



Official Publication of the Northern California Oracle Users Group

NoCOUG

J O U R N A L

Vol. 25, No. 4 • NOVEMBER 2011

\$15

25 Years of NoCOUG

Words As Hard As Cannon-balls

An interview with Professor Michael Stonebraker.

See page 4.

Stop Worrying and Love Oracle

By Guy Harrison.

See page 6.

A Relational Model of Data

The original paper by Dr. Codd.

See page 10.

Much more inside . . .

Thanking the Team

Take a moment to think about the huge amount of effort that goes into this publication. Your first thought might be about the care and attention required of the authors. Yes, writing is hard work. Now consider each author's years of hard-won experience; then add it up. The cumulative amount of time spent to acquire the knowledge printed in each issue is decades—maybe even centuries.

But let's take a moment to thank the people who make it possible for us to share this knowledge with you. Without the dedication and skill of our production team, all that we'd have is a jumble of Word files and a bunch of JPEGs. Copyeditor Karen Mead of Creative Solutions transforms our technobabble into readable English. Layout artist Kenneth Lockerbie and graphics guru Richard Repas give the *Journal* its professional layout.

Finally, what really distinguishes this *Journal* is that it is actually printed! Special thanks go to Jo Dziubek and Allen Hom of Andover Printing Services for making us more than just a magnetically recorded byte stream. ▲

—NoCOUG Journal Editor

Table of Contents

President's Message	3	ADVERTISERS	
Words as Hard as Cannon-balls.....	4	Delphix.....	7
Stop Worrying and Love Oracle	6	Quest Software	9
A Relational Model of Data	10	HiT Software.....	21
Second International NoCOUG SQL Challenge	20	Oracle Press	23
Sponsorship Appreciation.....	24	Embarcadero Technologies.....	25
NoCOUG Roll of Honor	26	Confio Software.....	25
Conference Schedule.....	28	Quilogy Services	25
		GridIron Systems	25
		Database Specialists.....	27

Publication Notices and Submission Format

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

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

The submission deadline for the upcoming February 2012 issue is November 30, 2011. Article submissions should be made in Microsoft Word format via email.

Copyright © 2011 by the Northern California Oracle Users Group except where otherwise indicated.

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

2011 NoCOUG Board

President

Iggy Fernandez
iggy_fernandez@hotmail.com

Vice President

Hanan Hit, HIT Consulting, Inc.
hithanan@gmail.com

Secretary/Treasurer

Naren Nagtode, eBay
nagtode@yahoo.com

Director of Membership

Vacant Position

Journal Editor

Dave Abercrombie
vendor_coordinator@nocoug.org

Webmaster

Eric Hutchinson, Independent Consultant
erichutchinson@comcast.net

Vendor Coordinator

Omar Anwar
aanwar@gwmail.gwu.edu

Director of Conference Programming

Chen (Gwen) Shapira, Pythian
cshapi@gmail.com

Director of Marketing

Vacant Position

Training Day Coordinator

Randy Samberg
rsamberg@sbcglobal.net

Volunteer Coordinator

Scott Alexander
alexander_scott@yahoo.com

Member-at-Large

Jen Hong, Stanford University
hong_jen@yahoo.com

Book Reviewer

Brian Hitchcock

ADVERTISING RATES

The *NoCOUG Journal* is published quarterly.

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

Personnel recruitment ads are not accepted.

journal@nocoug.org

The Little User Group That Could!

by Iggy Fernandez



Iggy Fernandez

The story of the *Little Engine That Could* is more than 100 years old and has been retold for generations. The version below is from *The Expositor and Current Anecdotes*, Volume XIII, Number 1 (1911).

Once upon a time a little freight car loaded with coal stood on the track in a coal-yard. The little freight car waited for an engine to pull it up the hill and over the hill and down the hill on the other side. Over the hill in the valley people needed the coal on the little freight car to keep them warm.

By and by a great big engine came along, the smokestack puffing smoke and the bell ringing, "Ding! Ding! Ding!"

"Oh, stop! Please stop, big engine!" said the little freight car. "Pull me up the hill and over the hill and down the hill, to the people in the valley on the other side."

But the big engine said, "I can't, I'm too busy." And away it went—Choo! Choo! Choo! Choo!

The little freight car waited again a long time till a smaller engine came puffing by.

"Oh, stop! dear engine, please stop!" said the little freight car. But the engine puffed a big puff and said, "I can't, you're too heavy." Then away it went, too—Choo! Choo! Choo!

"Oh, dear!" said the little freight car, "what shall I do? The people in the valley on the other side will be so cold without any coal."

After a long time a little pony engine came along, puffing just as hard as a little engine could.

"Oh, stop! dear engine, please stop and take me up the hill and over the hill and down the hill, to the people on the other side," said the patient little freight car.

The pony engine stopped right away and said, "You're very heavy and I'm not very big, but I think I can. I'll try. Hitch on!"

All the way up the hill the pony engine kept saying, "I think I can, I think I can, I think I can. I think I can!" quite fast at first.

Then the hill was steeper and the pony engine had to pull harder and go slower, but all the time it kept saying: "I think-I-can! I-think-I-can!" till it reached the very top with a long puff—"Sh-s-s-s-s!"

It was easy to go down the hill on the other side.

Away went the happy little pony engine saying very fast, "I

thought I could! I thought I could! I thought I could! I thought I could."

Don't forget the lesson, boys and girls. Think you can. Never think you can't.

NoCOUG is the little user group that could! As you might imagine, it requires a vast amount of work to organize a technical conference and publish a printed journal, but no sooner has a conference ended and a journal mailed than it is time to start work on the next conference and the next journal. We have very few resources compared to the national and international user groups, but the NoCOUG volunteers always manage to pull it off, quarter after quarter for 25 long years. There are so many to thank that their names will not all fit on this page, but the award for the longest-serving volunteer goes to Joel Rosingana who—along with staff member Nora Rosingana—were the anchors of NoCOUG for much of its history.

Conference #100 Sponsored by Quest Software—Simplicity at Work is the culmination of our long journey together. It will be held at the Computer History Museum in Mountain View—a fitting location for such an occasion. The museum features marvelous computing artifacts such as a Hollerith Tabulating Machine and an actual operational Babbage Difference Engine—one of two that were constructed in the past decade. Steven Feuerstein—the first Oracle Database expert featured on Wikipedia—will deliver an entertaining yet educational keynote address, *Coding Therapy for Database Professionals*, and will be followed by an all-star cast of internationally recognized speakers, including Craig Shallahamer, Alex Gorbachev, Dan Tow, and Andrew Zitelli. This history-making issue of the *NoCOUG Journal* features an interview with Michael Stonebraker—the high priest of relational databases—as well as the research paper by Dr. Edgar Codd that started the relational revolution in 1970—*A Shared Model of Data for Large Shared Data Banks*.

This is one NoCOUG conference that you won't want to miss. It will be our biggest, baddest conference ever, with the best speakers ever, the best food ever, and the most raffle prizes ever. Will you be there? ▲

"Words As Hard As Cannon-balls"

with Professor Michael Stonebraker



Michael Stonebraker

Michael Stonebraker has been a pioneer of database research and technology for more than a quarter of a century. He was the main architect of the INGRES relational DBMS, the object-relational DBMS, POSTGRES, and the federated data system, Mariposa. All three prototypes were developed at the University of California at Berkeley, where Stonebraker was a professor of computer science for 25 years. Stonebraker moved to MIT in 2001 where he focused on database scalability and opposed the old idea that one size fits all. He was instrumental in building Aurora (an early stream-processing engine), C-Store (one of the first column stores), H-Store (a shared-nothing row-store for OLTP), and SciDB (a DBMS for scientists). He epitomizes the philosophy of the American philosopher Emerson, who said: "A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines...speak what you think today in words as hard as cannon-balls, and tomorrow speak what tomorrow thinks in hard words again, though it contradict every thing you said to-day." (<http://books.google.com/books?id=RI09AAAAcAAJ&pg=PA30>)

Ingres and Postgres—The Backstory

The Ingres RDBMS was the first open-source software product, wasn't it? There's wasn't a GNU Public License at the time, so it was used to create commercial products. Why was Ingres distributed so freely? Which commercial database management systems owe their beginnings to the Ingres project?

Essentially all of the early RDBMS implementations borrowed from either Ingres or System R. Berkeley/CS has a tradition of open-source projects (Unix 4BSD, Ingres, Postgres, etc.)

The embedded query language used by the Ingres RDBMS was QUEL, not SQL. Like SQL, QUEL was based on relational calculus, but—unlike SQL—QUEL failed to win acceptance in the marketplace. Why did QUEL fail to win acceptance in the marketplace?

QUEL is an obviously better language than SQL. See a long paper by Chris Date in 1985 for all of the reasons why. The only reason SQL won in the marketplace was because IBM released DB2 in 1984 without changing the System R query language. At the time, it had sufficient "throw-weight" to ensure that SQL won. If IBM hadn't released DB2, Ingres Corp. and Oracle Corp. would have traded futures.

Postgres and PostgreSQL succeeded Ingres. Why was a replacement for Ingres necessary?

RDBMSs (at the time) were good at business data processing but not at geographic data, medical data, etc. Postgres was designed to extend database technology into other areas. All of the RDBMS vendors have implemented the Postgres extensibility ideas.

Structured Query Language

According to the inventors of SQL, Donald Chamberlin and Raymond Boyce, SQL was intended for the use of accountants, engineers, architects, and urban planners who, "while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language." (<http://www.joakimdalby.dk/HTM/sequel.pdf>) Why didn't things work out the way Chamberlin and Boyce predicted?

SQL is a language for programmers. That was well known by 1985. Vendors implemented other forms-based notations for non-programmers.

Chris Date quotes you as having said that "SQL is intergalactic data-speak." (http://archive.computerhistory.org/resources/access/text/Oral_History/102658166.05.01.acc.pdf#page=43) What did you mean?

SQL is intergalactic data speak—i.e., it is the standard way for programmers to talk to databases.

Dr. Edgar Codd said in 1972: "Requesting data by its properties is far more natural than devising a particular algorithm of sequence of operations for its retrieval. Thus a calculus-oriented language provides a good target language for a more user-oriented source language." (<http://www.eecs.berkeley.edu/~christos/classics/Codd72a.pdf>) With the benefit of hindsight, should we have rejected user-oriented calculus-oriented languages in favor of programmer-oriented algebra-oriented languages with full support for complex operations such as relational division, outer join, semi join, anti join, and star join?

Mere mortals cannot understand division. That doomed the relational algebra. It is interesting to note that science users seem to want algebraic query languages rather than calculus ones. Hence, SciDB supports both.

No to Structured Query Language?

NoSQL is confusing to many in the relational camp. Is

NoSQL a rejection of SQL or of relational database management systems, or both? Or is it just confused?

NoSQL is a collection of 50 or 75 vendors with various objectives. For some of them, the goal is to go fast by rejecting SQL and ACID. I feel these folks are misguided, since SQL is not the performance problem in current RDBMSs. In fact, there is a NewSQL movement that contains very high-performance ACID/SQL implementations.

Other members of the NoSQL movement are focused on document management or semi-structured data—application areas where RDBMSs are known not to work very well. These folks seem to be filling a market not well served by RDBMSs.

You've been championing NewSQL as an answer to NoSQL? What exactly is NewSQL?

Current RDBMSs are too slow to meet some of the demanding current-day applications. This causes some users to look for other alternatives. NewSQL preserves SQL and ACID, and gets much better performance with a different architecture than that used by the traditional RDBMS vendors.

Oracle Database did not enforce referential integrity constraints until Version 7. Back then, Berkeley/CS Professor Larry Rowe suggested that the best way for the CODASYL systems to compete against the relational systems was to point out that they did not [yet] support referential integrity. (http://findarticles.com/p/articles/mi_m0SMG/is_n1_v9/ai_7328281/) Can the new entrants in the DBMS marketplace prevail against the established players without enforcing integrity constraints and business rules?

I have seen several applications where the current RDBMS vendors are more than an order of magnitude too slow to meet the user's needs. In this world, the traditional vendors are nonstarters, and users are looking for something that meets their needs.

The older players in the DBMS marketplace are encumbered by enterprise-grade capabilities that hamper performance. (http://www.think88.com/Examples/Think88_SybaseIQ_wp.pdf) Are enterprise-grade capabilities and performance mutually exclusive?

Everybody should read a great book by Clayton Christensen called *The Innovator's Dilemma*. The established vendors are hampered (in my opinion) primarily by legacy code and an unwillingness to delete or change features in their products. As such, they are 30-year-old technology that is no longer good at anything. The products from the current vendors deserve to be sent to the Home for Tired Software. How to morph from obsolete products to new ones without losing one's customer base is a challenge—which is the topic of the book above.

The Cutting Edge

Why do you believe that it is time for a complete rewrite of relational database management systems?

In every market I can think of, the traditional vendors can be beaten by one to two orders of magnitude by something else. In OLTP, it is NewSQL; in data warehouses, it is column stores; in complex analytics, it is array stores; in document management, it is NoSQL. I see a world where there are (perhaps) a

half-dozen differently architected DBMSs that are purpose built. In other words, I see the death of one-size-fits-all.

Your latest projects, Vertica and VoltDB, claim to leave legacy database management systems in the dust, yet neither of them have published TPC benchmarks. How relevant are TPC benchmarks today?

It is well understood that the standard benchmarks have been constructed largely by the traditional RDBMS vendors to highlight their products. Also, it is clear that they can make their products go an order of magnitude faster on standard benchmarks than is possible on similar workloads.

I encourage customers to demand benchmark numbers on their real applications.

A massively parallel, shared-nothing database management system scales very well if the data can be sharded and if each node has all the data it needs. However, if the physical database design does not meet the needs of the application, then broadcasting of data over the network will result in diminished scalability. How can this problem be prevented?

Physical database design will continue to be a big challenge, for the reasons you mention. It is not clear how to get high performance from an application that does not shard well without giving something else up. This will allow application architects to earn the big bucks for the foreseeable future.

Go West, Young Woman, Go West and Grow Up with the Country

You've had a ringside seat during the relational era and have spent a lot of time in the ring yourself. What would you have changed if you could go back and start all over again?

I would have made Oracle do serious quality control and not confuse future tense and present tense with regard to product features.

Big Data is watching us every minute of the day. Every movement is tracked and recorded by cell towers; every thought is tracked and recorded by search engines; every financial transaction is tracked and recorded by the financial industry; and every text message, email message, and phone conversation is tracked and recorded by service providers. Are databases more evil than good?

A good example is the imminent arrival of sensors in your car, put there by your insurance carrier in exchange for lower rates. Of course, the sensor tracks your every movement, and your privacy is compromised. I expect most customers to voluntarily relinquish their privacy in exchange for lower rates. Cell phones and credit cards are similar; we give up privacy in exchange for some sort of service. I expect that our privacy will be further compromised, off into the future.

As long as we feel this way as a society, privacy will be non-existent.

What advice do you have for the young IT professional just starting out today? Which way is west?

The Internet made text search a mainstream task. Ad placement and web mass personalization are doing likewise for machine learning. Databases are getting bigger faster than hardware is getting cheaper. Hence, I expect DBMS technology will continue to enjoy a place in the sun. ▲

How I Learned to Stop Worrying and Love Oracle

by Guy Harrison



Guy Harrison

If asked to name the most influential and successful software company of the past 25 years, your average layperson would probably nominate IBM, Microsoft, or Google; however, for readers of this journal—and arguably for IT professionals as a whole—the most successful software company has actually been Oracle.

IBM has a longer history, but it's in second or third place at best in most of its key markets. Microsoft undoubtedly enabled and leveraged the PC revolution during the '80s and '90s, but it never quite broke into the enterprise software and server segments, and it seems constantly on the back foot today. Finally, Google undeniably revolutionized search as well as targeted Internet advertising, but in many respects, it's still a one-trick pony and generates only a fraction of its income from direct software sales.

Like many of you, I've made a career based largely around Oracle technologies. Looking back, it's been a good strategy. But there have definitely been many times over the years when I've worried that I may have backed the wrong horse. And while it took some time, I've learned to trust in Oracle.

Genesis of Oracle

In 1970, Edgar Codd first published the famous paper on the relational model. Although IBM was working toward a prototype implementation of a relational database ("System R"), there was no commercial relational system in 1977 when Larry Ellison founded the company that would become Oracle. Common wisdom at the time was that relational databases were incapable of providing the transactional performance delivered by the dominant hierarchical and network databases (IMS, for instance). But Larry Ellison believed that by decoupling the physical and logical representation of data, one could deliver the application and flexible reporting benefits of the relational model, as well as the performance of an optimised physical model.

The battle between the disruptive relational database model, as pioneered by Oracle, and the established—mostly mainframe—databases was waged primarily in the first half of the 1980s. By 1985 the relational database had achieved complete mindshare dominance. Virtually all databases in 1985 claimed to be relational—even those, such as Ashton-Tate's dBase, that were little more than flat file systems.

Battle of the Client Server Behemoths

I first started using Oracle technology around 1987 with Oracle RDBMS 5.1. At that point, a pitched battle between the relational contenders of Oracle, Ingres, Sybase, and Informix was in full swing—and Oracle was not always winning. Oracle was late in delivering features such as stored procedures, cost-based optimization, and referential integrity. Some commentators claimed that Oracle had the inferior technology.

However, Oracle demonstrated—throughout the late '80s and early '90s—a characteristic that we have seen again and again over the years: an ability to out-market and out-manoeuvre competitors who sometimes seem to have the technical high ground. Through aggressive marketing campaigns and relentless sales execution, Oracle won over the bigger corporate accounts; furthermore, while the competitive products often had more hyped-up bells and whistles, Oracle—particularly starting with version 6—introduced foundation technologies that delivered better business benefits. Oracle's superior locking and concurrency mechanisms, and the ability to back up a running database in particular, allowed the development of more serious business applications based on a relational database.

An Object-Oriented Distraction

Around the middle of the 1990s, object-oriented (OO) programming was revolutionizing software languages. It was widely suggested that this new paradigm should extend to databases as well. OO zealots complained of the "impedance mismatch" between relational structures and objects. What was needed, they claimed, was a database that could natively store objects rather than one that required objects to be decomposed into relational structures.

During this period, Informix aggressively claimed technical superiority over Oracle in OO database management and, for a while, it seemed to be winning the battle of public opinion. The most famous example of this was the "Warning: Dinosaurs Crossing" billboard, which Informix planted directly across from Oracle's headquarters!

Apparently accepting the OO paradigm shift, Oracle responded by releasing Oracle 8, which included facets of object database technology, including object tables, nested tables, and Varrays.

But it turned out that in the real world, these OO features addressed a demand that did not really exist. Informix imploded in an Enron-style fraud scandal in 1997. And by then, the IT world was focused on a true paradigm shift with e-commerce and the Internet, as well as the impending Y2K challenge.

The Internet Changes Everything

The rapid uptake of the Internet and the World Wide Web is, of course, as big a paradigm shift in computing—and maybe society—as many of us are likely to see. It generated a massive investment in software and hardware as businesses strived to become Internet-enabled.

Around the same time, Y2K projects provided another huge influx of funding, as all critical systems were examined for vulnerabilities and bugs. Though the actual impact of Y2K bugs was negligible, the effect on IT budgets was massive.

In 2000, with Y2K out of the way and the e-commerce bubble in full swing, we experienced an IT gold rush of unparalleled proportions. For Internet start-ups, money was no object and scalability was everything. This “budgetless funding” environment encouraged purchasing only premium hardware and software. Oracle—as the premium and most scalable database—thrived during this period.

The World Is Not Enough

Having fought and—seemingly won—the database battle, Oracle in the early 2000s was highly motivated to diversify beyond the database business. Oracle applications had been a significant business for some time and Oracle had introduced their own Java application server, but they were not the outright leader in either of these market segments.

Throughout the 2000s, Oracle achieved leadership in both the middleware and application-software categories through an aggressive series of acquisitions, which included PeopleSoft, Siebel, JD Edwards, Hyperion, BEA, and dozens of other significant acquisitions.

The Innovator's Dilemma

With the collapse of the e-commerce bubble in 2001 and throughout the succeeding decade, IT moved from a gold-rush mentality to a more austere environment in which return on investment (ROI) and total cost of ownership (TCO) became the driving principles. This gave the relatively cheaper Microsoft SQL Server database an edge in many cases and raised the possibility of Oracle's dominant position being disrupted by a low-cost, good-enough competitor—the classic “innovator's dilemma.”¹

During the years 2003 to 2006, MySQL looked like a significant threat to Oracle's database business. The MySQL free open source edition was enormously popular, and enterprise pricing was a fraction of Oracle pricing. With the enterprise focus on cost containment, MySQL looked potentially disruptive.

Oracle's response to MySQL's popularity stands, in my mind, as a classic defence to disruptive technology. Oracle competed at the low end by releasing a free version: Oracle XE. At the high end, Oracle had at last perfected its database clus-

tering technology (i.e., RAC). RAC allowed Oracle to maintain its dominance for highly available, mission-critical databases. In addition, Oracle acquired some of the MySQL foundation technologies, such as InnoDB and BerkeleyDB. Oracle effectively “disrupted the disruptor.” Of course, following the Sun acquisition, MySQL became an Oracle technology.

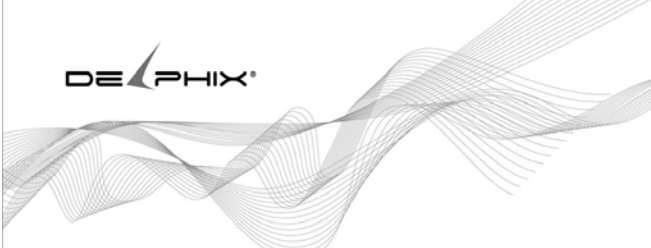
Shared Disk Clustering Pays Off at Last

Oracle had pursued a vision of database clustering since the early '90s. Oracle 6.1 introduced an early version of Oracle's parallel server (OPS) technology in beta. Initially, performance of the clustered Oracle database was far from satisfactory. The OPS technology was eventually released in production during the Oracle 7 timeframe, but it was regarded as a complex and risky technology and was not widely adopted.

However, Oracle relentlessly pursued a vision of database clustering that was virtually unique in the industry. Most database clustering technologies are “shared nothing”—in which the database is partitioned across cluster nodes, which “share nothing” with other nodes. Such an approach requires manual partitioning and load balancing and is impractical for many applications. Oracle believed that “shared disk” database clusters—in which each member of the cluster had access to the entire database—were possible. Such a cluster would allow an application to use a database cluster without modification.

Oracle released Real Application Clusters (RAC) with the Oracle 9i database. Competitors continue to claim that RAC is


¹ <http://amzn.to/mUftel>



DELPHIX®

Database Virtualization Software

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



www.delphix.com

a flawed technology—declaring their own “shared nothing” clustering as superior. Yet the widespread uptake of RAC within the Oracle community speaks for itself; the technology has clearly been successful.

It's Not Water Vapor!

In 2009, “cloud mania” hit the IT industry, promoting a vision of a world in which all computing would be done “in the cloud”—effectively outsourcing IT to companies such as Amazon and Google. Larry Ellison was famously skeptical—at one point claiming, “It’s absurdity—it’s nonsense . . . What are you talking about? It’s not water vapor. It’s a computer attached to a network!”²

However, cloud computing can be seen as a synonym for Internet grid computing—in which computing resources are provided on demand from a pool of virtualized computing resources made available across the Internet. Ironically, Oracle had done more than any other commercial company to popularize many of the concepts underlying cloud computing through its grid vision (after all, the “g” in 10g stands for grid).

Despite that, Oracle has continued to develop key “cloud-like” technologies, such as Oracle Virtual Machine (VM). We expect to hear more from Oracle regarding a cloud vision at OOW 2011.

Exadata

One of the reasons for Oracle’s early success was its portability; by writing code in the then-relatively new “C” language, Oracle was capable of porting its software to new hardware platforms with relative ease. The downside of this portability is that Oracle software is often installed upon under-configured or unbalanced hardware configurations. This actually became a disadvantage in the data-warehousing world, where vendors such as Teradata would deliver a solution as a bundled appliance running a balanced hardware and software stack.

Oracle was, therefore, highly motivated to provide a database appliance. Their first attempt—Exadata V1—was based on a partnership with HP. The subsequent acquisition of Sun Microsystems in 2010 allowed Oracle to create a fully integrated appliance completely in-house. The result—Exadata V2—has been a remarkable technology and market success story.

Exadata succeeds, in my mind, on three levels. First, it uses best-of-breed standard technologies such as InfiniBand and Flash solid state disk (SSD). Second, it exploits some of the unique capabilities of the Oracle DBMS—shared disk clustering, ASM, parallel query, and so on. Third, Oracle has incorporated new technologies, such as smart scans, enhanced hybrid columnar compression, and storage indexes. The result is a highly optimized hardware or software combination that will be very difficult for competitors to match.

Combined with Exalogic—an Exadata-like appliance for Oracle middleware—Oracle can offer a complete hardware and software solution, the likes of which has not been seen since the days of the IBM mainframe.

Why I Stopped Worrying

It’s been a long time since I—or probably anybody—has had reason to worry about Oracle. Oracle has shown, over

more than two decades, a remarkable ability to execute on technology and generate profitable growth. Oracle’s technical strength is coupled with an unwavering business focus: Oracle never forgets how to make money from their technical innovations. As an individual, I’ve found working with Oracle technology to be a constant challenge—without a boring minute. I look forward to seeing what Oracle comes up with next.

Quest Viewpoint

Since the 1990s, Quest Software has developed innovative tools, including flagship brands such as Toad® and Foglight®, to make Oracle professionals more productive and efficient, and we expect to continue doing so for decades to come. As a true independent software vendor, Quest has no stake in a particular database and no potential conflicts of interests when helping you maximize your database and related infrastructure investments.

As a result, Quest’s footprint includes support for not only Oracle but a variety of other major relational and non-relational database platforms to meet our customers’ needs for more heterogeneous toolsets. Quest is committed to providing tools with the deepest functionality in the business while giving you the freedom to choose the platforms that are the best fit, and the most cost-effective, for your environment.

Toad, for example, not only provides specific editions built for various job functions, but it also supports more than 15 types of databases, including NoSQL platforms like Hadoop. Imagine being able to leverage your existing skill set on new platforms and technologies to meet future demands: you can today with Toad. It gives you a simple, consistent way to build, access, manage, optimize, and maintain database applications. ▲

Guy Harrison is an Oracle ACE and director of research and development at Quest Software. Guy is the author of Oracle Performance Survival Guide (Prentice Hall, 2009) and MySQL Stored Procedure Programming (O’Reilly, 2006, with Steven Feuerstein) as well as other books, articles, and presentations on database technology. Guy also writes a monthly column for Database Trends and Applications (<http://www.dbta.com>). Guy can be found on the Internet at <http://www.guyharrison.net> and on email at guy.harrison@quest.com, and he is @guy-harrison on Twitter.

Quest Software (Nasdaq: QSFT) simplifies and reduces the cost of managing IT for more than 100,000 customers worldwide. Our innovative solutions make solving the toughest IT management problems easier, enabling customers to save time and money across physical, virtual, and cloud environments. For more information about Quest solutions, visit <http://www.quest.com>.

² <http://www.youtube.com/watch?v=UOEFXaWHppE>



2,000,001

ORACLE PROFESSIONALS COUNT ON TOAD

ORACLE PROFESSIONALS COUNT ON TOAD

Two Million Database Professionals Count on One Solution.

Simply the best for Oracle database professionals - Toad 11. Supported by over a decade of proven excellence, only Toad combines the deepest functionality available, extensive automation, and a workflow that enables database professionals of all skill and experience levels to work efficiently and accurately. Countless organizations empower their database professionals with Toad. The best just got better.

Watch the video at www.quest.com/Toad11SimplyBetter.



© 2011 Quest Software, Inc. ALL RIGHTS RESERVED. Quest, Quest Software and the Quest Software logo are registered trademarks of Quest Software, Inc. in the U.S.A. and/or other countries. All other trademarks and registered trademarks are property of their respective owners. ADD_Toad_FullPage_US_INK_201108

A Relational Model of Data for Large Shared Data Banks

E. F. Codd^{*}

IBM Research Laboratory, San Jose, California

ABSTRACT

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n -ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

1. RELATIONAL MODEL AND NORMAL FORM

1.1 Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language

which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the “connection trap”).

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2 Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1 Ordering Dependence

Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the stored ordering. Those application programs which take advantage of the stored ordering of a file are likely to

^{*}E. F. Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377-387. DOI=10.1145/362384.362685 <http://doi.acm.org/10.1145/362384.362685>

fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one. Similar remarks hold for a stored ordering implemented by means of pointers.

It is unnecessary to single out any system as an example, because all the well-known information systems that are marketed today fail to make a clear distinction between order of presentation on the one hand and stored ordering on the other. Significant implementation problems must be solved to provide this kind of independence.

1.2.2 Indexing Dependence

In the context of formatted data, an index is usually thought of as a purely performance-oriented component of the data representation. It tends to improve response to queries and updates and, at the same time, slow down response to insertions and deletions. From an informational standpoint, an index is a redundant component of the data representation. If a system uses indices at all and if it is to perform well in an environment with changing patterns of activity on the data bank, an ability to create and destroy indices from time to time will probably be necessary. The question then arises: Can application programs and terminal activities remain invariant as indices come and go?

Present formatted data systems take widely different approaches to indexing. TDMS [7] unconditionally provides indexing on all attributes. The presently released version of IMS [5] provides the user with a choice for each file: a choice between no indexing at all (the hierarchic sequential organization) or indexing on the primary key only (the hierarchic indexed sequential organization). In neither case is the user's application logic dependent on the existence of the unconditionally provided indices. IDS [8], however, permits the file designers to select attributes to be indexed and to incorporate indices into the file structure by means of additional chains. Application programs taking advantage of the performance benefit of these indexing chains must refer to those chains by name. Such programs do not operate correctly if these chains are later removed.

1.2.3 Access Path Dependence

Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. A simple example follows.

Suppose the data bank contains information about parts and projects. For each part, the part number, part name, part description, quantity-on-hand, and quantity-on-order are recorded. For each project, the project number, project name, project description are recorded. Whenever a project makes use of a certain part, the quantity of that part committed to the given project is also recorded. Suppose that the system requires the user or file designer to declare or define the data in terms of tree structures. Then, any one of the hierarchical structures may be adopted for the information mentioned above (see Structures 1-5).

Now, consider the problem of printing out the part number, part name, and quantity committed for every part used in the project whose project name is "alpha." The following observations may be made regardless of which available tree-oriented information system is selected to tackle this problem. If a program *P* is developed for this problem assuming one of the five structures above—that is, *P* makes

Structure 1. Projects Subordinate to Parts

<i>File</i>	<i>Segment</i>	<i>Fields</i>
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description
		quantity committed

Structure 2. Parts Subordinate to Projects

<i>File</i>	<i>Segment</i>	<i>Fields</i>
F	PROJECT	project #
		project name
		project description
	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order

Structure 3. Parts and Projects as Peers
Commitment Relationship Subordinate to Projects

<i>File</i>	<i>Segment</i>	<i>Fields</i>
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description

Structure 4. Parts and Projects as Peers
Commitment Relationship Subordinate to Parts

<i>File</i>	<i>Segment</i>	<i>Fields</i>
F	PART	part #
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		quantity committed

Structure 5. Parts, Projects, and Commitment
Relationship as Peers

<i>File</i>	<i>Segment</i>	<i>Fields</i>
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description

no test to determine which structure is in effect—then P will fail on at least three of the remaining structures. More specifically, if P succeeds with structure 5, it will fail with all the others; if P succeeds with structure 3 or 4, it will fail with at least 1, 2, and 5; if P succeeds with 1 or 2, it will fail with at least 3, 4, and 5. The reason is simple in each case. In the absence of a test to determine which structure is in effect, P fails because an attempt is made to execute a reference to a nonexistent file (available systems treat this as an error) or no attempt is made to execute a reference to a file containing needed information. The reader who is not convinced should develop sample programs for this simple problem.

Since, in general, it is not practical to develop application programs which test for all tree structurings permitted by the system, these programs fail when a change in structure becomes necessary.

Systems which provide users with a network model of the data run into similar difficulties. In both the tree and network cases, the user (or his program) is required to exploit a collection of user access paths to the data. It does not matter whether these paths are in close correspondence with pointer-defined paths in the stored representation—in IDS the correspondence is extremely simple, in TDMS it is just the opposite. The consequence, regardless of the stored representation, is that terminal activities and programs become dependent on the continued existence of the user access paths.

One solution to this is to adopt the policy that once a user access path is defined it will not be made obsolete until all application programs using that path have become obsolete. Such a policy is not practical, because the number of access paths in the total model for the community of users of a data bank would eventually become excessively large.

1.3 A Relational View of Data

The term relation is used here in its accepted mathematical sense. Given sets S_1, S_2, \dots, S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n -tuples each of which has its first element from S_1 , its second element from S_2 , and so on.¹ We shall refer to S_j as the j th domain of R . As defined above, R is said to have *degree* n . Relations of degree 1 are often called *unary*, degree 2 *binary*, degree 3 *ternary*, and degree n *n -ary*.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An array which represents an n -ary relation R has the following properties:

1. Each row represents an n -tuple of R .
2. The ordering of rows is immaterial.
3. All rows are distinct.
4. The ordering of columns is significant—it corresponds to the ordering S_1, S_2, \dots, S_n of the domains on which R is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
5. The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

¹More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \dots \times S_n$.

<i>supply</i>	(<i>supplier</i>	<i>part</i>	<i>project</i>	<i>quantity</i>)
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

FIG. 1. A relation of degree 4

<i>component</i>	(<i>part</i>	<i>part</i>	<i>quantity</i>)
	1	5	9
	2	5	7
	3	5	2
	2	6	12
	3	6	3
	4	7	1
	6	7	1

FIG. 2. A relation with two identical domains

The example in Figure 1 illustrates a relation of degree 4, called *supply*, which reflects the shipments-in-progress of parts from specified suppliers to specified projects in specified quantities.

One might ask: If the columns are labeled by the name of corresponding domains, why should the ordering of columns matter? As the example in Figure 2 shows, two columns may have identical headings (indicating identical domains) but possess distinct meanings with respect to the relation. The relation depicted is called *component*. It is a ternary relation, whose first two domains are called *part* and third domain is called *quantity*. The meaning of *component* (x, y, z) is that part x is an immediate component (or subassembly) of part y , and z units of part x are needed to assemble one unit of part y . It is a relation which plays a critical role in the parts explosion problem.

It is a remarkable fact that several existing information systems (chiefly those based on tree-structured files) fail to provide data representations for relations which have two or more identical domains. The present version of IMS/360 [5] is an example of such a system.

The totality of data in a data bank may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each n -ary relation may be subject to insertion of additional n -tuples, deletion of existing ones, and alteration of components of any of its existing n -tuples.

In many commercial, governmental, and scientific data banks, however, some of the relations are of quite high degree (a degree of 30 is not at all uncommon). Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering *supplier*, then *part*, then *project*, then *quantity* in the relation *supply*). Accordingly, we propose that users deal, not with relations which are domain-ordered, but with *relationships* which are their domain-unordered counterparts.² To accomplish this, domains must be uniquely identifiable at least within any given relation, without using position. Thus, where there are two or more identical domains, we require in each case that the domain name be qualified by a distinctive *role name*, which serves to identify the role played by that domain in the given relation. For example, in the relation *component* of Figure 2, the first domain *part* might be

²In mathematical terms, a relationship is an equivalence class of those relations that are equivalent under permutation of domains (see Section 2.1.1).

qualified by the role name *sub*, and the second by *super*, so that users could deal with the relationship *component* and its domains—*sub.part super.part, quantity*—without regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary).³ Even this information might be offered in menu style by the system (subject to security and privacy constraints) upon request by the user.

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, nonsimple domain) and establish some links with terminology currently in use in information systems programming. In the remainder of this paper, we shall not bother to distinguish between relations and relationships except where it appears advantageous to be explicit.

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called *part* is defined on the following domains:

1. part number
2. part name
3. part color
4. part weight
5. quantity on hand
6. quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part names, and part numbers are. We shall call the set of values represented at some instant the *active domain* at that instant.

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (*n*-tuple) of that relation. Such a domain (or combination) is called a *primary key*. In the example above, part number would be a primary key, while part color would not be. A primary key is *nonredundant* if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called the *primary key* of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such crossreferences. We shall call a domain (or domain combination) of relation *R* a *foreign key* if it is not the primary key of *R* but

its elements are values of the primary key of some relation *S* (the possibility that *S* and *R* are identical is not excluded). In the relation *supply* of Figure 1, the combination of *supplier, part, project* is the primary key, while each of these three domains taken separately is a foreign key.

In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the *supply* relation). This distinction is difficult to maintain when one may have foreign keys in any relation whatsoever. In the user's relational model there appears to be no advantage to making such a distinction (there may be some advantage, however, when one applies relational concepts to machine representations of the user's set of relationships).

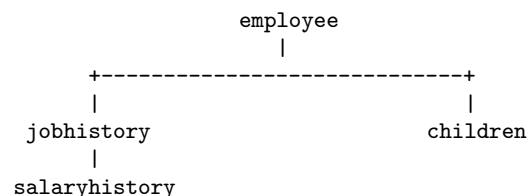
So far, we have discussed examples of relations which are defined on simple domains—domains whose elements are atomic (nondecomposable) values. Nonatomic values can be discussed within the relational framework. Thus, some domains may have relations as elements. These relations may, in turn, be defined on nonsimple domains, and so on. For example, one of the domains on which the relation *employee* is defined might be *salary history*. An element of the salary history domain is a binary relation defined on the domain *date* and the domain *salary*. The *salary history* domain is the set of all such binary relations. At any instant of time there are as many instances of the *salary history* relation in the data bank as there are employees. In contrast, there is only one instance of the *employee* relation.

The terms attribute and repeating group in present data base terminology are roughly analogous to simple domain and nonsimple domain, respectively. Much of the confusion in present terminology is due to failure to distinguish between type and instance (as in "record") and between components of a user model of the data on the one hand and their machine representation counterparts on the other hand (again, we cite "record" as an example).

1.4 Normal Form

A relation whose domains are all simple can be represented in storage by a two-dimensional column-homogeneous array of the kind discussed above. Some more complicated data structure is necessary for a relation with one or more nonsimple domains. For this reason (and others to be cited below) the possibility of eliminating nonsimple domains appears worth investigating.⁴ There is, in fact, a very simple elimination procedure, which we shall call normalization.

Consider, for example, the collection of relations exhibited in Figure 3(a). *Job history* and *children* are nonsimple domains of the relation *employee*. *Salary history* is a nonsimple domain of the relation *job history*. The tree in Figure 3(a) shows just these interrelationships of the nonsimple domains.



³Naturally, as with any data put into and retrieved from a computer system, the user will normally make far more effective use of the data if he is aware of its meaning.

⁴M. E. Sanko of IBM, San Jose, independently recognized the desirability of eliminating nonsimple domains.

employee (*man#*, name, birthdate, jobhistory, children)
 jobhistory (*jobdate*, title, salaryhistory)
 salaryhistory (*salarydate*, salary)
 children (*childname*, birthyear)

FIG. 3(a). Unnormalized set

employee' (*man#*, name, birthdate)
 jobhistory' (*man#*, jobdate, title)
 salaryhistory' (*man#*, jobdate, salarydate, salary)
 children' (*man#*, childname, birthyear)

FIG. 3(b). Normalized set

Normalization proceeds as follows. Starting with the relation at the top of the tree, take its primary key and expand each of the immediately subordinate relations by inserting this primary key domain or domain combination. The primary key of each expanded relation consists of the primary key before expansion augmented by the primary key copied down from the parent relation. Now, strike out from the parent relation all nonsimple domains, remove the top node of the tree, and repeat the same sequence of operations on each remaining subtree.

The result of normalizing the collection of relations in Figure 3(a) is the collection in Figure 3(b). The primary key of each relation is italicized to show how such keys are expanded by the normalization.

If normalization as described above is to be applicable, the unnormalized collection of relations must satisfy the following conditions:

1. The graph of interrelationships of the nonsimple domains is a collection of trees.
2. No primary key has a component domain which is nonsimple.

The writer knows of no application which would require any relaxation of these conditions. Further operations of a normalizing kind are possible. These are not discussed in this paper.

The simplicity of the array representation which becomes feasible when all relations are cast in normal form is not only an advantage for storage purposes but also for communication of bulk data between systems which use widely different representations of the data. The communication form would be a suitably compressed version of the array representation and would have the following advantages:

1. It would be devoid of pointers (address-valued or displacement-valued).
2. It would avoid all dependence on hash addressing schemes.
3. It would contain no indices or ordering lists.

If the user's relational model is set up in normal form, names of items of data in the data bank can take a simpler form than would otherwise be the case. A general name would take a form such as

$$R(g).r.d$$

where R is a relational name; g is a generation identifier (optional); r is a role name (optional); d is a domain name. Since g is needed only when several generations of a given relation exist, or are anticipated to exist, and r is needed only when the relation R has two or more domains named d , the simple form $R.d$ will often be adequate.

1.5 Some Linguistic Aspects

The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus. A first-order predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding (with appropriate syntactic modification) in a variety of host languages (programming, command- or problem-oriented). While it is not the purpose of this paper to describe such a language in detail, its salient features would be as follows.

Let us denote the data sublanguage by R and the host language by H . R permits the declaration of relations and their domains. Each declaration of a relation identifies the primary key for that relation. Declared relations are added to the system catalog for use by any members of the user community who have appropriate authorization. H permits supporting declarations which indicate, perhaps less permanently, how these relations are represented in storage. R permits the specification for retrieval of any subset of data from the data bank. Action on such a retrieval request is subject to security constraints.

The universality of the data sublanguage lies in its descriptive ability (not its computing ability). In a large data bank each subset of the data has a very large number of possible (and sensible) descriptions, even when we assume (as we do) that there is only a finite set of function subroutines to which the system has access for use in qualifying data for retrieval. Thus, the class of qualification expressions which can be used in a set specification must have the descriptive power of the class of well-formed formulas of an applied predicate calculus. It is well known that to preserve this descriptive power it is unnecessary to express (in whatever syntax is chosen) every formula of the selected predicate calculus. For example, just those in prenex normal form are adequate [9].

Arithmetic functions may be needed in the qualification or other parts of retrieval statements. Such functions can be defined in H and invoked in R .

A set so specified may be fetched for query purposes only, or it may be held for possible changes. Insertions take the form of adding new elements to declared relations without regard to any ordering that may be present in their machine representation. Deletions which are effective for the community (as opposed to the individual user or subcommunities) take the form of removing elements from declared relations. Some deletions and updates may be triggered by others, if deletion and update dependencies between specified relations are declared in R .

One important effect that the view adopted toward data has on the language used to retrieve it is in the naming of data elements and sets. Some aspects of this have been discussed in the previous section. With the usual network view, users will often be burdened with coining and using more relation names than are absolutely necessary, since names are associated with paths (or path types) rather than with relations.

Once a user is aware that a certain relation is stored, he will expect to be able to exploit⁵ it using any combination of its arguments as "knowns" and the remaining arguments as "unknowns," because the information (like Everest) is there. This is a system feature (missing from many current infor-

⁵Exploiting a relation includes query, update, and delete.

mation systems) which we shall call (logically) *symmetric exploitation* of relations. Naturally, symmetry in performance is not to be expected.

To support symmetric exploitation of a single binary relation, two directed paths are needed. For a relation of degree n , the number of paths to be named and controlled is n factorial.

Again, if a relational view is adopted in which every n -ary relation ($n > 2$) has to be expressed by the user as a nested expression involving only binary relations (see Feldman's LEAP System [10], for example) then $2n - 1$ names have to be coined instead of only $n + 1$ with direct n -ary notation as described in Section 1.2. For example, the 4-ary relation *supply* of Figure 1, which entails 5 names in n -ary notation, would be represented in the form

$$P(\text{supplier}, Q(\text{part}, R(\text{project}, \text{quantity})))$$

in nested binary notation and, thus, employ 7 names.

A further disadvantage of this kind of expression is its asymmetry. Although this asymmetry does not prohibit symmetric exploitation, it certainly makes some bases of interrogation very awkward for the user to express (consider, for example, a query for those parts and quantities related to certain given projects via Q and R).

1.6 Expressible, Named, and Stored Relations

Associated with a data bank are two collections of relations: the *named set* and the *expressible set*. The named set is the collection of all those relations that the community of users can identify by means of a simple name (or identifier). A relation R acquires membership in the named set when a suitably authorized user declares R ; it loses membership when a suitably authorized user cancels the declaration of R .

The expressible set is the total collection of relations that can be designated by expressions in the data language. Such expressions are constructed from simple names of relations in the named set; names of generations, roles and domains; logical connectives; the quantifiers of the predicate calculus;⁶ and certain constant relation symbols such as $=$, $>$. The named set is a subset of the expressible set—usually a very small subset.

Since some relations in the named set may be time-independent combinations of others in that set, it is useful to consider associating with the named set a collection of statements that define these time-independent constraints. We shall postpone further discussion of this until we have introduced several operations on relations (see Section 2).

One of the major problems confronting the designer of a data system which is to support a relational model for its users is that of determining the class of stored representations to be supported. Ideally, the variety of permitted data representations should be just adequate to cover the spectrum of performance requirements of the total collection of installations. Too great a variety leads to unnecessary overhead in storage and continual reinterpretation of descriptions for the structures currently in effect.

For any selected class of stored representations the data system must provide a means of translating user requests expressed in the data language of the relational model into

⁶Because each relation in a practical data bank is a finite set at every instant of time, the existential and universal quantifiers can be expressed in terms of a function that counts the number of elements in any finite set.

corresponding—and efficient—actions on the current stored representation. For a high level data language this presents a challenging design problem. Nevertheless, it is a problem which must be solved—as more users obtain concurrent access to a large data bank, responsibility for providing efficient response and throughput shifts from the individual user to the data system.

2. REDUNDANCY AND CONSISTENCY

2.1 Operations on Relations

Since relations are sets, all of the usual set operations are applicable to them. Nevertheless, the result may not be a relation; for example, the union of a binary relation and a ternary relation is not a relation.

The operations discussed below are specifically for relations. These operations are introduced because of their key role in deriving relations from other relations. Their principal application is in noninferential information systems—systems which do not provide logical inference services—although their applicability is not necessarily destroyed when such services are added.

Most users would not be directly concerned with these operations. Information systems designers and people concerned with data bank control should, however, be thoroughly familiar with them.

2.1.1 Permutation

A binary relation has an array representation with two columns. Interchanging these columns yields the converse relation. More generally, if a permutation is applied to the columns of an n -ary relation, the resulting relation is said to be a *permutation* of the given relation. There are, for example, $4! = 24$ permutations of the relation *supply* in Figure 1, if we include the identity permutation which leaves the ordering of columns unchanged.

Since the user's relational model consists of a collection of relationships (domain-unordered relations), permutation is not relevant to such a model considered in isolation. It is, however, relevant to the consideration of stored representations of the model. In a system which provides symmetric exploitation of relations, the set of queries answerable by a stored relation is identical to the set answerable by any permutation of that relation. Although it is logically unnecessary to store both a relation and some permutation of it, performance considerations could make it advisable.

2.1.2 Projection

Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a *projection* of the given relation.

A selection operator π is used to obtain any desired permutation, projection, or combination of the two operations. Thus, if L is a list of k indices⁷ $L = i_1, i_2, \dots, i_k$ and R is an n -ary relation ($n \geq k$), then $\pi_L(R)$ is the k -ary relation whose j th column is column i_j of R ($j = 1, 2, \dots, k$) except that duplication in resulting rows is removed. Consider the relation *supply* of Figure 1. A permuted projection of this relation is exhibited in Figure 4. Note that, in this particular case, the projection has fewer n -tuples than the relation

⁷When dealing with relationships, we use domain names (role-qualified whenever necessary) instead of domain positions.

from which it is derived.

2.1.3 Join

Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a ternary relation which preserves all of the information in the given relations?

The example in Figure 5 shows two relations R , S , which are joinable without loss of information, while Figure 6 shows a join of R with S . A binary relation R is *joinable* with a binary relation S if there exists a ternary relation U such that $\pi_{12}(U) = R$ and $\pi_{23}(U) = S$. Any such ternary relation is called a *join* of R with S . If R , S are binary relations such that $\pi_2(R) = \pi_1(S)$, then R is joinable with S . One join that always exists in such a case is the *natural join* of R with S defined by

$$R * S = \{(a, b, c) : R(a, b) \wedge S(b, c)\}$$

where $R(a, b)$ has the value true if (a, b) is a member of R and similarly for $S(b, c)$. It is immediate that

$$\pi_{12}(R * S) = R$$

and

$$\pi_{23}(R * S) = S.$$

Note that the join shown in Figure 6 is the natural join of R with S from Figure 5. Another join is shown in Figure 7.

Inspection of these relations reveals an element (element 1) of the domain *part* (the domain on which the join is to be made) with the property that it possesses more than one relative under R and also under S . It is this element which gives rise to the plurality of joins. Such an element in the joining domain is called a *point of ambiguity* with respect to the joining of R with S .

If either $\pi_{21}(R)$ or S is a function,⁸ no point of ambiguity can occur in joining R with S . In such a case, the natural join of R with S is the only join of R with S . Note that the reiterated qualification “of R with S ” is necessary, because S might be joinable with R (as well as R with S), and this join would be an entirely separate consideration. In Figure 5, none of the relations R , $\pi_{21}(R)$, S , $\pi_{21}(S)$ is a function.

Ambiguity in the joining of R with S can sometimes be resolved by means of other relations. Suppose we are given, or can derive from sources independent of R and S , a relation T on the domains *project* and *supplier* with the following properties :

1. $\pi_1(T) = \pi_2(S)$,
2. $\pi_2(T) = \pi_1(R)$,
3. $T(j, s) \rightarrow \exists p(R(S, p) \wedge S(p, j))$,
4. $R(s, p) \rightarrow \exists j(S(p, j) \wedge T(j, s))$,
5. $S(p, j) \rightarrow \exists s(T(j, s) \wedge R(s, p))$,

then we may form a three-way join of R , S , T ; that is, a ternary relation such that

$$\pi_{12}(U) = R, \pi_{23}(U) = S, \pi_{31}(U) = T.$$

Such a join will be called a *cyclic 3-join* to distinguish it from a *linear 3-join* which would be a quaternary relation V such that

$$\pi_{12}(V) = R, \pi_{23}(V) = S, \pi_{34}(V) = T.$$

$\Pi_{31}(\text{supply})$	(<i>project</i>	<i>supplier</i>)
	5	1
	5	2
	1	4
	7	2

FIG. 4. A permuted projection of the relation in Figure 1

R	(<i>supplier</i>	<i>part</i>)	S	(<i>part</i>	<i>project</i>)
1	1	1	1	1	1
2	1	1	1	2	2
2	2	2	2	1	1

FIG. 5. Two joinable relations

$R * S$	(<i>supplier</i>	<i>part</i>	<i>project</i>)
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

FIG. 6. The natural join of R with S (from Figure 5)

U	(<i>supplier</i>	<i>part</i>	<i>project</i>)
	1	1	2
	2	1	1
	2	2	1

FIG. 7. Another join of R with S (from Figure 5)

R	(s	p)	S	(p	j)	T	(j	s)
1	a			a	d		d	1
2	a			a	e		d	2
2	b			b	d		e	2
				b	e		e	2

FIG. 8. Binary relations with a plurality of cyclic 3-joins

U	(s	p	j)	U'	(s	p	j)
	1	a	d		1	a	d
	2	a	e		2	a	d
	2	b	d		2	a	e
	2	b	e		2	b	d
					2	b	e

FIG. 9. Two cyclic 3-joins of the relations in Figure 8

$R \cdot S$	(<i>project</i>	<i>supplier</i>)
	1	1
	1	2
	2	1
	2	2

FIG. 10. The natural composition of R with S (from Figure 5)

T	(<i>project</i>	<i>supplier</i>)
	1	2
	2	1

FIG. 11. Another composition of R with S (from Figure 5)

R	(supplier	part)	S	(part	project)
1	a		a	g	
1	b		b	f	
1	c		c	f	
2	c		c	g	
2	d		d	g	
2	e		e	f	

FIG. 12. Many joins, only one composition

While it is possible for more than one cyclic 3-join to exist (see Figures 8,9, for an example), the circumstances under which this can occur entail much more severe constraints than those for a plurality of 2-joins. To be specific, the relations R , S , T must possess points of ambiguity with respect to joining R with S (say point x), S with T (say y), and T with R (say z), and, furthermore, y must be a relative of x under S , z a relative of y under T , and x a relative of z under R . Note that in Figure 8 the points $x = a; y = d; z = 2$ have this property.

The natural linear 3-join of three binary relations R , S , T is given by

$$R * S * T = \{(a, b, c, d) : R(a, b) \wedge S(b, c) \wedge T(c, d)\}$$

where parentheses are not needed on the left-hand side because the natural 2-join ($*$) is associative. To obtain the cyclic counterpart, we introduce the operator γ which produces a relation of degree $n - 1$ from a relation of degree n by tying its ends together. Thus, if R is an n -ary relation ($n \geq 2$), the *tie* of R is defined by the equation

$$\gamma(R) = \{(a_1, a_2, \dots, a_{n-1}) : R(a_1, a_2, \dots, a_{n-1}, a_n) \wedge a_1 = a_n\}.$$

We may now represent the natural cyclic 3-join of R , S , T by the expression

$$\gamma(R * S * T).$$

Extension of the notions of linear and cyclic 3-join and their natural counterparts to the joining of n binary relations (where $n \geq 3$) is obvious. A few words may be appropriate, however, regarding the joining of relations which are not necessarily binary. Consider the case of two relations R (degree r), S (degree s) which are to be joined on p of their domains ($p < r, p < s$). For simplicity, suppose these p domains are the last p of the r domains of R , and the first p of the s domains of S . If this were not so, we could always apply appropriate permutations to make it so. Now, take the Cartesian product of the first $r-p$ domains of R , and call this new domain A . Take the Cartesian product of the last p domains of R , and call this B . Take the Cartesian product of the last $s-p$ domains of S and call this C .

We can treat R as if it were a binary relation on the domains A, B . Similarly, we can treat S as if it were a binary relation on the domains B, C . The notions of linear and cyclic 3-join are now directly applicable. A similar approach can be taken with the linear and cyclic n -joins of n relations of assorted degrees.

2.1.4 Composition

The reader is probably familiar with the notion of composition applied to functions. We shall discuss a generalization of that concept and apply it first to binary relations. Our definitions of composition and composability are based very directly on the definitions of join and joinability given above.

Suppose we are given two relations R, S . T is a *composition* of R with S if there exists a join U of R with S such that $T = \pi_{13}(U)$. Thus, two relations are composable if and only if they are joinable. However, the existence of more than one join of R with S does not imply the existence of more than one composition of R with S .

Corresponding to the natural join of R with S is the *natural composition*⁹ of R with S defined by

$$R \cdot S = \pi_{13}(R * S).$$

Taking the relations R, S from Figure 5, their natural composition is exhibited in Figure 10 and another composition is exhibited in Figure 11 (derived from the join exhibited in Figure 7).

When two or more joins exist, the number of distinct compositions may be as few as one or as many as the number of distinct joins. Figure 12 shows an example of two relations which have several joins but only one composition. Note that the ambiguity of point c is lost in composing R with S , because of unambiguous associations made via the points a, b, d, e .

Extension of composition to pairs of relations which are not necessarily binary (and which may be of different degrees) follows the same pattern as extension of pairwise joining to such relations.

A lack of understanding of relational composition has led several systems designers into what may be called the *connection trap*. This trap may be described in terms of the following example. Suppose each supplier description is linked by pointers to the descriptions of each part supplied by that supplier, and each part description is similarly linked to the descriptions of each project which uses that part. A conclusion is now drawn which is, in general, erroneous: namely that, if all possible paths are followed from a given supplier via the parts he supplies to the projects using those parts, one will obtain a valid set of all projects supplied by that supplier. Such a conclusion is correct only in the very special case that the target relation between projects and suppliers is, in fact, the natural composition of the other two relations—and we must normally add the phrase “for all time,” because this is usually implied in claims concerning path-following techniques.

2.1.5 Restriction

A subset of a relation is a relation. One way in which a relation S may act on a relation R to generate a subset of R is through the operation *restriction* of R by S . This operation is a generalization of the restriction of a function to a subset of its domain, and is defined as follows.

Let L, M be equal-length lists of indices such that $L = i_1, i_2, \dots, i_k, M = j_1, j_2, \dots, j_k$ where $k \leq \text{degree of } R$ and $k \leq \text{degree of } S$. Then the L, M restriction of R by S denoted $R_L \mid_M S$ is the maximal subset R' of R such that

$$\pi_L(R') = \pi_M(S).$$

The operation is defined only if equality is applicable between elements of $\pi_{i_h}(R)$ on the one hand and $\pi_{j_h}(S)$ on the other for all $h = 1, 2, \dots, k$. The three relations R, S, R' of Figure 13 satisfy the equation $R' = R_{(2,3)} \mid_{(1,2)} S$.

We are now in a position to consider various applications of these operations on relations.

2.2 Redundancy

Redundancy in the named set of relations must be distinguished from redundancy in the stored set of representations. We are primarily concerned here with the former. To begin with, we need a precise notion of derivability for relations.

⁸A function is a binary relation, which is one-one or many-one, but not one-many.

⁹Other writers tend to ignore compositions other than the natural one, and accordingly refer to this particular composition as *the* composition—see, for example, Kelley’s “General Topology.”

R	(s	p	j)	S	(p	j)	R'	(s	p	j)
1	a	A		a	A		1	a	A	
2	a	A		c	B		2	a	A	
2	a	B		b	B		2	b	B	
2	b	A								
2	b	B								

FIG. 13. Example of restriction

Suppose θ is a collection of operations on relations and each operation has the property that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation R is θ -derivable from a set S of relations if there exists a sequence of operations from the collection θ which, for all time, yields R from members of S . The phrase “for all time” is present, because we are dealing with time-varying relations, and our interest is in derivability which holds over a significant period of time. For the named set of relationships in noninferential systems, it appears that an adequate collection θ_1 contains the following operations: projection, natural join, tie, and restriction. Permutation is irrelevant and natural composition need not be included, because it is obtainable by taking a natural join and then a projection. For the stored set of representations, an adequate collection θ_2 of operations would include permutation and additional operations concerned with subsetting and merging relations, and ordering and connecting their elements.

2.2.1 Strong Redundancy

A set of relations is *strongly redundant* if it contains at least one relation that possesses a projection which is derivable from other projections of relations in the set. The following two examples are intended to explain why strong redundancy is defined this way, and to demonstrate its practical use. In the first example the collection of relations consists of just the following relation:

employee(serial#, name, manager#, managername)

with *serial#* as the primary key and *manager#* as a foreign key. Let us denote the active domain by Δ_t , and suppose that

$$\Delta_t(\text{manager\#}) \subset \Delta_t(\text{serial\#})$$

and

$$\Delta_t(\text{managername}) \subset \Delta_t(\text{name})$$

for all time t . In this case the redundancy is obvious: the domain *managername* is unnecessary. To see that it is a strong redundancy as defined above, we observe that

$$\pi_{34}(\text{employee}) = \pi_{12}(\text{employee})_1 \mid_1 \pi_3(\text{employee}).$$

In the second example the collection of relations includes a relation S describing suppliers with primary key $s\#$, a relation D describing departments with primary key $d\#$, a relation J describing projects with primary key $j\#$, and the following relations:

$$P(s\#, d\#, \dots), Q(s\#, j\#, \dots), R(d\#, j\#, \dots),$$

where in each case \dots denotes domains other than $s\#$, $d\#$, $j\#$. Let us suppose the following condition C is known to hold independent of time: supplier s supplies department d (relation P) if and only if supplier s supplies some project j (relation Q) to which d is assigned (relation R). Then, we can write the equation

$$\pi_{12}(P) = \pi_{12}(Q) \cdot \pi_{21}(R)$$

and thereby exhibit a strong redundancy.

An important reason for the existence of strong redundancies in the named set of relationships is user convenience. A particular case of this is the retention of semiobsolete relationships in the named set so that old programs that refer to them by name can continue to run correctly. Knowledge of the existence of strong redundancies in the named set enables a system or data base administrator greater freedom in the selection of stored representations to cope more efficiently with current traffic. If the strong redundancies in the named set are directly reflected in strong redundancies in the stored set (or if other strong redundancies are introduced into the stored set), then, generally speaking, extra storage space and update time are consumed with a potential drop in query time for some queries and in load on the central processing units.

2.2.2 Weak Redundancy

A second type of redundancy may exist. In contrast to strong redundancy it is not characterized by an equation. A collection of relations is *weakly redundant* if it contains a relation that has a projection which is not derivable from other members but is at all times a projection of *some* join of other projections of relations in the collection.

We can exhibit a weak redundancy by taking the second example (cited above) for a strong redundancy, and assuming now that condition C does not hold at all times.

The relations $\pi_{12}(P), \pi_{12}(Q), \pi_{12}(R)$ are complex¹⁰ relations with the possibility of points of ambiguity occurring from time to time in the potential joining of any two. Under these circumstances, none of them is derivable from the other two. However, constraints do exist between them, since each is a projection of some cyclic join of the three of them. One of the weak redundancies can be characterized by the statement: for all time, $\pi_{12}(P)$ is *some* composition of $\pi_{12}(Q)$ with $\pi_{21}(R)$. The composition in question might be the natural one at some instant and a nonnatural one at another instant.

Generally speaking, weak redundancies are inherent in the logical needs of the community of users. They are not removable by the system or data base administrator. If they appear at all, they appear in both the named set and the stored set of representations.

2.3 Consistency

Whenever the named set of relations is redundant in either sense, we shall associate with that set a collection of statements which define all of the redundancies which hold independent of time between the member relations. If the information system lacks—and it most probably will—detailed semantic information about each named relation, it cannot deduce the redundancies applicable to the named set. It might, over a period of time, make attempts to induce the redundancies, but such attempts would be fallible.

Given a collection C of time-varying relations, an associated set Z of constraint statements and an instantaneous value V for C , we shall call the state (C, Z, V) *consistent* or *inconsistent* according as V does or does not satisfy Z . For example, given stored relations R, S, T together with the constraint statement “ $\pi_{12}(T)$ is a composition of $\pi_{12}(R)$ with $\pi_{12}(S)$ ”, we may check from time to time that the values stored for R, S, T satisfy this constraint. An algorithm

¹⁰A binary relation is complex if neither it nor its converse is a function.

for making this check would examine the first two columns of each of R, S, T (in whatever way they are represented in the system) and determine whether

1. $\pi_1(T) = \pi_1(R)$,
2. $\pi_2(T) = \pi_2(S)$,
3. for every element pair (a, c) in the relation $\pi_{12}(T)$ there is an element b such that (a, b) is in $\pi_{12}(R)$ and (b, c) is in $\pi_{12}(S)$.

There are practical problems (which we shall not discuss here) in taking an instantaneous snapshot of a collection of relations, some of which may be very large and highly variable.

It is important to note that consistency as defined above is a property of the instantaneous state of a data bank, and is independent of how that state came about. Thus, in particular, there is no distinction made on the basis of whether a user generated an inconsistency due to an act of omission or an act of commission. Examination of a simple example will show the reasonableness of this (possibly unconventional) approach to consistency.

Suppose the named set C includes the relations S, J, D, P, Q, R of the example in Section 2.2 and that P, Q, R possess either the strong or weak redundancies described therein (in the particular case now under consideration, it does not matter which kind of redundancy occurs). Further, suppose that at some time t the data bank state is consistent and contains no project j such that supplier 2 supplies project j and j is assigned to department 5. Accordingly, there is no element $(2, 5)$ in $\pi_{12}(P)$. Now, a user introduces the element $(2, 5)$ into $\pi_{12}(P)$ by inserting some appropriate element into P . The data bank state is now inconsistent. The inconsistency could have arisen from an act of omission, if the input $(2, 5)$ is correct, and there does exist a project j such that supplier 2 supplies j and j is assigned to department 5. In this case, it is very likely that the user intends in the near future to insert elements into Q and R which will have the effect of introducing $(2, j)$ into $\pi_{12}(Q)$ and $(5, j)$ in $\pi_{12}(R)$. On the other hand, the input $(2, 5)$ might have been faulty. It could be the case that the user intended to insert some other element into P —an element whose insertion would transform a consistent state into a consistent state. The point is that the system will normally have no way of resolving this question without interrogating its environment (perhaps the user who created the inconsistency).

There are, of course, several possible ways in which a system can detect inconsistencies and respond to them. In one approach the system checks for possible inconsistency whenever an insertion, deletion, or key update occurs. Naturally, such checking will slow these operations down. If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently. Inputs causing the inconsistencies which remain in the data bank state at checking time can be tracked down if the system maintains a journal of all state-changing transactions. This latter approach would certainly be superior if few non-transitory inconsistencies occurred.

2.4 Summary

In Section 1 a relational model of data is proposed as a basis for protecting users of formatted data systems from the

potentially disruptive changes in data representation caused by growth in the data bank and changes in traffic. A normal form for the time-varying collection of relationships is introduced.

In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

Many questions are raised and left unanswered. For example, only a few of the more important properties of the data sublanguage in Section 1.4 are mentioned. Neither the purely linguistic details of such a language nor the implementation problems are discussed. Nevertheless, the material presented should be adequate for experienced systems programmers to visualize several approaches. It is also hoped that this paper can contribute to greater precision in work on formatted data systems.

Acknowledgment. It was C. T. Davies of IBM Poughkeepsie who convinced the author of the need for data independence in future information systems. The author wishes to thank him and also F. P. Palermo, C. P. Wang, E. B. Altman, and M. E. Senko of the IBM San Jose Research Laboratory for helpful discussions.

3. REFERENCES

- [1] CHILDS, D. L. Feasibility of a set-theoretical data structure—a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co., Amsterdam, p. 162–172.
- [2] LEVEIN, R. E., AND MARON, M. E. A computer system for inference execution and data retrieval. *Comm. ACM* 10, 11 (Nov. 1967), 715–721.
- [3] BACHMAN, C. W. Software for random access processing. *Datamation* (Apr. 1965), 36–41.
- [4] MCGEE, W. C. Generalized file processing. In *Annual Review in Automatic Programming* 5, 13, Pergamon Press, New York, 1969, pp. 77–149.
- [5] Information Management System/360, Application Description Manual H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.
- [6] GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.
- [7] BLEIER, R. E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 41–49.
- [8] IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Phoenix, Ariz., CPB 1093B, Feb. 1968.
- [9] CHURCH, A. *An Introduction to Mathematical Logic I*. Princeton U. Press, Princeton, N.J., 1956.
- [10] FELDMAN, J. A., AND ROVNER, P. D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.

Second International NoCOUG SQL Challenge

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.

(continued on page 22)

TOP PERFORMANCE

Change Data Capture and Data Integration for Oracle®, MySQL®, IBM® DB2, SQL Server®, Informix®, Sybase® and others.

www.hitsw.com



A BackOffice Associates, LLC Company

DBMoto®

DBMoto® is the best independent data replication and data integration solution available today for real-time Change Data Capture across multiple databases. No consulting required — data replication functions, mapping and verification are all available through prebuilt graphical interfaces!

- Change Data Capture for fast, non-intrusive data updates
- Open APIs to integrate DBMoto into your architecture
- Supports Oracle databases and Oracle® RAC
- Excellent for migrating data between different versions of Oracle
- Multi-server synchronization
- DBMoto Verifier™ for data comparisons before and after data replication
- Easy-to-use — wizards, intuitive graphical interfaces, customizable functions
- Small footprint — extreme performance!

"As a non-DBA expert, I am thrilled to have a software package that was easy to understand and deploy, and that manages my data synchronization for me."

— **TriActive**

"We had a very specific replication requirement to transform the data before it was entered into the MySQL databases. Traditionally this process required manual input to set up and initiate. With DBMoto it's as simple as pressing a button."

— **Gullivers Travel Associates**

"We evaluated virtually every other data replication product on the market and DBMoto came up as the clear winner on price, functionality and usability."

— **EFCO Corporation**

Visit www.hitsw.com/NoCOUG for more information, including:

- **Free comparison sheet "Why DBMoto for Oracle?"**
- **Read case study "TriActive Supports SaaS Business Model using DBMoto for Data Synchronization"**
- **Download FREE trial of DBMoto (with full support)**

DATABASES SUPPORTED:

Oracle (v.9 and above) · MySQL · Microsoft SQL Server · IBM DB2 (all versions) · Netezza · Informix · Sybase ASE · SQL Anywhere · Ingres · PostgreSQL

T +1.408.345.4001 www.hitsw.com info@hitsw.com

Copyright © 2011 Hit Software, Inc., A BackOffice Associates, LLC Company. All rights reserved. Hit Software®, Hit Software logo, and DBMoto® are either trademarks or registered trademarks of Hit Software and BackOffice Associates, LLC in the United States and other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. All other products, company names, brand names, trademarks and logos are the property of their respective companies.



(continued from page 20)

```
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');
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);
```

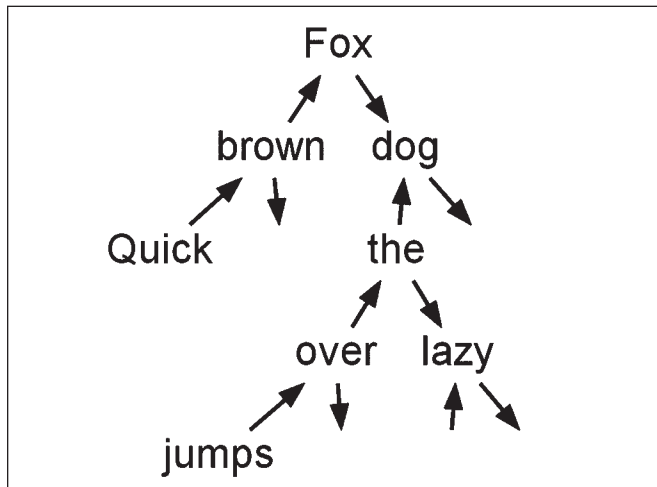


Figure 1.

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.

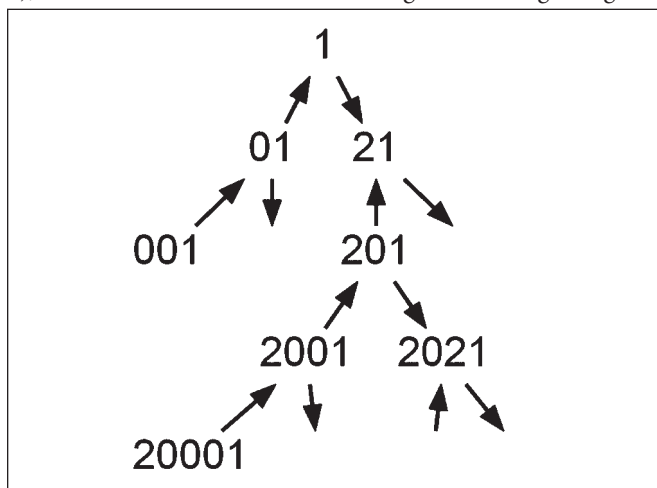


Figure 2.

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.

```
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. ▲

ORACLE®

ORACLE PRESS

YOUR DESTINATION FOR ORACLE AND JAVA EXPERTISE

Written by leading technology professionals, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Oracle products and technologies available.



Effective MySQL: Optimizing SQL Statements

Ronald Bradford

Improve database and application performance



Java Programming

Poornachandra Sarang

Learn advanced skills from a Java expert



Oracle Business Process Management Suite 11g Handbook

Manoj Das, Manas Deb, and Mark Wilkins
Implement successful business process management projects



Oracle Hyperion Financial Management Tips And Techniques

Peter John Fugere, Jr.
Consolidate financial data and maintain a scalable compliance framework



E-BOOKS: Go to OraclePressBooks.com for Adobe Digital Editions (PDF) or Amazon for Kindle Editions.

Join the Oracle Press Community: www.OraclePressBooks.com



Follow us @OraclePress

Many Thanks to Our Sponsors

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

Long-term event sponsorship:

CHEVRON

ORACLE CORP.

Thank you! Year 2011 Gold Vendors:

- Confio Software
- Database Specialists
- Delphix
- Embarcadero Technologies
- GridIron Systems
- Quest Software
- Quilogy Services

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



TREASURER'S REPORT

Naren Nagtode, *Treasurer*

Beginning Balance

July 1, 2011

\$ 66,555.16

Revenue

Membership Dues	1,140.00
Meeting Fees	410.00
Vendor Receipts	4,500.00
Advertising Fee	—
Training Day	4,200.00
Sponsorship	—
Interest	4.26
Paypal balance	—
Total Revenue	\$ 10,254.26

Expenses

Regional Meeting	9,793.62
Journal	3,803.91
Membership	48.46
Administration	20.00
Website	—
Board Meeting	22.56
Marketing	—
Insurance	—
Vendors	14.80
Tax	—
Training Day	123.60
IOUG Registration	—
Miscellaneous	(190.00)
Total Expenses	\$ 13,636.95

Ending Balance

September 30, 2011

\$ 63,172.47

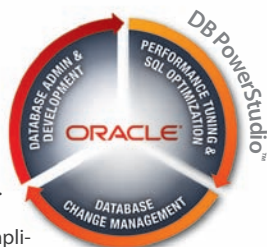


Oracle Database Administration, Development, and Performance Tuning... Only Faster.

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

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

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



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

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

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

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

embarcadero

Ignite8

Turn Up the Heat on Oracle Performance

Database Monitoring and Performance Analysis

Only Ignite delivers a complete picture of database performance from an end user perspective. It analyzes response time, queries, sessions, and server resources to show both historical and real-time performance.

Go to www.Confio.com and try Ignite 8 today!

CONFIO
Confio Software, Boulder Colorado.

QUILOGY SERVICES

FROM ASPECT

Oracle Professional Consulting and Training Services

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

ORACLE APPROVED EDUCATION CENTER

ORACLE EDUCATION RESELLER

www.quilogyservices.com
education@aspect.com
866.784.5649

Aspect

Boost Oracle Performance up to 10x

Bring *tier 0* performance to your IT environment!

- > ZERO changes to existing servers, storage and applications
- > Cost saving by leveraging current infrastructure



GridIron TurboCharger™

Market's First SAN Application Acceleration Appliance

See Customer Success Stories at www.GridIronSystems.com
Schedule a Briefing by Emailing Sales@GridIronSystems.com

NoCOUG Roll of Honor

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
1. Ann Seki	T	T	T	T	T																
2. Dan Lamb	VP																				
3. Gary Falsken	S	VP	P	PE																	
4. James Moore	JE	JE	JE	JE	IL		MAL	T													
5. John De Voy	P	P		DI		P	WM	WM													
6. Kathy Kamibayashi	OL																				
7. Merrilee Nohr	DM	DM	DM	DM	DM	DM	DM	DM	DM	DM	DM										
8. Dale Benjamin		DS	DS																		
9. Katy Walneuski		OL																			
10. Mike Oelkers		S	S	S	S																
11. Rita Palanov		DI	VP	P	P	PE	DM	P	PE												
12. Janice Ogi			IL																		
13. Judy Boyle			DI																		
14. Tina Kraus				IL																	
15. Gary Johnson				VP																	
16. Karen Kirsten				VC																	
17. Arnie Weinstein				VP		VC															
18. Barry Geraghty				DI	DP	VP	DP	DP	IL												
19. John Pon				DP	JE	IL	MAL														
20. Mark Warren				T	T																
21. Corinna Taruc-Burk				VP	P	IL	IL	VP	VP	P	DM	DM	DM	DM	DM	DM	DM	DM	DM	DM	DM
22. Joel Rosingana				S																	
23. Kari Esler				IL	VC	VC	MAL														
24. Loren Gruner					S																
25. Breana Benton Bartholomew						MAL	MAL														
26. Jerry Hughes						MAL															
27. John Rommel						DP															
28. Rich Matheson					S	WM															
29. Karen Bukowski					MAL	VP	P	P	IL	MAL	MAL	MAL									
30. Vilin Roufchaie					VP	P															
31. Walter Schenk																					
32. Hans Yip																					
33. Marshall Stevenson																					
34. Richard Flores																					
35. Cecile Lavoie																					
36. Judy Lyman																					
37. Ken Leonard																					
38. Lisa Loper																					
39. Roger Schrag																					
40. Darrin Swan																					
41. Ganesh Sankar																					
42. Hamid Minoui																					
43. Vadim Barilko																					
44. Colette Lamm																					
45. Eric Buskirk																					
46. Mike DeVito																					
47. Eric Hutchinson																					
48. Jen Hong																					
49. Laurie Robbins																					
50. Les Kopari																					
51. Randy Samberg																					
52. Diane Lee																					
53. Iggy Fernandez																					
54. Naren Nagtode																					
55. Hanan Hit																					
56. Claudia Zeiler																					
57. Gwen Shapira																					
58. Noelle Stimely																					
59. Jenny Lin																					
60. Omar Anwar																					
61. Scott Alexander																					
62. Dave Abercrombie																					



Joel Rosingana

-LEGEND-

- DCP Director of Conference Programming
- DI Director of Improvements
- DM Director of Membership
- DP Director of Publicity
- DS Director of SIGs
- IL IOUG Liaison
- JE Journal Editor
- MAL Member at Large
- OL Oracle Liaison
- P President
- PE President Emeritus
- S Secretary
- SC Speaker Coordinator
- ST Secretary/Treasurer
- T Treasurer
- TC Training Coordinator
- TL Track Leader
- VC Vendor Coordinator
- VP Vice President
- WM Webmaster

Database Specialists: DBA Pro Service



DBA PRO BENEFITS

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

CUSTOMIZABLE SERVICE PLANS FOR ORACLE SYSTEMS

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

DBA Pro's mix and match service components

Access to experienced senior Oracle expertise when you need it

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

24 x 7 availability with guaranteed response time

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

Daily review and recommendations for database care

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

Monthly review and report

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

Proactive maintenance

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

Onsite and offsite flexibility

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



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

NoCOUG

P.O. Box 3282
Danville, CA 94526

NoCOUG Conference #100

Sponsored by Quest Software—*Simplicity at Work*

November 9, 2011—Computer History Museum, Mountain View

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

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

8:00 a.m.–9:00	Registration and Continental Breakfast—Refreshments served
9:00–9:30	Welcome: Iggy Fernandez, NoCOUG president
9:30–10:30	Keynote: <i>Coding Therapy for Database Professionals</i> —Steven Feuerstein, Quest Software
10:30–11:00	Break and Book Signing
11:00–11:50	Parallel Sessions #1 Hahn: <i>Four Things Every DBA and Developer Should Know About Oracle</i> —Andrew Zitelli, Thales Raytheon Boole: <i>Making the Most of PL/SQL Error Management Features</i> —Steven Feuerstein, Quest Software Lovelace: <i>Oracle Database Cloud Service</i> —Richard Greenwald, Oracle
11:50–12:40 p.m.	Lunch
12:40–1:30	Parallel Sessions #2 Hahn: <i>The History of Oracle Performance Analysis</i> —Craig Shallahamer, OraPub Boole: <i>Real-Time SQL Monitoring</i> —Greg Rahn, Oracle Lovelace: <i>The Case for Manual SQL Tuning</i> —Dan Tow, Singing SQL
1:30–2:00	Break and Refreshments
2:00–2:50	Parallel Sessions #3 Hahn: <i>Oracle Database Appliance</i> —David Crawford, Cloud Creek Systems Boole: <i>Resolving Buffer Busy Waits</i> —Craig Shallahamer, OraPub Lovelace: <i>Best Practices for Managing Optimizer Statistics</i> —Maria Colgan, Oracle
2:50–3:10	Raffle
3:10–4:00	Parallel Sessions #4 Hahn: <i>Under the Hood of Oracle ASM: Fault Tolerance</i> —Alex Gorbachev, Pythian Boole: <i>Visual SQL Tuning By Example</i> —Kyle Hailey, Delphix Lovelace: <i>Oracle NoSQL Database</i> —Marie-Anne Neimat, Oracle
4:00–5:00	Exhibition

RSVP online at <http://www.nocoug.org/rsvp.html>