of those IOUG-A conferences in San Diego, Calif., that I met Eyal Aronoff, the CTO of Quest Software at the time. He was considered a product visionary in the Oracle tools market. He provided the vision for many products at Quest Software, including Space Manager, LiveReorg, Spotlight on Oracle, Quest Central, and SharePlex. We chatted after one of my Oracle on RAID presentations, and that resulted in an offer for me to work at Quest Software. My role was to head up the Storage Management Products Group and to provide technical and strategic direction for a new product line. This was my first stint at product management and the year was 2000. Although I still lived in Dallas, Texas, at the time, I was most definitely transported to the moon and back!

On my first visit to Tel Aviv, Israel, in 2000 (at the height of the Israeli-Palestinian conflict), I met the StorageXpert engineering team. I have some very interesting stories to tell you on how I did StorageXpert product demos for the Israeli Security Forces at Ben Gurion Airport at 3:00 a.m. on the day(s) of my departure(s)—but that is for another time. Quest Software Israel was located in Or Yehuda, one of Tel Aviv's suburbs. Apart from learning a little Hebrew and savoring Mediterranean cuisine (*chatzileem*, *baba ghanoush*, *tahini*, and more), I got to know two incredible guys really well: Eyal Kalderon and Shlomo Urbach. They were senior engineers on the team. I have met some really smart guys in my work life, but these guys were on a different level. They took me on a deep dive into the internals of some Unix system calls including (but not limited to) *wstop*. This system call allowed you to stop a system call, capture the parameters in the call, and let it proceed, all within a fraction of a microsecond. Cool, huh?

Geekcruises—Database Discovery (2001)

Hotsos conference speakers (2003). Standing: Anjo Kolk, Carol Dacko, Julian Dyke, Wolfgang Breitling, Cary Millsap, Jonathan Lewis, and Tom Kyte. Kneeling: Gaja Krishna Vaidyanatha, Kyle Hailey, Eric Grancher, Mogens Norgaard, and Stephan Haisley.

In the latter half of 2001 we brought to market the first-ever object-level I/O bottleneck detection tool: StorageXpert for Oracle. StorageXpert was supported on Oracle 7.3, 8.0, and 8i across all relevant Unix flavors (Solaris, HP-UX, Digital UNIX,

AIX, and Windows NT). As a software vendor in the early 2000s, you were required to support the various flavors of the operating systems (OS) that the server vendors supported. Porting software across multiple flavors of Unix had a plethora of issues, primarily due to the lack of a standard or a core OS code set in Unixland. Although IEEE released the Portable Operating System Interface (POSIX) standard in 1988, not all hardware vendors embraced it immediately, and the different flavors of Unix had varying degrees of POSIX compliance over the years.

StorageXpert was a revolutionary product as it visually exposed block-level hot spots within a table or index. The identification of hot tables and indexes allowed an administrator to take the necessary action from an optimization standpoint (e.g., tune the SQL, reduce block-level contention, and so on). Administrators were also able to perform relevant tier-based storage provisioning for the colder objects. When you moved colder objects out of Tier 1 storage, you created more capacity (both space and performance) for the primary hot objects (usually a smaller subset).

This was especially true for ERP applications that boasted tens of thousands of tables and even more indexes. Tier 1 storage was still very expensive in those days, and StorageXpert helped companies maximize and leverage the Tier 1 storage investment for much longer time periods. It was also a period of recession (after the internet bubble burst) and IT spending was cut across the board, making the value proposition unquestionable. The movement of objects from one tablespace to another (Tier 1 to Tier 2) was integrated with LiveReorg. I wish someone would lend me a time machine so that I could go back to those days. I truly was in techie heaven.

Floating Points with Oracle and Orcas

In June 2001, I had the pleasure of presenting at a very special conference. Let's just say that it gave new meaning to the term "floating point." The presentation room swayed from side to side during my talks and it depended on a variety of geo-climatic factors. I was aboard a Holland America cruise ship, *MS Veendam*, on a "geek cruise" titled Database Discovery 2001. The cruise took us through the Inside Passage from Vancouver, BC, to Seward, Alaska, making port calls at Ketchikan, Sitka, and Juneau. The cruise lasted seven days and it was a fantastic experience. I did three Oracle talks at the conference and—in exchange—got to see nature like never before. Alaska was stunningly beautiful. I was fortunate enough to learn more about the Native American culture at Sitka and interact with a pod of orcas during a shore excursion in Juneau. I also got to share a meal one evening with my dear friend Tim Gorman, who was radiating while wearing a Scottish kilt! I couldn't have asked for more out of a technical conference. Thank you, Neil Bauman, for enriching my life with this wonderful experience.

Back to the Mothership

I rejoined Oracle Corporation at Redwood Shores, Calif., in the System Management Products Group in June 2002. I was initially asked to build functionality in Oracle Enterprise Manager (OEM) that competed with StorageXpert. But the universe had different plans for me. As it turned out, Kyle Hailey, John Beresniewicz, James Morle, Graham Wood, and I got together to re-architect the performance pages of OEM (renamed as Oracle GridControl) in Oracle 10*g*. Graham was already working on Active Session History (ASH) and Automatic Database Diagnostic Monitor (ADDM), and our collaboration (between system management products and server technologies) was deemed perfect. The foundation of the performance screens in Oracle GridControl 10*g* revolved around three things: SQL, Waits, and Sessions. I called it the Triangle of Performance (ToP). We used Scalar Vector Graphics (SVG), an Adobe plugin for rendering the real-time graphs. And, with that, we changed Oracle's offering in the database performance tools market.

Flying Solo (2004–2016)

In March 2004 I decided to go on my own and thus launched a small consulting company: DBPerfMan LLC. DBPerfMan stands for *D*ata*B*ase *Perf*ormance *Man*agement, and we made our humble beginnings in my apartment in Sunnyvale, Calif. We primarily helped customers in the area of Oracle Systems Performance, in both the proactive and reactive contexts.

Performance Management in the Land of the Rising Sun

The first project fell into my lap, thanks to my dear friend Anjo Kolk (the author of the YAPP paper I mentioned in the Growth Years story). Anjo was friends with Ichiro Obata (Obata-san) of Insight Technologies in Chigasaki, Japan (a small town on the outskirts of Tokyo, in the Kanagawa Prefecture). Anjo and Obata-san had known each other for many years, since both had worked at Oracle. Insight was looking for someone with experience in the area of Oracle performance to evaluate and upgrade their Performance Insight tool. They wished to play a significant part in the Oracle performance tools market in Japan and wanted to make sure the product was ready for prime time. I had just launched DBPerfMan after a four-year stint in product management at Quest and Oracle. It was mutually agreed that our partnership was a good fit.

"When summoned to solve a performance problem without any background or context, I first try to get a good understanding of the workload that the database is processing. This includes getting a high-level one-hour workload profile via Oracle's Automatic Workload Repository (AWR) report and looking at where the database is spending most of its time: DB Time (CPU Time + Wait Time)."

Japan, an incredible country with deep roots in culture and tradition, provides anyone a kaleidoscope of experiences—mountainous national parks, temples, shrines, historic monuments, and imperial palaces. Mt. Fuji stands majestically above everything else, lending an incredible backdrop to this great

country. In addition, it is the land of bullet trains (*Shinkansen*), sushi, anime and Pokémon. The Japanese were also pioneers in providing mobility to music. Sony invented the Walkman and Discman many moons ago.

Doing business or working in Japan requires a totally different mindset. Apart from the obvious need for translation from English to Japanese and back (which makes for very long meetings), the work culture is mostly structured in a 100% consensus-based approach. At least that was my impression with the two projects, one with Tokyo Disneyland and other with Insight Technologies. If you were in a meeting, everyone in the room had to be convinced of a decision that was being made. Reading body language during meetings is a required skill. Work relationships are very hierarchical—respect is an integral part of everyday human interaction, and very few Japanese will openly disagree with anyone, let alone their superiors. As an external consultant, things were a tad different, but not by much. The universal head-nod always indicated to me that I was on the right track. As for disagreement, I quickly picked up an audio cue (a very long *aaanhh* before the actual sentence is spoken), which is a clear signal that what I've just said is not sitting well with the people in the room.

"When you start designing and drafting any architecture, the problems that you solve during the initial rounds should not keep resurfacing. That is the first sign that something is fundamentally wrong with the approach. For me, this moment of truth arrived when I realized that the problem I was trying to solve could not be solved in the realm of Oracle and relational databases."

The Japanese are hardworking people and have a great sense of pride in their work and their culture—and rightfully so. Companies like Hitachi, Mitsubishi, Nissan, Mizuho Financial, Softbank, Honda, Fujitsu, Toyota, and many others are household names all over the world. The term *kaizen* has deep roots in the Japanese work culture. It signifies change for the better, continuous improvement with small daily changes. It propagates the idea that the workforce will better absorb continuous small incremental changes than occasional big radical changes. Japan should be given credit for the wide adoption and use of kanban boards in software development. After all, they were invented by Toyota.

For all practical purposes we were on a performance tuning *kaizen*, making small yet significant changes to the product on a daily basis. The first few weeks were spent sorting out the Oracle performance data that was feeding the various screens of the product. Once we got the data synchronization issues sorted out, we then went down the path of minimalism.

Don't Make Me Think

The process of converting data to information is an art form. No two people see the world the same way, yet there is a need to ensure that only relevant information is rendered in an application. A well-designed application should automatically draw users to the important information. This goes with the construct of developing effective applications that have minimal (if any) distractions. If you are planning on building applications and care about usability and HCI, I'd highly recommend that you read *Don't Make Me Think*, by Steve Krug. It will transform your perspectives on visualization.

Forty-eight hours after a GUI team meeting in which we discussed car dashboards as a visualization metaphor, the lead GUI engineer came up with mockups of how to meaningfully display the Oracle performance data in a dashboard. A car's dashboard is minimalistic by nature. It does not show every agonizing detail of how a car functions. It displays the most important five aspects of the car. Yet when something malfunctions, the error is displayed in a context-sensitive message. By building a series of visually intuitive widgets, the GUI engineers triggered the rebirth of the product. That was the day Insight Technologies reinvented themselves and their Oracle performance product.

I worked with Insight Technologies through November of 2004. I loved my interactions with everyone at Insight and E-GlobalEdge. Special thanks are in order to Toyosuke Torimoto (Torimoto-san) for providing translation services for many weeks. This project gave me an opportunity to engage in product innovation while experiencing the beautiful history and culture of Japan. As the locals would say at the end of a hard day's work, "Otskare Samadesu! Domo arigato gosaimasu! Mata ashita ne!" (That was a great day's work! Thank you very much! See you tomorrow!)

A Weird Database Performance Problem

When summoned to solve a performance problem without any background or context (other than the fact that the application is slow), I usually begin my investigation from the database and work toward the application. I first try to get a good understanding of the workload that the database is processing. This includes getting a high-level one-hour workload profile via Oracle's *Automatic Workload Repository (AWR)* report and looking at where the database is spending most of its time: DB Time (CPU Time + Wait Time). Along with this, the relevant operating system metrics, CPU usage (*%usr*, *%sys*, *%wio*, and *%idle*), CPU run queue (the runnable queue—column "r" in a *vmstat* output), and memory usage (other relevant columns of *vmstat*) are also reviewed for the same time period. It is important to superimpose the database workload with the operating system metrics from the same time period in order to understand fully the interplay between Oracle and the operating system.

In 2005–2006, during one such performance tuning engagement, I found the database not doing much. A one-hour AWR report showed about five minutes of real work (DB Time), yet the claim was that the application was running at full speed. There was nothing inherently wrong with the SQL that was captured. However, it was observed that as the days went by in a given month, the application got progressively slower. That was the first clue that time played a crucial factor in the performance of the application. How does one reconcile such a huge disparity? Sixty minutes of clock time on a multi-core system versus a total of five minutes of database time. Something was not adding up!

The relevant next step was to trace one or more active application sessions that were connected to the Oracle database. Normally, tracing the SQL workload provides deep insight into how the application is interacting with the database. But clearly in that case an extended SQL trace (via the diagnostic event 10046) was not going to reveal anything different from the AWR report. Nevertheless, in an effort not to deviate from my own process, I traced one of the active application sessions. As expected, I did not find anything to write home about. It was consistent with the AWR report. Not much was happening in regard to work on the database.

At this stage I would have done a Java trace using JProfiler or an equivalent to find out more about the application code. The code was a third-party package, and I was given the option of doing a source code review with the vendor. After an overview discussing the application architecture, we began to walk through the code. The application in question was scheduling software, which provided three available timeslots when a customer called in for service. The elapsed time for determining these three available timeslots increased as the month progressed. At the beginning of the month, timeslots were generated quickly, but as time passed, the availability of timeslots was reduced and elapsed time for timeslot generation became an issue.

Complexity and Big O

When we finally got to the core of the scheduling functionality, I asked the lead engineer what data structure was used to store the available timeslots. He said, "We are using a linked list." Analysis of algorithms from graduate school flashed before me. I recalled the whole discussion about algorithmic complexity, its relevance on runtime performance, and how data structures played a critical part in it. If I'd been playing the lottery, I just hit the jackpot. What are the odds that I had just uncovered the root cause of the application's performance problem with one question?

O (known as Big O) denotes the algorithmic complexity of a given algorithm. An algorithmic complexity of O(n) is considered linear, which implies that the complexity (runtime) of the algorithm proportionately increases with n. In comparison, an algorithmic complexity of O(logn) is considered logarithmic, which implies that the increase in runtime is not proportionate to n. Said in another way, O(logn) demonstrates a disproportionate increase in runtime, with an increase in n, and that is exactly what you want when you write code. From an algorithm's standpoint, O(logn) is much better than O(n). An algorithm is considered scalable when it demonstrates a complexity of O(logn) on average.

A linked list has an algorithmic complexity of O(n) for access or searches. Here *n* is the number of nodes in the data structure. In comparison, a binary tree or a skip list has an algorithmic complexity of O(logn) for access or searches. Thus, it became clear that as time progressed in a given month, the search time it took to get an empty slot in the scheduling calendar increased linearly. This is due to the sequential access method of a linked list. The engineers who designed this may have chosen this data structure, which is optimized for data insertion. They were probably blindsided by the search time issue. An obvious side effect of increased search time is increased CPU usage. I will leave you with the following links, which will serve as additional study material if you are interested in this aspect of Computer Science:

1. Linked list: **en.wikipedia.org/wiki/Linked_list**
2. Skip list: **en.wikipedia.org/wiki/Skip_list**
3. Algorithmic complexity: **en.wikipedia.org/wiki/Time_complexity**
4. Big O cheat sheet: **bigocheatsheet.com/**

The Public Cloud, the Private Cloud, and Reinventing the Wheel

Over the years, hosting companies evolved from an application service provider (ASP) model in the dot-com era to a managed service provider (MSP) model when they took complete responsibility for the entire IT infrastructure, in addition to providing hosted database and application environments. These days you will find many an MSP rebranding their services as a cloud service provider (CSP). To some extent that might be true, but they would have to provide way more functionality than a classical MSP (hosting and managed support) to become a CSP. In addition to all of the usual hosting and managed support that an MSP provides, a CSP has to provide and support the following:

1. Self-service portal
2. Automation
3. Dynamic provisioning and teardown
4. Elasticity
5. Scalability
6. Chargeback and metering
7. Monitoring
8. Data consolidation
9. High availability and disaster recovery
10. Government-grade security

I was a spectator (as a consultant) at a few corporations where a quest to build a private cloud was in progress. To everyone's discomfort in every relevant meeting, I repeatedly asked the question, "Why they we doing this?" Data privacy and regulatory compliance were common reasons tendered, but more often than not there was no convincing answer. The thought process went on these lines, "If Amazon, Microsoft, or Google can do it, so can we." Oh dear, where do I start? Getting private clouds done right is a very difficult undertaking.

If data privacy and security is the primary argument and if

"Data integration is a significant and complex effort to bring data and data models together from multiple systems. Data integration allows organizations to achieve a more holistic view of their businesses by providing high-quality integrated data. It helps them understand their customers, products, and services better. If your data is not integrated, your business is flying blind".

"Evolution brings about change in our thought processes. Learning occurs when the mind is open to receive new ideas. Transformation is a frequent and continuous process that converts new ideas into pertinent information and actions. Just because I have done something in a certain way for many years, it does not necessarily warrant that I do it the same way now."

regulation mandates that client data should never leave the premises, then so be it. At the least, the majority of the data footprint (other than client data) can be stored on the cloud. How does a transaction table purely with numeric keys and numbers in any way jeopardize the privacy of a client? Are we seriously stating that reverse engineering of client_id and asset_id key values can be done on the fly without any lookup tables? That's a discussion for another day.

There is this false sense of security in some circles that anything within the firewall is secure and anything outside is not. However, there are reports of corporate data centers being compromised left and right (either breached or under a cyber attack). This should get us to re-think our opinions and biases about data in the public cloud. AWS, Azure, or Google Cloud provide very secure infrastructure and implementation practices for securing your systems and data. But if we don't do our part in securing our own data, we can't then turn around and blame the CSPs for it.

For example, if a company enables (inadvertently or otherwise) public access to an Amazon S3 data bucket on AWS with sensitive information on it, blaming AWS for a data breach is not reasonable. Data breaches don't happen on CSPs because of some mythical public cloud vulnerability or conspiracy theory. They occur because someone has carelessly left a window or door open, or taped the code to the alarm system in plain sight on the keypad. This is true even with recent events where a certain corporation's employees left a door ajar, only to quit the company and then break in—which could happen whether or not something is in the cloud or within the firewall. The true issue is how do you prevent such leaving-the-door-ajar events? The public cloud does not in any way obviate the need to secure the data exactly the way it was on-premises. If we did our part in data security, I wager that data stored in the public cloud (in, say, AWS) is more secure than in your own data center. This is because of the CSPs' continuous investment in security monitoring and hardening of the attack surfaces.

Some final thoughts on this subject: Let's stop reinventing the wheel when it comes to infrastructure and service provisioning. Let's work with the relevant governing bodies and regulators in our respective vertical markets and get the required clearances to get our data into the public cloud. Let's build the necessary controls and security mechanisms to ensure that systems and data are safe. If the prediction that 70% or more of all of the world's computing will be in the cloud by this year (2020) is valid, this computer systems engineering shift is more prevalent than the desktop revolution of the 1990s. If you are planning to truly innovate, disrupt your marketplace, and scale your business with computing at web scale, you need to get out of data centers. Public cloud vendors like AWS, Azure, and Google Cloud are definitely where your data and your systems belong.

Evolve, Learn, and Transform (2017)

"You can't handle the truth." Those were famous words by Col. Nathan R. Jessep, played by Jack Nicholson in the 1992 blockbuster *A Few Good Men*. Evolution brings about change in our thought processes. Learning occurs when the mind is open to receive new ideas.

Transformation is a frequent and continuous process that converts new ideas into pertinent information and actions. This was and still is my ELT, and it has revealed this truth to me: Just because I have done something in a certain way for many years, it does not necessarily warrant that I do it the same way now.

The Truth Arrives

"The law of the instrument" was proposed by Abraham Kaplan in 1964. He said, "Give a small boy a hammer, and he will find that everything he encounters needs pounding." We have rephrased this: "If you have a hammer, every problem looks like a nail." In my humble opinion, this cognitive bias prevents us from reaching our fullest potential in problem solving. In early 2017, I almost became the poster child for the law of the instrument.

As mentioned before, I had worked with relational databases (specifically Oracle) for many years. I had the mindset that I could figure out a way to do almost anything with Oracle. And for the most part that was true. When you start designing and drafting any architecture, the problems that you solve during the initial rounds should not keep resurfacing. That is the first sign that something is fundamentally wrong with the approach. For me, this moment of truth arrived when I realized that the problem I was trying to solve could not be solved in the realm of Oracle and relational databases.

Integrate or Go Out of Business

Data integration is a significant and complex effort to bring data and data models together from multiple systems. Data integration allows organizations to achieve a more holistic view of their businesses by providing high-quality integrated data. It helps them understand their customers, products, and services better. With this renewed understanding, organizations are able to provide better products and services, reduce costs, increase sales, expand market-share, and increase revenue. Said in another way, data integration fuels growth—and it also mitigates risk and regulatory issues that may arise due to bad data. Data integration is a fundamental right of every organization. If your data is not integrated, your business is flying blind.

Harmonize and Canonicalize

The process of harmonization and canonicalization is used to

bring different versions of the data to a standardized single source of the truth. Let's set the stage by giving you a very specific example: Take, for instance, the case of the CUSTOMER table. Let's keep this very simple with one column, the CUSTOMER_ID, the primary key (PK) of CUSTOMER. Let's also assume that CUSTOMER_ID is our standard name for this PK. When you have dozens of systems and they all have their own version of the CUSTOMER table (due to a lack of consistent data management practices of the past), the PK will have different names.

This poses a huge problem for the target data integration repository. You do not have the option of changing the name of the CUSTOMER table's PK in existing systems. Yet we are faced with the task of equalizing CUST_ID, CUSTOMERID, CID, C_ID, and 42 other versions to the standard CUSTOMER_ID. How do you go about this? Let me gently remind you that we are discussing a single column in a single table. There are hundreds of tables, each having dozens of columns, and this blows out the scope and complexity of the problem. I have not even touched on the 42 different versions of a given customer's record (yes, the data). That is another issue that needs to be dealt with. We'll put it on the back burner for now.

The act of enforcing naming standards needs to be done not in the current systems but in the data integration repository. It is unrealistic to expect existing systems and the relevant applications to change to the standard CUSTOMER_ID. That would be a disruptive and expensive approach to standardization, and nobody is going to sign up for that. The process of enforcing standards is called "canonicalization." Data model and data harmonization are the Holy Grail for building any data integration repository. They make a single version from many. Combined, the two functions convert the 42 different versions of the CUSTOMER's primary key, to look and feel as one, CUSTOMER_ID. Canonicalization is a continuous and frequent process, and is an integral part of the evolution of data quality. However, without canonicalization the value proposition of the unified data integration repository's target state significantly diminishes. In the end, data integration fails without canonicalization.

So how do enterprises achieve this in the relational world? Well, they start by first creating an Enterprise Data Model (EDM) at the conceptual level, followed by creating it at the logical level, and then finally at the physical model. When the EDMs become available, they need to be continually kept up to date, as it needs to change along with the business. Also, every system from the past and in the future would be required to conform to the physical EDM. This ensures the use of standardized names for the relevant entities and attributes (eventually resulting in standardized table and column names). The problem of different versions of data in different systems is still not addressed by data modeling. However, the problem of 42 different versions of data can be addressed in the relational world with large amounts of custom code that can be run at the Extract Transform Load (ETL) layer during ingestion time. All of this requires a significant amount of policing. This is expensive, time consuming, and difficult to sustain in the long run.

Undertaking the effort of everything mentioned in the previous paragraph (after the fact) is monumentally complex. Sadly, tragedy strikes right after the EDMs are ready. They become obsolete right out of the starting block. Why so? In the many months that it took to develop the three EDMs, the business undergoes a significant transformation, requiring additional effort to update the EDMs—and this cycle repeats all over again.

For a business that requires a holistic 360-degree view of every facet, waiting for many months to get a data integration repository ready and then figuring out that it needs more work is not acceptable. This changes the rules of the game. We are faced with a situation where data and data model harmonization needs to be achieved without the three different EDMs. Enter the database that saves the day: NoSQL.

ACID, BASE, and the Database

Even as I made my switch to NoSQL, there were certain things that I was unwilling to compromise on. In the world of transactional databases, maintaining ACID (Atomicity, Consistency, Isolation, Durability) properties is something we all take for granted. It is beyond the scope of this article to discuss in detail what ACID is all about. However, I will cover ACID's viable alternative, BASE (Basically Available, Soft-state, Eventual Consistency). Although it is normal for NoSQL vendors to relax the consistency property and adopt the eventual consistency aspect of BASE, the implementation of this aspect needs to be done correctly. Let's take a closer look at this.

"It is powerful to pick and choose columns from different data sources and harmonize data—and the idea of provisioning a golden merged version of the customer record from above is pretty awesome. We have done all of this without an EDM. You are free to change any of it at any time. It is truly evolutionary and clearly supports the ever-changing business environment of today."

First of all, ACID compliance should not have disclaimers attached to it: single document ACID compliance, single row ACID compliance, or single object ACID compliance. ACID compliance should never be limited to a single row or object or document. It should be across multiple rows or objects or documents. And please, let's not call something ACID-compliant and then turn around and support dirty reads!

Dan Pritchett, the author who published BASE, is a former eBay Technical Fellow. In his BASE paper (**dl.acm.org/citation.cfm?id=1394128**), he refers to Eric Brewer's CAP theorem and why building purely ACID-compliant applications can have an impact on scalability. The CAP theorem states that you can have only up to two of three characteristics of CAP (Consistency, Availability, and Partition Tolerance) to maintain scalability. Thus was born the idea of BASE.

First of all, BASE is not purely relevant to NoSQL databases. It is implementable even in a relational ACID environment. BASE was originally proposed with a relational backdrop, and it brings forth the idea of designing applications with decoupled

"If the data model cannot change with time, it becomes obsolete—and so does the system that is built upon it. In the end, the system does not match the business requirements and problems arise. In the case of NoSQL databases, it even supports storing a different number of columns/elements for each record/document. That is also unheard of in the realm of relational databases."

transaction management. This is achieved by primarily utilizing asynchronous queues and achieving eventual consistency. Decoupling is definitely a good thing for scalability—when it's done right. Just to be clear, Mr. Pritchett never postulated that ACID-compliant applications would never scale. He urged the reader to understand the limitations of coupling in an application's transaction architecture. Coupling can cause scalability bottlenecks; thus, BASE proposes decoupling to eliminate these bottlenecks.

The rudiments of ACID ensure us that when a transaction commits, there is no question whether the data shows up in the database. A COMMIT's guarantee should never be in question. On those lines, it is required to ensure that in a distributed database configuration, the primary database and secondary databases are eventually in sync. If we choose to utilize the primary for transactions and the secondary for reporting, database replication (logical or physical) has to do its part in keeping the secondary up to date. The secondary should be eventually consistent with the primary within a finite period of time. We should not have to worry about write concerns or read preferences to make a distributed database environment functional. Both the primary and the secondary databases should work out of the box. I urge you to do your homework with regard to reliable COMMITs in NoSQL databases and make an informed decision, lest you regret your choice later.

To recap: I needed to design and create a data integration repository to provide a holistic 360-degree view of the business. There were dozens of existing systems with no standard naming conventions that sourced the data. This required data and data model harmonization, due to the absence of a physical EDM. Time was not on my side, as I did not have many months to invest in creating an EDM. I needed the NoSQL database (non-cloud) to be reliable, consistent, scalable, secure (element-level security), and rock-solid, and to support ACID properties and be enterprise class. The NoSQL database chosen was MarkLogic.

Note: You may wonder why I chose to work in a non-cloud version of NoSQL. Let's just say, for simplicity, that at the time I was not given much of a choice.

The Richness of MarkLogic—Indexing, Search Engine, and More

MarkLogic was an incredibly powerful NoSQL database that boasted some great features. The Universal Indexing policy ensures that pretty much everything (other than binary files) is indexed upon loading. This facilitated quick access to the data upon loading. Unlike the others, you did not have to provide a path to create an index. Everything got indexed automatically upon loading. The value proposition was apparent right after the data was loaded. This database came with a built-in search engine out of the box and was 100% integrated. You did not have to attach a third-party search engine to this NoSQL database. You could, quickly, launch a web application that talked to the database, putting the data to use. Simultaneously, you could work on the data quality in an iterative manner.

As a relational technologist, I have spent many hours lining up data and ensuring that everything is perfect before applications could utilize it. With NoSQL databases, even with partial canonicalization, data presents itself with great value. For example, here is a sample query right off the loading process: "Give me all occurrences of the string 'Acme' regardless of which element/column it occurs in." If you have tried anything remotely resembling this in SQL across your entire database, you will appreciate how difficult this is to pull off in the world of relational databases. Yet queries like these are child's play for NoSQL databases.

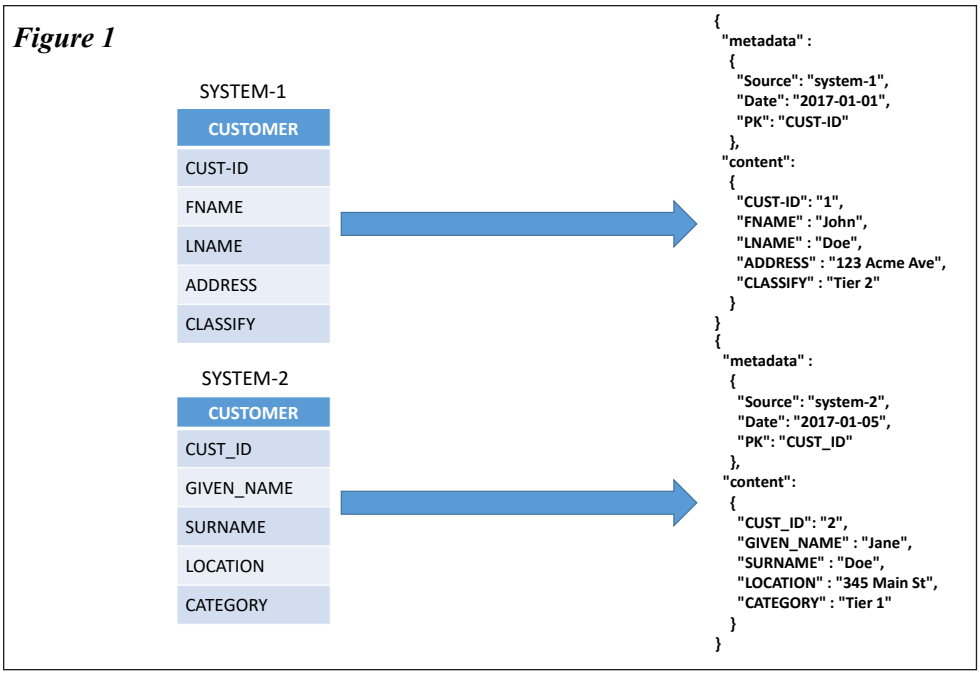
Data Integration at Work

The need for data and data model harmonization brought me to NoSQL. In my mind this is fundamental to data integration. In this final chapter of my journey to NoSQL I will dive a bit deeper into this subject. In a previous section I introduced you to harmonization with a simple CUSTOMER table. Now I want to show, by example, the power of canonicalization, which allows us to achieve both data and data model harmonization.

NoSQL databases support different persistence formats, "document" being one of the more common formats. In keeping brevity and a smaller persistence footprint in mind, using JSON as the document type is an easy choice for relational data. Harmonization is not a one-shot effort. It is iterative by nature, just as any transformation effort should be. The quality and content of data improves over time. This brings to focus a novel approach to data management—ELT.

ELT (yes, the other one) stands for Extract, Load, and Transform. When compared to ETL (the classical approach to data

"I did not even think once during my Oracle career that I would open myself up to another relational database, let alone a NoSQL database. The land of NoSQL promises some very exciting times in my data management journey. A part of me will always relational. As for the rest, time will tell."



management), ELT is much more powerful. With ELT, you do not leave any data on the table. You load the data (as is) and then transform it, post-load, in place—multiple times: ELT(n). This transformation process is usually done in phases—first for data model harmonization, then data harmonization, and then with data enrichment (i.e., adding geographical and spatial characteristics). ETL is a one-shot approach, and what is not loaded, due to the rules at the time of loading, does not make it to the database.

The following steps explain the process of data harmonization and canonicalization.

Step #1—Relational to JSON conversion and metadata incorporation

In Step #1, we bring CUSTOMER data from two systems. The data is brought into the database as is. Please note that data from the first system has CUST-ID as the primary key (PK), and the second system has CUST_ID as the PK. We have made the decision that CUSTOMER_ID is going to be our standard. We will deal with the other columns later. The data is ingested from a CSV file, and the rows are converted into individual JSON documents. We have also added metadata for each document, and this can be done in a variety of ways (details are beyond the scope of this article). We have pictorially represented the relational to JSON representation in Figure 1.

Step #2—Canonical identifies CUSTOMER_ID as the standard

Welcome to canonicalization. In Step #2, the canonical directive is introduced and this identifies CUSTOMER_ID as the standard column. Figure 2 visually describes how this is done.

Notice that in this simple example, we have brought two customer records into the database, one from SYSTEM-1 for CUSTOMER_ID = 1 and the other from SYSTEM-2 CUST_ID = 2. The column names are all over the place. We are just standardizing on CUSTOMER_ID in this iteration.

Note: Canonicalization is shown in red and is actually achieved using a simple program after the data is loaded.

Step #3—Additional source with differing column names

In Step #3, we will bring more data, this time from SYSTEM-3, and in this case all data columns will be harmonized for all sources in an iterative fashion. The ingestion of data from SYSTEM-3 is illustrated in Figure 3. We have chosen to canonicalize the remaining columns.

Observe that the canonical directive is applied for the PK CUSTOMER_ID and beyond. We have chosen our standard for the remaining columns: LAST_NAME, FIRST_NAME, ADDRESS, and CLASSIFICATION. Notice how the column names in SYSTEM-3 are different from SYSTEM-1 and SYSTEM-2.

Without canonicalization, these naming differences of the data elements will make it very difficult to interact with the data integration repository. Here we have truly integrated three different data models from three different source

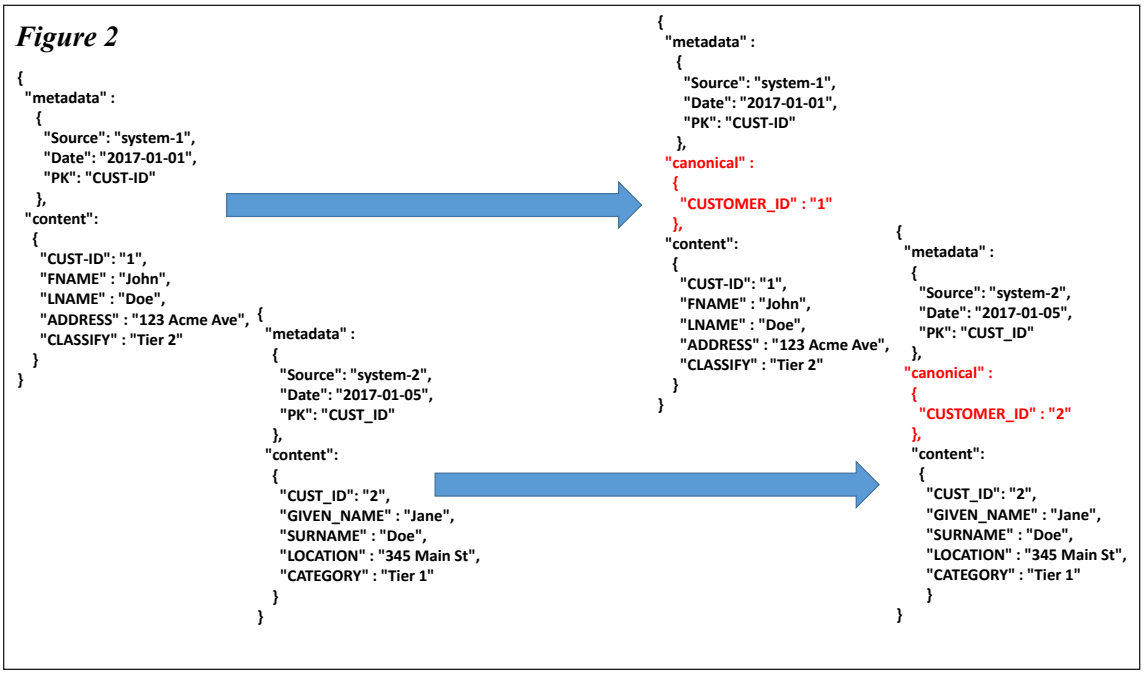
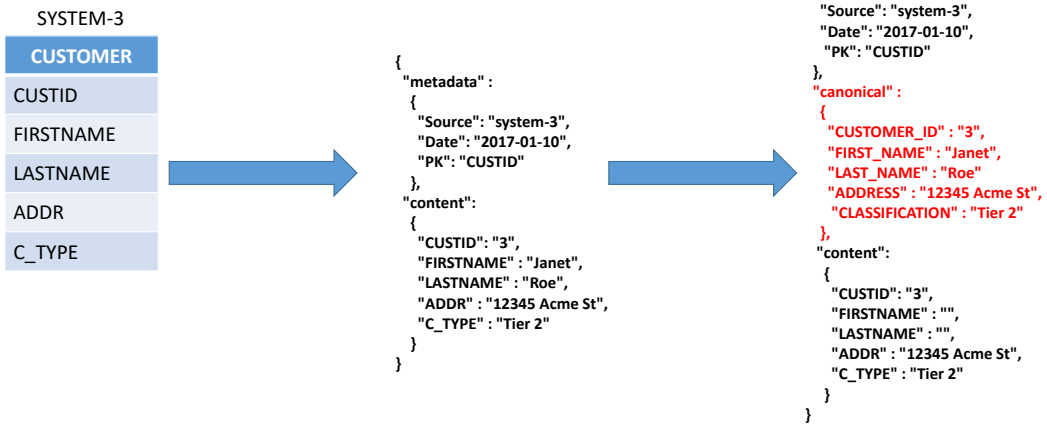


Figure 3



systems. This data model integration lays the foundation for the bigger data harmonization and the program at large.

Step #4—One more source with differing column names and data

In Step #4, we are going to complicate life a little by bringing in another copy of CUSTOMER_ID = 3, but this time from SYSTEM-4. Why are we intentionally bringing in a duplicate record? The answer is data quality. Our own internal data quality analysis has determined the following:

The version of CUSTOMER#3 from SYSTEM-3 has good data for FIRST_NAME and LAST_NAME. Notice that the ADDRESS of Jane Roe is listed as 12345 Acme Street. It needs to be Acme Ave. The ADDRESS and CLASSIFICATION data is incorrect.

It has also been determined that ADDRESS and CLASSIFICATION is good from SYSTEM-4. Notice the name is all in upper case and Janet's last name is misspelled as "DOE" instead of "Roe". The ADDRESS and CLASSIFICATION in SYSTEM-4 is deemed more reliable.

This provides you the rationale as to why data harmonization needs to be undertaken. There is no point creating a data integration repository where data quality is questionable. A pictorial representation of data ingestion from SYSTEM-4 is presented in Figure 4.

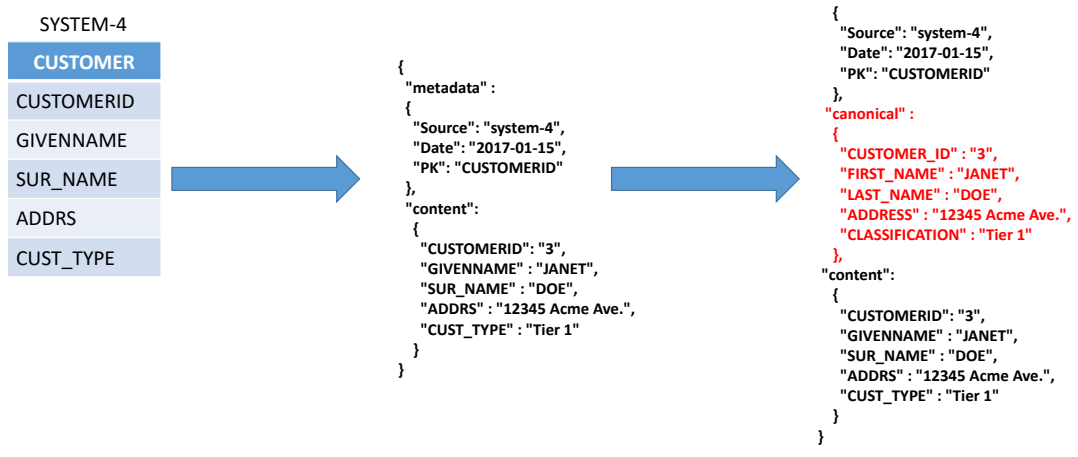
The NoSQL database will retain both original records (documents) from SYSTEM-3 and SYSTEM-4. This allows us to trace the lineage of the data. We will create a new standardized version for CUSTOMER#3 in the next step. So, in theory, we will have three versions of CUSTOMER_ID = 3, with the merged version as the "golden" record.

Step #5—Create merged record using two sources

Now we are taking data integration to a new level by performing data harmonization. The new merged record (document) is the golden copy, and it contains the good data elements from two previous records (documents) from SYSTEM-3 and SYSTEM-4. This is illustrated in Figure 5. It is powerful to pick and choose columns from different data sources and harmonize data—and the idea of provisioning a golden merged version of the customer record from above is pretty awesome. We have done all of this without an EDM. Now, that's impressive!

To be clear, nothing here is cast in concrete. You are free to change any of it at any time. It is truly evolutionary and clearly supports the ever-changing business environment of today. Data models in today's world need to be fluid. It is difficult to support today's businesses with rigid models. If the data model cannot change with time, it becomes obsolete—and so does the system that is built upon it. In the end, the system does not match the business requirements and problems arise. In the case of NoSQL databases, it even supports storing a different number of columns/elements for each record/document. That is also unheard of in the realm of relational databases, as the closest you get to this is a sparse matrix. Figure 5 represents the design of the merged record.

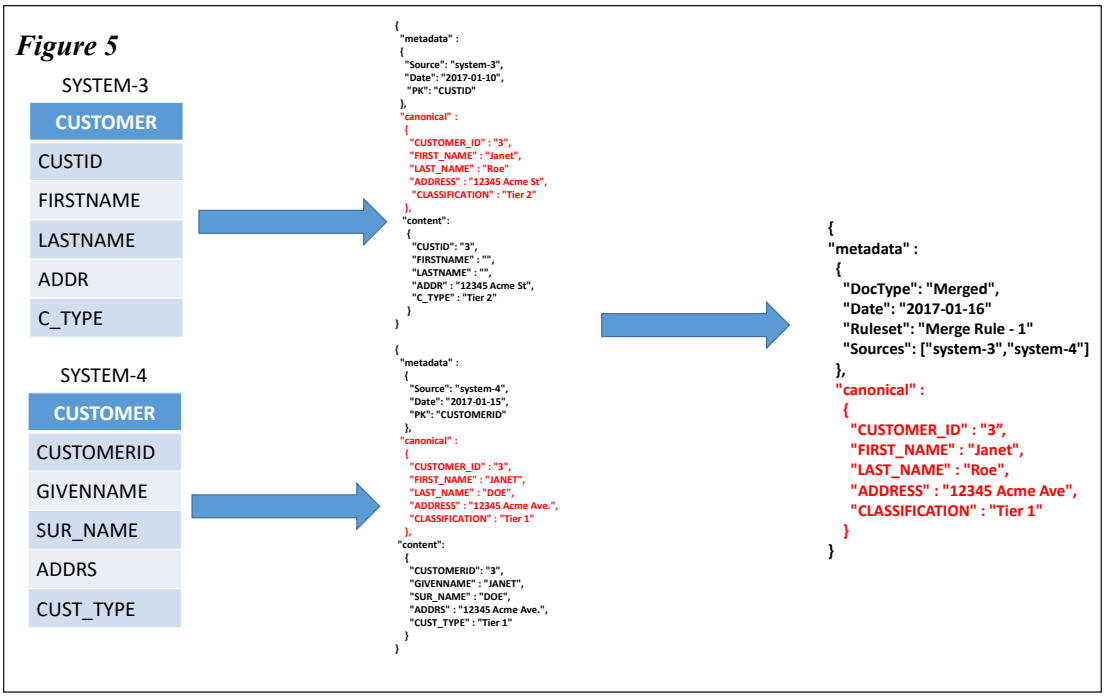
Figure 4



With the aforementioned 5-step process, our data and the data model are both harmonized. This is integral to the quality of what is being stored in the database. Now you can trust the business insights coming out of your analytics initiative.

One Final Confession

I need to fess up that I tried to implement rela-



tional canonicalization at the record level by creating the necessary metadata columns at the table level and then populating it after the load. But this approach did not pan out well. In many cases I ended up with a cumbersome metadata column because the canonical directive—which included all columns—needed to be stored as text within a single column.

With a single metadata column, a significant parsing overhead of the text (post load) comes in play. Think about it: If you have 100 columns in your table and if you are going to have a canonical directive for each column, you are going to end up with a very long string. When element-level operations were attempted, things got messy very quickly. Yes, I can perform JSON-like operations in Oracle 12*c*, but I will still have to store the content in a VARCHAR or CLOB—and then create virtual columns and virtual indexes on top of it. That is analogous to placing a square object inside a larger circle and then desperately trying to justify its congruence. In this case it makes no sense to store relational data in string format in a VARCHAR or CLOB column and then turn around and attempt to JSON-ize it.

When the canonical directive was broken down into individual columns, it got very unwieldy (100 metadata columns for 100 data columns is just crazy). If I created a pivoted metadata table and took the row-based approach, I ended up with 100 million rows of metadata for a table with 100 columns and 1 million rows of data. Any way I looked at it, it was messy in the relational realm. This approach also increased the risk of bad data and inconsistency. Thus, based on the above, it is safe to state that we should refrain from canonicalization attempts using relational databases! JSON is natively supported in most NoSQL databases via the document persistence model and thus makes harmonization, canonicalization, and integration much easier. It gets the job done.

Epilogue

I owe a great deal of gratitude to Jason Hunter for the many hours of discussions, white-boarding sessions, and technical content review of this article. Also, thanks to Frank Pacione for providing me a relational-to-JSON strawman, which I could build upon for my learning.

As I look back to my humble professional beginnings at Owens-Corning in 1992, I can't help but wonder how different things would have been if Mark Amos had not been my manager. What would have become of me if he had not guided me into the relational world? I did not even think once during my Oracle career that I would open myself up to another relational database, let alone a NoSQL database. I am very thankful for having been a student of the Oracle RDBMS all these years. It has enriched me in more ways than meet the eye. It has provided me a livelihood and more.

The land of NoSQL promises some very exciting times in my data management journey. A part of me will always remain relational. As for the rest, time will tell. It has been my pleasure writing about my first 25 years. I thank you for joining me on my journey. I wish you and your family the very best that life has to offer. Like the Jedi Master Obi-Wan Kenobi has always said: "May the Force be with you!"

Until we meet again . . . ▲

Gaja Krishna Vaidyanatha is a seasoned data practitioner with a 27-year proven track record of managing and integrating large data footprints, on-premises and in the cloud. During his career he has worked with a variety of data stores—relational (Oracle and MySQL), Hadoop, NoSQL (HBase, MarkLogic, DynamoDB, and FireStore), and columnar databases (BiqQuery). Gaja is an AWS Certified Cloud Practitioner and AWS Certified Solutions Architect Associate. He has co-authored two books—Oracle Performance Tuning 101, published by Oracle Press, and Oracle Insights: Tales of the Oak Table, published by Apress. His publications on Serverless Data Integration can be found at **www.cloud-data.biz/publications**. *He is a member of the OakTable Network and the inventor of the term Compulsive Tuning Disorder (CTD), circa 2000.*

Originally published as a series of articles on LinkedIn in 2017

Preventing the Post-Production Performance Problem

How confident are you that those new features you're adding to your production application will be fast and efficient? What if they're not? You need a process that finds inefficient code earlier, and a process to fix the inefficiencies that evade early detection. One process can accomplish both goals.

by Cary Millsap

Oracle tracing is an economical but rich performance data source.

Good tools make huge volumes of trace data manageable.

If you're building your own application, you can make tracing an easy-to-use feature.

Tracing reduces chaos and simplifies performance optimization into a single process throughout your software life cycle.

Using trace data in code reviews leads to more efficient applications, and it bonds DBAs and developers into better partners.

Trace data can help you detect performance regressions before your users notice them.

Trace data helps everyone in your organization make better-informed business decisions.

Problem

Many Oracle adopters have no process in place for detecting performance problems before the application go-live milestone. The results can be ugly. Once I saw an application that was so inefficient it couldn't support its initial 20-user training rollout. This thing was supposed to serve thousands of users. It cost $30 million and had taken five years to write.

Companies that don't have a process for preventing performance problems have to figure out a process for solving them later. Do you really want to wager your career on expensive-sounding ideas derived from troubleshooting methods and tools you've created on the fly, through trial and error, under the crushing, unrelenting pressure of your users and leadership scrutinizing every step you take?

You know that solving problems earlier in the software life cycle is better for everybody, but how?

Plan

You need a method for troubleshooting performance problems that works both before and after go-live, for *any* performance problem. You need a method you can practice and get good at, that everyone on your team—your DBAs, your developers, your sysadmins, your architects, even your users and leadership—can learn.

Next, you need for your application to integrate with your method. You want it to be easy to collect the performance data you need, throughout your software life cycle: from testing your earliest prototypes, to evaluating the efficiency of new features, to baselining your well-behaved applications, to diagnosing your misbehaving applications.

Finally, you need software tools that help you manage, mine, and manipulate the performance data your application collects. You need tools that help everyone on your team understand how your programs spend your time.

Analysis

The method and tools for accomplishing these goals do exist. We learned in the early 2000s that the most valuable performance data for Oracle Database applications is *extended SQL trace* data. Tracing is a basic feature of every release of every Oracle edition. Used correctly, tracing is a high-detail data source that creates virtually no perceptible sacrifice in system performance.

Tracing is infeasible when you don't have good tools to manage, mine, and manipulate the huge volume of detail that it gives you. Our Method R Workbench software is the interface between your mountain of data and the job to be done.

The method, itself called Method R, keeps you focused on measuring the *response times* of the programs that are important to your business. With Method R, your process for *preventing* problems before go-live becomes identical to your process for *diagnosing* problems in production.

Solution

Accomplishing your goals begins with a few process changes:

1. Application designers and developers make your application easy to trace. This way, everyone on your team can have easy access to your programs' detailed performance data. If you've bought (not built) your application, it may not have all the performance measuring features you'd like built in, but any Oracle-based application can be traced.
2. If you have in-house programmers, they trace routinely during development to find inefficiencies and understand how their programs spend time.
3. Before any new application feature is promoted to production, a database performance specialist reviews the new code's trace data. If your programmers are in-house, the trace data for their code is analyzed at every code review.
4. Database operators routinely trace key application features in production. Routine tracing reveals tiny performance regressions before your users notice them, and performance baselines make it easier to diagnose problems later.
5. When you do encounter a production performance problem, use trace data to find where your programs spend their time. Even if you have to trace every program on your system for a few hours, Oracle and Method R Workbench can handle the volume.
6. Everyone—both technical and non—participates in the culture of *measuring over guessing*. For example, before your next upgrade, *measure* the response times of your key application features. *Predict* how the upgrade should change the response time for each feature. After the upgrade, *measure* response times again and *calibrate* your forecast. If your forecast was wrong, learn why. It's your fast-path to performance expertise.

Results

Every code review that finds an inefficiency is a problem that no user will ever experience. It's a programmer learning how to write better code, and it's an organization that is more intimate with the performance of the software it counts on for survival.

The rare problem that does slip through your code review filter, you'll deal with using the same method and data that you use in your code reviews. At first, you'll think, "I can't believe I didn't notice that; I'll never make *that* mistake again." And then you won't. Over time your performance (both your application's and yours personally) will get more and more bulletproof.

On your production system, your database operators will trace the performance of your application's key features even on good days when users aren't having problems, because you want to see any little jiggle of creeping performance problems. Keeping those trace files will make it easier to diagnose problems later.

Your company will spend less on hardware upgrades, because more efficient software runs faster on less hardware. When you do upgrade, you'll predict how much faster your different programs will be. "TPS reports will run in 72% less time. Pick-to-Ship will improve by only 8%; but it's already subsecond, so it doesn't matter." And you'll be right.

Your whole organization will achieve a continual intimacy with the performance of your application. You know where your performance risks are, and you know what it looks like when your application is creeping toward problematic behavior. You'll still find the occasional surprise, but most of your surprises will be in pre-production tests, and for the rare surprise that occurs in production, you'll have your method and your data ready.

Technology

Method R Workbench is easy-to-use, high-precision *Oracle time measurement software* for software development, code reviews, performance tests, concept proofs, hardware and software evaluations, upgrades, troubleshooting, and more—for Oracle developers, DBAs, and decision-makers in every phase of the software life cycle.

◇ Method R™
method-r.com
info@method-r.com

Second International NoCOUG SQL Challenge

Reprinted from the November 2011 issue

The Second International NoCOUG SQL Challenge was published on February 13, 2011, in the February 2011 issue of the *NoCOUG Journal* (**http://bit.ly/gVNZsW**). SQL commands to create the data were provided at **http://bit.ly/g58WVn**. The challenge was to find the secret message hidden in a seemingly random collection of words. The winners are Andre Araujo (Australia), Rob van Wijk (Netherlands), and Ilya Chuhnakov (Russia.) Each winner will receive an Amazon Kindle from contest sponsor Pythian and the August Order of the Wooden Pretzel, in keeping with the pronouncement of Steven Feuerstein that *"some people can perform seeming miracles with straight Es-Cue-El, but the statements end up looking like pretzels created by somebody who is experimenting with hallucinogens."*

The first reaction to the challenge was one of puzzlement. Van Wijk wrote on his blog on February 14, 2011: "*Unfortunately, I don't understand what needs to be done. Is it forming a sentence? Three sentences? Do all words need to be used? If so, lots of sentences can be made; how do I know which is the right one? I'm afraid I don't think it is a *SQL* Challenge, but I may be missing something.*" However, the puzzle quickly fell to the combined onslaught of the international database community. At 6:11 a.m. PST on February 15, 2011, we received a solution from Araujo. He had realized that the words formed a binary tree and used "recursive common table expressions" to decode the secret message (the winning solution to a contest conducted by columnist Marilyn vos Savant in which contestants had to write a sensible paragraph of one hundred unique words).

"TRYING TO TYPE ONE HUNDRED DISTINCT WORDS IN A SINGLE PARAGRAPH IS REALLY TOUGH IF I CANNOT REPEAT ANY OF THEM THEN PROBABLY THOSE WITH MANY LETTERS SHOULD BE USED MAYBE SOME READERS WILL UTILIZE DICTIONARIES THESAURUSES THESAURI OR POSSIBLY EVEN ENCYCLOPEDIAS BUT MY PREFERENCE HAS ALWAYS BEEN THAT GRAY MATTER BETWEEN YOUR EARS SERIOUSLY MARILYN CHALLENGES SUCH AS THIS REQUIRE SKILLS BEYOND MATH SCIENCE AND PHYSICS SO WHAT DO YOU ASK READING COMPREHENSION WRITING ABILITY GOOD OLD FASHIONED ELBOW GREASE SCIENTISTS DON'T CARE ABOUT STRUCTURE THEY WANT RESULTS HEY LOOK ONLY ELEVEN MORE LEFT FIVE FOUR THREE TWO DONE"

Araujo posted a detailed analysis of the problem at **http://www.pythian.com/news/20757/nocoug-sql-challenge-entry-2/**. He admitted that—even though he had successfully decoded the secret message—his solution would not work for all binary trees. At 1:08 p.m. PST the same day, van Wijk sent us a recursive CTE solution that works for all binary trees. Here is the solution with some modifications for extra clarity.

```
-- Assign an ordering string to each node
WITH CTE(word1, word2, word3, ordering) AS
(
  -- This is the anchor member of the recursive CTE
  -- Identify the root of the binary tree
  SELECT
    r.word1, r.word2, r.word3,
    -- The ordering string for the root node is '1'
    cast('1' AS VARCHAR2(4000) ) AS ordering
  FROM riddle r
  WHERE NOT EXISTS (
    SELECT * FROM riddle r2
    WHERE r.word2 IN (r2.word1, r2.word3) )

  UNION ALL

  -- This is the recursive member of the recursive CTE
  -- Identify the left and right nodes if any
  SELECT
    r.word1, r.word2, r.word3,
    -- Compute the ordering string for this node
    CASE
      -- Handle the case of a left node
      WHEN r.word2 = CTE.word1
      -- Change the last digit to '0' and then append '1'
      THEN replace(CTE.ordering, '1', '0') || '1'
      -- Handle the case of a right node
      WHEN r.word2 = CTE.word3
      -- Change the last digit to '2' and then append '1'
      THEN replace(CTE.ordering, '1', '2') || '1'
    END AS ordering
  FROM riddle r JOIN CTE
  ON r.word2 IN (CTE.word1, CTE.word3)
)
-- Sort the words using the ordering string
SELECT word2 FROM CTE ORDER BY ordering;
```

A recursive CTE consists of an "anchor" member and one or more "recursive" members. The anchor member generates seed data, while the recursive member generates additional data. Any additional data is fed right back to the recursive member, and the process continues until no more data is found. To help understand van Wijk's solution, store the words of the sentence *"Quick brown Fox jumps over the lazy dog"* in a binary tree as shown in Figure 1.

```
CREATE TABLE riddle
(
  word1 VARCHAR2(32),
  word2 VARCHAR2(32) NOT NULL,
  word3 VARCHAR2(32)
);
INSERT INTO RIDDLE  VALUES (NULL, 'Quick', NULL);
INSERT INTO RIDDLE  VALUES ('Quick', 'brown', NULL);
INSERT INTO RIDDLE  VALUES ('brown', 'Fox', 'dog' );
```

(continued on page 25)

Second International

NoCOUG SQL Challenge

BE IT KNOWN BY THESE PRESENTS that the great Wizard of Odds at Hogwash School of Es-Cue-El needs your help in solving the riddle of the ancient manuscript found in the secret chamber of mystery. A great tournament has been organized, and all practitioners of the ancient arts of Es-Cue-El have been invited to demonstrate their prowess.

Unsolvable Riddle

An ancient manuscript titled "Love Your Data" has been discovered in the secret chamber of mystery at Hogwash School of Es-Cue-El. The manuscript was covered with mysterious words such as those shown below; the complete list has been published in the *Wizarding Journal* of the great Oracles of Northern California. The great Wizard of Odds implores you to create an Es-Cue-El spell that reveals the secret message.

	A	
COMPREHENSION	ABILITY	OLD
	ABOUT	
	ALWAYS	
SCIENCE	AND	PHYSICS
	ANY	
	AS	
SO	ASK	ABILITY

Big Prizes

The *August Order of the Wooden Pretzel* will be conferred on the winner, in keeping with the celebrated pronouncement of another great wizard that *"some people can perform seeming miracles with straight Es-Cue-El, but the statements end up looking like pretzels created by somebody who is experimenting with hallucinogens."* As if that singular honor were not enough, a marvelous collection of Oracular tomes will be bestowed upon the champion. May the best wizard win!

RULES: The winner will receive a Kindle and his or her choice of six books from the Apress book catalog. Due to shipping costs and limitations for certain parts of the world, electronic copies may be substituted. Prizes may be awarded to runners-up at the discretion of the organizers. Submissions should be emailed to **SQLchallenge@nocoug.org**. Contestants may use any database technology at their disposal, but the submitted solutions should be compatible with at least one of the following database technologies: Oracle 11*g* for Windows, SQL Server 2008, DB2 9.5 for Windows, and MySQL 5.1 for Windows. The competition will be judged by Jonathan Gennick, Assistant Editorial Director at Apress, and Iggy Fernandez, author of *Beginning Oracle Database 11g Administration* published by Apress. Judging criteria include correctness, originality, efficiency, portability, and readability. The judges' decisions are final. The competition will close at a time determined by the organizers. The judges and organizers reserve the right to publish and comment on any of the submissions with due credit to the originators. More information about the problem and additional rules can be found at **http://www.nocoug.org**.

Love Your Data

	A	
COMPREHENSION	ABILITY	OLD
	ABOUT	
	ALWAYS	
SCIENCE	AND	PHYSICS
	ANY	
	AS	
SO	ASK	ABILITY
	BE	
POSSIBLY	BEEN	MARILYN
GRAY	BETWEEN	EARS
	BEYOND	
	BUT	
	CANNOT	
SCIENTISTS	CARE	STRUCTURE
	CHALLENGES	
READING	COMPREHENSION	WRITING
WILL	DICTIONARIES	THESAURI
HUNDRED	DISTINCT	WORDS
WHAT	DO	YOU
THREE	DONE	
	DONT	
YOUR	EARS	SERIOUSLY
ASK	ELBOW	WANT
	ELEVEN	
EVEN	ENCYCLOPEDIAS	BUT
	EVEN	
	FASHIONED	
ELBOW	FIVE	DONE
	FOUR	
	GOOD	
THAT	GRAY	MATTER
	GREASE	
PREFERENCE	HAS	ALWAYS
RESULTS	HEY	LOOK
	HUNDRED	
IN	I	THOSE
	IF	
ONE	IN	IS
SINGLE	IS	TOUGH
	LEFT	
	LETTERS	
	LOOK	
WITH	MANY	LETTERS
BETWEEN	MARILYN	THIS
SOME	MATH	FIVE
	MATTER	
	MAYBE	
ELEVEN	MORE	LEFT
ENCYCLOPEDIAS	MY	HAS

REPEAT	OF	THEN
GOOD	OLD	FASHIONED
TO	ONE	DISTINCT
HEY	ONLY	MORE
	OR	
	PARAGRAPH	
	PHYSICS	
DICTIONARIES	POSSIBLY	MY
	PREFERENCE	
	PROBABLY	
	READERS	
	READING	
	REALLY	
CANNOT	REPEAT	ANY
	REQUIRE	
	RESULTS	
	SCIENCE	
GREASE	SCIENTISTS	DONT
	SERIOUSLY	
MANY	SHOULD	USED
A	SINGLE	PARAGRAPH
REQUIRE	SKILLS	BEYOND
AND	SO	DO
I	SOME	BEEN
ABOUT	STRUCTURE	THEY
CHALLENGES	SUCH	AS
	THAT	
	THEM	
THEM	THEN	PROBABLY
THESAURUSES	THESAURI	OR
	THESAURUSES	
	THEY	
SUCH	THIS	SKILLS
OF	THOSE	SHOULD
FOUR	THREE	TWO
TRYING	TO	TYPE
REALLY	TOUGH	IF
	TRYING	
	TWO	
	TYPE	
BE	USED	MAYBE
	UTILIZE	
CARE	WANT	ONLY
	WHAT	
READERS	WILL	UTILIZE
	WITH	
	WORDS	
	WRITING	
	YOU	
	YOUR	

Pythian

love your data

(continued from page 20)

```
INSERT INTO RIDDLE  VALUES (NULL, 'jumps', NULL);
INSERT INTO RIDDLE  VALUES ('jumps', 'over', NULL);
INSERT INTO RIDDLE  VALUES ('over', 'the', 'lazy' );
INSERT INTO RIDDLE  VALUES (NULL, 'lazy', NULL);
INSERT INTO RIDDLE  VALUES ('the', 'dog', NULL);
```

The sentence can be reconstructed by traversing the tree in "in-order" fashion, which involves performing the following operations recursively at each node, starting with the root node: first traverse the left sub-tree (if any), then process the node itself, and finally traverse the right sub-tree (if any.) Recursion can be achieved in SQL queries using "recursive common table expressions" (recursive CTEs). However, recursive CTEs only permit "pre-order" traversal (parent, left sub-tree, right sub-tree), not "in-order" traversal. Van Wijk worked around the problem by using a two-phase approach. An ordering string is generated for each node during the pre-order traversal of the tree (Figure 2), and the results are then sorted using the ordering string.

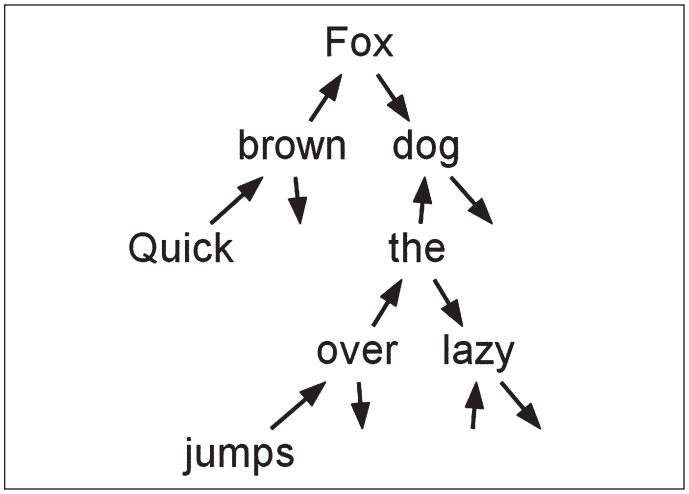


Figure 1.

On March 16, 2011, Chuhnakov submitted two solutions. The first used the MODEL clause, which—in his words—works "automagically." Columns are classified into "dimension" and "measure" arrays where the two terms have the same meaning as for fact tables in data warehouses. The Word2 column is the obvious dimension array, while Word1 and Word3 are measure arrays. Chuhnakov creates another measure array called Text to store messages contained in sub-trees. Each Text value is recursively defined in terms of other Text values by concatenating the left sub-tree with the current node and the right sub-tree. Note that the CV function returns the current value of its argument.

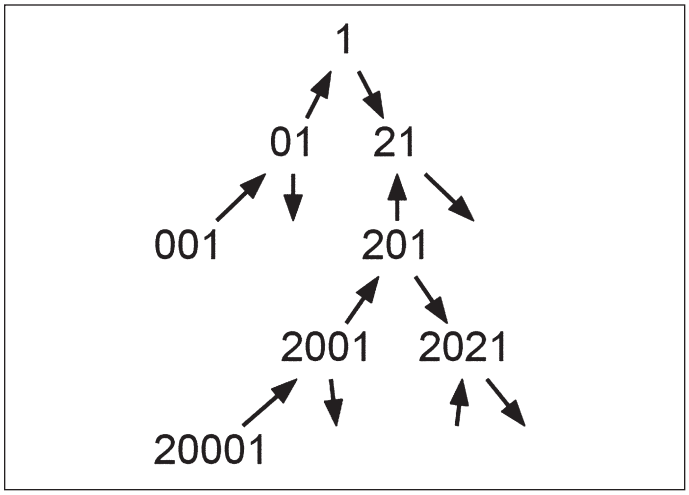


Figure 2.

```
SELECT MAX(text)
  KEEP (DENSE_RANK LAST ORDER BY length(text) ) AS text
FROM
(
  SELECT * FROM riddle
  MODEL
    DIMENSION BY (word2)
    MEASURES
    (
      word1,
      word3,
      CAST(NULL AS VARCHAR2(4000) ) AS text
    )
    RULES AUTOMATIC ORDER
    (
      text  [ word2 ] = trim ( text [ word1 [ CV ( word2 ) ] ]
        || ' ' || CV ( word2 )
        || ' ' || text [ word3 [ CV ( word2 ) ] ] )
    )
);
```

Chuhnakov's second solution was similar to van Wijk's solution but used the CONNECT BY method.

```
WITH CTE AS
(
  SELECT
    r.word2,
    -- Compute the ordering string for this node
    replace(sys_connect_by_path(
      CASE
        WHEN r.word2 = PRIOR word1 THEN '0'
        WHEN r.word2 = PRIOR word3 THEN '2'
      END, '/' ), '/' ) || '1' AS ordering
  FROM riddle r

  -- Identify the root of the binary tree
  START WITH NOT EXISTS (
    SELECT * FROM riddle r2
    WHERE r.word2 IN (r2.word1, r2.word3) )

  -- Identify the left and right nodes if any
  CONNECT BY r.word2 IN (PRIOR r.word1, PRIOR r.word3)
)
-- Sort the words using the ordering string
SELECT word2 FROM CTE ORDER BY ordering;
```

Other solutions using techniques similar to the ones already described were subsequently received. ▲

NoCOUG Conference #133

Post-Conference Reception Sponsored by Quest

The SQL tuning boot camp by SQL goddess Maria Colgan (SQL Maria) and the RDS for Oracle boot camp may have had something to do with the excellent attendance. There were cutesy Valentine's Day doughnuts for breakfast. Gyan Bhatnagar won the ping pong competition at the post-conference reception; Monica Snow won the pool competition.

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED