



Official Publication of the Northern California Oracle Users Group

NoCOUG

J O U R N A L

Vol. 30, No. 1 • FEBRUARY 2016

\$15

The Rise and Fall of the NoSQL Empire

The Internet of "Things"

With Shyam Varan Nath.
See page 4.

Brian's Notes

Oracle Database 12c New Features.
See page 6.

The Rise and Fall of the NoSQL Empire

With comments by Chris Date.
See page 12.

Much more inside . . .



Toad enables connections with a community of millions

The Toad World community? It's experience. It's knowledge. It's tips, tricks, and real-time help straight from the experts. And it's inside the latest version of Toad for Oracle. Realize the power of connected intelligence.

Learn more at software.dell.com/toadconnected



Software

Professionals at Work

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

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

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

And, finally, David Gonzalez at Layton Printing Services deftly brings the *Journal* to life on an offset printer.

This is the 117th issue of the *NoCOUG Journal*. Enjoy! ▲

—NoCOUG Journal Editor

Table of Contents

Interview.....	4	ADVERTISERS	
Brian's Notes.....	6	Dell Software	2
NoSQL Corner.....	12	Axxana	11
SQL Corner.....	19	HGST.....	17
Tips and Tricks	22	Delphix	23
Treasurer's Report.....	24	OraPub	25
Board Message	26	Database Specialists	27
		SolarWinds.....	28

Publication Notices and Submission Format

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

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

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

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

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

2016 NoCOUG Board

President

Iggy Fernandez

President Emeritus

Hanan Hit

President Emeritus

Naren Nagtode

Vice President

Jeff Mahe

Secretary/Treasurer

Sri Rajan

Membership Director

Stephen Van Linge (Board Associate)

Conference Director

Sai Devabhaktuni

Vendor Coordinator

Omar Anwar

Webmaster

Jimmy Brock

IOUG Liaison

Kyle Hailey (Board Associate)

Training Director

Vacant Position

Social Media Director

Vacant Position

Marketing Director

Vacant Position

Members at Large

Eric Hutchinson
Kamran Rassouli
Linda Yang

Board Advisor

Tim Gorman

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 Internet of “Things”

by Shyam Varan Nath



Shyam Varan Nath

Who comes up with these names? The Internet that we know has always been an Internet of things, including web servers, application servers, database servers, file servers, name servers, routers, printers, etc. What difference does it really make if other sorts of “things” are plugged into the Internet?

The term “Internet of Things” (IoT) was coined by British tech-preneur Kevin Ashton in 1999. He worked at Auto-ID labs using RFID. The keyword here is “Things” and it generally excludes computers, web servers, application servers, database servers, file servers, name servers, etc. “Things” refers to items that were traditionally not “smart,” e.g., jet engines, cars, refrigerators, and even wearables like shoes, ski jackets, etc. Once these “things” are connected to a network, the information collected can be used to make the assets perform better, proactively predict the likelihood of failures, etc. This was not possible in the pre-IoT era.

IoT allows traditional product companies to offer new hybrids of products and services. For example, instead of selling tires, by embedding sensors in the tires and vehicles, tire manufacturers can sell Tires as a Service to, say, a trucking fleet, on a pay-as-you-go basis. Such companies can also provide optimal routing and fuel efficiency advisory services.

Likewise, on the business continuity side, unplanned downtime can be reduced by using IoT in industrial machine settings. The data collected from the sensors can be used to apply industrial data science. This leads to predictive maintenance and increases the lead time of alerts before actual problems arise.

What are the enabling technologies for this Internet of Things? Are there standards?

The enabling technologies include sensors on the devices, machine-to-machine (M2M) communications, device configuration and security, connectivity to the cloud or a data center, data ingestion and storage, analytics to drive the business applications, and a user interface (UI) for user consumption.

There are several standards and related groups emerging, such as the Industrial Internet Consortium (IIC—including Oracle and GE), the Open Interconnect Consortium (OIC), and the AllSeen Alliance. These groups are trying to increase the interoperability of the various solution providers for real-world use cases leveraging IoT. The industry has realized that IoT is a team sport. With so many moving parts, the ecosystem approach to solving complex IoT problems seems to be the norm here.

Which technology companies will benefit from this Internet of Things?

The companies focusing on IoT platforms, the enabling technologies, and the IoT business applications that are outcomes based will benefit the most. This will include technology companies as well as companies who make the “things” and want to add intelligence to these devices. Two kinds of solution providers will likely emerge. One will be software companies that have the technology building blocks and will partner with customers to acquire the domain knowledge and build the domain-specific IoT applications. An example is Oracle’s CMRO, which is targeted toward the aviation industry, where Oracle partnered with select customers to develop the solution. The second will be the manufacturers of products such as airplanes or jet engines that will take the lead in developing IoT applications, leveraging a diverse set of technologies, both for internal consumption and for selling to the industry vertically. Both categories will heavily leverage IoT platforms to rapidly build and deploy the applications. Increasingly, such platforms will be cloud based. Public clouds will provide the ubiquitous connectivity to ingest the data from the “things” that are often outside the corporate firewalls, such as machines operating on the customer’s premises or in the public domain (such as cars operating on the roads).

What is the state of the union? Where is it in the hype cycle? When will it get to the “plateau of productivity”?

“‘Things’ refers to items that were traditionally not ‘smart,’ e.g., jet engines, cars, refrigerators, and even wearables like shoes, ski jackets, etc. Once these ‘things’ are connected to a network, the information collected can be used to make the assets perform better, proactively predict the likelihood of failures, etc.”

“Connecting airplanes, airports, baggage, and baggage-handling equipment with sensors to provide continuous location and status information—and then consuming the sensor data for business benefits—would be an example of digitizing the airline industry.”

I will talk about the state of the IoT at the BIWA Summit, January 26 to January 28, which is open to all NoCOUG members. IoT is on the top of the mind of every CIO. Every business is trying to become a digital business. Let's take the example of airlines: the SITA Airlines IT 2015 survey indicates that “86% of airlines expect the IoT to provide clear benefits over the next three years.” Connecting airplanes, airports, baggage, and baggage-handling equipment with sensors to provide continuous location and status information—and then consuming the sensor data for business benefits—would be an example of digitizing the airline industry. Such information can be used to track the airline baggage end to end and provide the information to passengers with a mobile app.

Progressive Casualty Insurance Company's trademarked Snapshot device is an example of digitizing the auto insurance industry, which can lead to behavior-based pricing or even mileage-based pricing in the auto insurance industry. Wearable and connected health trackers on humans can lead to the digitization of the healthcare industry.

IoT has been on the peak of the hype cycle for two years, and 2016 will be the pivotal year to see the business outcomes that have been promised to end users. The plateau of productivity of IoT is expected in the next three to five years in some industries. On the consumer side, such as wearables and smart homes, we may see rapid adoption of IoT, although ROI may be smaller. On the industrial IoT front, where large-scale benefits can be derived by making the infrastructure more efficient, it may take five to ten years to see reasonable levels of adoption.

What role do business intelligence (BI) and big data play in the Internet of Things?

Big data plays a key role on the cloud side of the IoT architecture. The data from IoT devices varies in nature, and big data analytics is the key to deriving business value. For example, the big data from jet engines or aircraft, which runs into hundreds of GBs per flight, can be analyzed to reduce unscheduled downtime and to schedule proactive maintenance events. Today it costs the airline industry \$5,000 to \$6,000 per delayed flight, and early morning delays have a cascading impact. Cancellations costs upwards of \$25,000 per flight. BI and data visualization provide visibility to the end user and fertile ground for the business SMEs to spot the trends and patterns in IoT data. This in turn allows data scientists to convert these trends and patterns into analytics that can be run as scheduled activity or as real-time processing on IoT data.

What about security? What will stop foreign (or domestic) actors from hacking into cars and planes?

The consortiums, standard bodies, and early adopters are all focused on the security needs of IoT. In the initial phases, you will see more IoT applications that only read data from machines and store it securely, rather than controlling the machine directly. The human in the loop will initially ensure that big data

analytics provide additional value over traditional human-operated workflows. This will reduce the chances of hacking into planes and cars. In addition, industrial firewalls are being developed to ensure that only known types of data transmission are permitted when industrial machines are connected to the network.

What are companies like GE doing in this space?

GE Digital is actively working on an Industrial Internet platform called Predix. It is also actively working with the industry ecosystem under the IIC umbrella. Industrial IoT platforms like Predix allow the capture of data at the “edge” (e.g., a connected car) and transfer it securely to the cloud, where the data is in-

“Big data plays a key role on the cloud side of the IoT architecture. The data from IoT devices varies in nature, and big data analytics is the key to deriving business value.”

gested, organized, and analyzed. This provides the necessary data to drive the IoT business applications. Such business applications, on top of the Industrial IoT platform, can be developed by customers, partners, and the ecosystem at large, leading to an app marketplace—a Platform as a Service (PaaS) approach—where the SaaS is accelerated by the power of the ecosystem, and the marketplace provides joint monetization. This is no different from the method taken by Amazon Web Services (AWS) or Salesforce on the typical IT applications side, such as ERP/CRM.

Thanks for spending so much time with us today. Can you recommend any books or other resources? How can we follow this topic? People? Blogs?

The Industrial Internet website has a wealth of industrial use cases and testbeds for IoT scenarios where companies like GE and Oracle are working together. ▲

Shyam V. Nath is an industrial IoT architect at GE Digital. He has 25 years of industry experience in areas like IoT, big data analytics, BI, and data warehousing. He worked for IBM, Deloitte, Oracle, and Halliburton prior to GE. He is a regular speaker at Oracle Openworld, IOUG Collaborate, BIWA Summit, and user groups like NoCOUG and NYOUG.

Oracle Database 12c New Features

Book Notes by Brian Hitchcock

Details

Author: Robert G. Freeman

ISBN-13: 978-0071799317

Pages: 480

Year of Publication: 2013

Edition: 1

List Price: \$35

Publisher: Oracle Press



Summary

This book is a great way to come up to speed on the many—and there are many—new features and enhancements that are part of the Oracle 12c Database. Anyone working with, developing for, or supporting an Oracle database will benefit from reading this book. If you have limited time, I strongly suggest you read Chapter 3, which covers the 12c multitenant database. I plan to take the OCP DBA Upgrade Exam for 12c at some point, and reading this book is part of my preparation. I read this book online at Safari Books Online. I suggest you find out if you can access Safari Books Online through your employer, so you can get access at no cost to you.

Foreword

Tom Kyte describes 12c with the Oracle Multitenant database as the first major architectural change since version 6. I don't recall version 6, but if we are to believe Mr. Kyte, 12c is a really big deal!

Introduction

The author tells us that this book was not written to help readers prepare for the OCP exam. I am reading several 12c books as part of my preparation for the OCP exam, but I wasn't expecting this book to be an exam preparation book. I bring this up because Oracle Press offers a separate book that is specifically designed to help prepare for the OCP 12c exam, so I'm not sure why the author was concerned about this. Also, the point is made about what exactly qualifies as a “new” feature. This isn't as simple as I would have thought. There are features that are new in 12c that have been backported to 11gR2—so they are new, but they're not unique to 12c.

Chapter 1: Getting Started with Oracle Database 12c

This chapter shows us how to download and install 12c. During the discussion of where to get the software, we discover that raw partitions are no longer supported. This takes me back. I remember learning about raw and “cooked” partitions many years ago. I'm curious as to why raw devices are no longer sup-

ported. As we are preparing to install 12c, we learn that the most common cause of upgrade issues is not reading the upgrade guide. I'm not sure how the author knows this, and I wonder if this issue is unique to upgrades. For fresh installs, what is the most common cause of problems? While discussing the minimum hardware requirements for 12c, we are told that 1 GB of memory is required, but using anything less than 8 GB is “probably bordering on the insane.” I'm assuming the author feels this is a bad thing. You don't have to be insane to do my job but it really isn't a bad thing.

Part of the installation process is checking that the operating system has all the needed packages, and for this we learn about the Oracle Database preinstall RPM. I had never heard of this, and—no surprise—we are advised to review the install guide for the operating system. Also new to me was the `/etc/oracle-release` command used to check the Linux version installed. The installation process is covered in detail, including many screenshots from the Oracle Universal Installer (OUI). Not surprisingly, one of the OUI screens has a new option to create the new database as a Multitenant Container Database, which is, in my opinion, the primary new feature of 12c. Note that this option is selected by default, but you can create an old-fashioned, non-multitenant database if you want to. Also new in 12c is the demise of Database Control, replaced by Database Express (more on this later). It turns out that the way you use OUI to deinstall a database has changed with 12c.

Database Express is covered next. It has a smaller footprint—and we are told it is easier to manage—but, according to the author, it has “limited functionality” compared to Database Control. You can't start or stop the database from Database Express. There are sections covering how to navigate Database Express for configuration, storage, security, and performance. Screenshots are included so you can see what the new screens look like. However, we are told that what we really want to use is Oracle Database 12c Cloud Control. For all of its faults, the performance of Database Control is described as exceptional.

This chapter ends with a description of 12c as the biggest change and most significant release in a long time.

Chapter 2: Upgrading and Migrating to Oracle Database 12c

This chapter covers the basics of upgrading or migrating an existing Oracle database to 12c. The more we are told that 12c is such a big change, the more we need to be concerned that the upgrade process may be longer, more complicated, and riskier than previous new versions.

While reading this chapter, I would have been helped by diagrams. Perhaps all the steps will be obvious to others, but I got confused sometimes about which server was the source and/or destination and where the various Oracle homes were.

A few diagrams would have clarified where everything is located for an existing non-12c database and the new 12c database.

Before getting into specifics, we are given a definition of “upgrading” versus “migrating.” When we upgrade, we don’t change the data in the database; we only upgrade the data dictionary. When we migrate, we move an existing database to new hardware, a new operating system, or a new character set. I hadn’t thought about it before, but this makes sense. Next is an explanation of why the upgrade process is complicated, and we are warned to “tread carefully.” This means that the specific upgrade path will depend on lots of things, including the database version, features used, and security.

Given that this is a book about upgrading Oracle, perhaps it can be assumed that we know why we need to upgrade, but the reasons given here are worth thinking about. Two basic reasons are given. First, support for any version gets more expensive as that version ages, and we will reach a point where we can’t get new bug fixes. Note that 11gR2 Premier (i.e., the best) support ended in January of 2015. Second is security. The advice is that upgrading ensures that your database has “most” of the current database patches and security-related features. I like the use of the word “most.” Despite what you may have been told, it’s an endless process, and you never have all the current patches. With respect to the endless need to upgrade your Oracle database, I’m sure you recall, while nursing something with tequila in it, that 8i database that ran flawlessly and never gave you any problems. I can help you get more tequila, but preventing the need to upgrade, not so much.

Oracle 12c has new features for upgrading, including a pre-upgrade information tool, support for parallel processing during the upgrade, restartable database upgrade assistant, improved reporting, and installation of the XML database. All of these are covered in detail. We are advised to carefully prepare for the upgrade, and many specific suggestions are provided. My personal favorite is that you must have a test system that duplicates your production system. As with so much advice about Oracle, I agree completely, and I’ve never seen this done in the wild. No matter how much the risk of a bad upgrade will cost, there is never enough money to fund a test system that really reflects production.

Different upgrade approaches are discussed, such as direct and indirect, using the Upgrade Assistant, and several others. Note that you can only upgrade directly to 12c if your existing database is at or above specific release levels for 10g and 11g.

Using the Database Upgrade Assistant is covered in detail with lots of screenshots, as well as upgrading manually where you choose not to use the Upgrade Assistant. There is a separate discussion of using Data Pump to upgrade, which is useful when you’re upgrading across platforms that have different endian (I just like using that word) byte formats. The upgrade process can be rolled back, but only under limited circumstances. Note that you can roll back from 12c to fewer versions than you can upgrade from; just like your relationship status on Facebook, it’s complicated.

Chapter 3: Oracle Multitenant

This chapter is, in my opinion, the one that matters. If you only read one chapter of this book, this is the one. I’m sure there are many users who are really interested in all the other new

features of 12c, but let’s face it, multitenant is the big news—the new feature that can really change how you use the database. We are told that multitenant is one of the biggest changes to the Oracle database ever, and that it was designed to address many problems, including provisioning, patching, consolidation, and improved resource utilization.

At this point, I learned some things that surprised me. Oracle multitenant is optional and requires additional license fees. The intense marketing effort surrounding 12c had me convinced that the new version was basically multitenant, so I’m surprised that it isn’t part of the basic database. I assumed that Oracle wanted everyone to move to multitenant and therefore it wouldn’t be an extra-cost feature, but it appears I was wrong. I was further surprised when my very unscientific poll of fellow attendees at the latest NoCOUG Conference revealed many who were not going to use multitenant because of the additional license cost. One person told me that their company could hire a team of DBAs to manage multiple databases for less than the cost of multitenant. I will have to contemplate this, along with another adult beverage, but we must move on.

Since multitenant is optional, this means that a database created in 12c will be, by default, the same database architecture we are all familiar with, which means no containers and no pluggable databases. Having said that, the next section tells us that multitenant will be the only database architecture at some point in the future, and that there are no license issues for using multitenant with a single pluggable database. We will all be moving to multitenant at some point, so we need to learn how it works. We are told that the non-12c database architecture is officially referred to as the “legacy” architecture. I feel old.

The 12c multitenant architecture, at its simplest, consists of container and pluggable databases. When you create a multitenant database, you create a multitenant container database (CDB), which can then hold one or more pluggable databases (PDBs). Each PDB appears to be an independent database. Once you are connected to a PDB, things are almost identical to the architecture we are all familiar with now. When I first read through this chapter, I was thinking that the pluggable databases are not container databases. But when I got to the part where we want to take a non-12c, non-pluggable (i.e., the legacy architecture) database and plug it into a 12c container database, I became confused. I had to back up and have a “do-over.” It is explicitly stated that “a PDB is also called a container.” In 12c, for multitenant, you create one multitenant container database called the “root” that will hold (contain) all the pluggable databases. But these pluggable databases are still called “container” databases, if only to distinguish them from a non-12c database. To create a multitenant database, you use a new option for the create database command: enable pluggable database.

This is a long chapter, and most of it can’t be covered here. I learned many things. Once you create a database in 12c, whether it is a container DB or not, you can’t change to the other kind. However, you can run both CDBs and non-CDBs on the same database server. There is only one set of online redo logs, in the CDB, and all the pluggable databases use this same set. I can see some issues coming from this, but we are assured this isn’t a problem. The benefits of moving to multitenant are presented, and it all revolves around how it will be easier to manage multiple databases in a container database on consolidated hardware.

One of the stated advantages of multitenant is the ease with which we can copy (clone) a PDB in one CDB and move it to another CDB where we just plug it in. For example, we could copy a PDB in the test CDB, move it to the production CDB, and plug it in. I agree that this could be a great feature, but something that I wanted to learn is not discussed: how long does it take to do this? Yes, mileage may vary, but once you copy the database files of the PDB from one CDB to another, does it take a lot longer? Is there some initialization process that has to run after this? I think this issue should have been covered; this is a 12c feature that could really change, for the better, how things are done for many customers.

This chapter has lots more to offer. Common users are created in the CDB and can access all the PDBs. These common users must have usernames that start with c## (or C##), which I find strange. It makes me think I'm looking at code or comments, perhaps. You can use sqlplus / to connect, but that will only work for the CDB; then you have to switch to one of the PDBs. You have to use a connect string to connect directly to a PDB. There are many new data dictionary views for the CDB and the PDBs as well as new commands to see where you are (CDB or one of the PDBs). There are sections that discuss how to use multitenant to consolidate systems and the new tools provided to deal with resource management among the pluggable databases.

Chapter 4: Oracle Grid Infrastructure

This chapter covers the changes to RAC for 12c. I don't support many RAC systems so I'm not qualified to comment on the usefulness of these new features. We start with a diagram showing how RAC works in 11g, with two database instances accessing a single database. Each of the nodes is the same and runs the same software. In 12c, we have Flex Clusters, which can have two types of nodes: hub and leaf. The hub node has a full stack of the cluster software and looks like a pre-12c RAC node, while the leaf node has a lightweight software stack and may not have direct access to the shared storage. Leaf nodes are optional, but you must have a hub node. This is good when your processing requires lots of CPU but not lots of data. The example given is spatial data processing, where most of the processing could be run on the leaf nodes.

The steps to convert to flex clusters are covered, and once you convert you can't go back. Along with flex clusters we also have Flex ASM.

Before 12c, ASM managed disk space, but the database instance processes did the actual reading and writing of data. The ASM and database instances had to run on the same machine. With Flex ASM, the ASM instance can be on a separate server from the database, and you don't need to have an ASM instance for every database node. For RAC clusters with many nodes, this can save significant resources. The process of installing and configuring Flex ASM is shown, using the Oracle Universal Installer (OUI).

There are new features for ASM itself, including shared password files and the ability to rebalance multiple disk groups at the same time.

We also see some ACFS enhancements, but I didn't know ACFS even existed. This is the Oracle Automatic Storage Manager Cluster File System, which allows administrators to support ASM like other file systems.

Chapter 5: Backup, Recovery, and Data Guard New Features

Most of these new features are for RMAN and the rest are for Data Guard. Incremental and multisection backups allow data-file backups to be broken into smaller pieces and do this in parallel. Other improvements allow recovery of a production database from a standby database. Cross-platform backup and recovery is improved as well. New options for the RMAN commands include "to platform" and "from platform," and several examples of how to use them are provided.

RMAN in 12c can recover a specific table or a specific partition of a table to a point in time. These new features require many prerequisites and these are covered in detail. For example, the database must have been in ARCHIVELOG mode when it was backed up and up to the point you want to recover to. You can also use the new remap option of the recover table command to restore a table to a different name. There are a lot of new options to learn about, and we see an example where all the steps and SQL are covered.

It will not be a surprise that the RMAN new features include support for container and pluggable databases. There is a new SYSBACKUP privilege for users that only need to handle backup tasks, and you can execute SQL commands directly from the RMAN prompt.

There is a new SYSDG privilege for administering the Data Guard standby database. We also have the Far Sync instance, which supports processing of redo for a standby database that is far away from the primary. When configured for zero data loss, the latency caused by long distances can cause the primary database to hang while waiting for the redo logs to be moved and applied to the standby. The Far Sync instance is a separate instance that collects redo logs generated on the primary. Once the Far Sync instance has confirmed it has the redo log, the primary instance can continue, and the Far Sync instance will ensure that the redo is applied at the standby.

Cascading Redo Transport destinations allows a standby database to forward redo logs to other standby databases. Fast-Sync Mode is another feature to help with latency issues. The primary will wait for the redo log to be received at the standby but will not wait for that redo log to be written to disk.

Finally, Data Guard supports physical or logical standby databases for a container database, and Active Data Guard allows DML operations and sequences in the standby database.

Chapter 6: Oracle Database 12c SQL and PL/SQL New Features

We have new features for DML, DDL, and PL/SQL. For DML, we start with row pattern matching. Here the author makes a comment that reflects how I feel about a lot of the new features: "Very often we find great uses for new features once we figure out what they are useful for." In this case, we can now do pattern matching of columns across a set of rows using the new `pattern_matching_clause` option. We are told this is useful for identifying patterns in large data sets. The discussion covers what patterns are and gives a detailed example of this new functionality. Next we have enhancements to the Oracle native left join syntax. It seems these changes will correct some of the missing pieces of the syntax and will make the syntax more versatile. When we want to return a specific number of rows, we use the top-n query syntax. In 12c we can limit the result set to a specific number of rows or a percentage of rows.

The DDL section of this chapter covers those new DDL features that are not covered in other chapters. We have increased size limits for various datatypes, identity columns, cascading truncate statements, invisible columns, default values for columns based on sequences or when NULLS are inserted, and new subquery features. Each of these has a separate section describing the enhancements and most have an example of using the new SQL.

For PL/SQL new features, 12c gives us more options on the privileges a function will be executed with, using white lists to improve security, data types across the SQL interface, changes to invoker rights for program units, and a bunch of miscellaneous stuff. The details of each of these are discussed and examples are presented.

Chapter 7: Partitioning Enhancements

This chapter is broken into seven sections, the first covering moving partitions online. In 12c you can move partitions and subpartitions online. This is useful for moving older partitions to lower-cost (i.e., lower-performance) storage as part of an overall plan to implement a data lifecycle plan. You can also compress partitions as you move them, and you can move datafiles online as well.

For interval-reference partitioning, some existing restrictions have been removed. You can now use interval partitioning on the parent table as part of reference partitioning, and an example is shown using this.

Cascade functionality—an enhancement to the truncate table command—is provided in 12c. Using the cascade option for the truncate partition and the exchange partition commands, truncate table can truncate both the parent and child tables. There does need to be a constraint defined using the on delete cascade syntax.

Partition maintenance is better in 12c because you can now apply the alter partition add, truncate, drop, split, and merge commands to multiple partitions. The SQL is discussed for an example that uses the options to add, split, and merge partitions.

Partial indexes remove the restriction that an index had to span all the partitions of a partitioned table. While there was a minimal version of this feature in 11g, it was difficult to use. You can have a mix of local and global indexes at the partition level. You can identify which partitions should be indexed. This saves space since indexes don't have to cover all partitions, and it can improve performance by reducing the processing needed to create and maintain indexes.

Global index maintenance is needed when global indexes become stale. This happens when you drop or truncate a partition and when you use the modify partition index off command, any of which will leave orphaned entries in the affected indexes. New to 12c is the `dbms_part.cleanup_gidx` procedure, which identifies and cleans up any such indexes.

Knowing when global statistics need to be gathered is important because computing statistics takes significant resources and should only be done when needed. 12c provides the `incremental_staleness` setting for a table, which sets a tolerance level for stale statistics. I found the discussion of all the options for `incremental_staleness` to be amusing. Why don't we just take all the data and pack it into some new Tupperware and eliminate this staleness issue altogether? Even better, dump all the data and buy Oreos and Spam—they never go bad!

Chapter 8: Business Intelligence and Data Warehousing

Here we find several new things, starting with advanced analytics. 12c provides new algorithms for Expectation Maximization (EM), Singular Value Decomposition (SVD), and Generalized Linear Models (GLM). I don't know what any of that means, but it sure sounds like really good stuff to mention at your next staff meeting! There is a description of what each of these is used for. EM is good for determining target clusters for an advertising campaign, SVD does large matrix stuff, and GLM reduces the number of predictors used by a model. Sounds impressive!

For online analytical processing (OLAP), 12c provides performance improvements for cubes by using shared cursors, reduced memory requirements, and reduced I/O. Managing cube statistics is also easier, as they are part of AWR, ASH, and ADDM.

Next is Information Lifecycle Management (ILM), which assigns value to data based on business rules. The author tells us that older data is less valuable and should be moved to lower-cost storage. In general I would agree, but the issue is complicated. The value of old social media data when you are trying to optimize a marketing campaign for the next week is very different from the value of data from a medical trial where data from decades ago may be highly relevant. The argument that ILM isn't necessary because "storage is cheap" is discussed. While it may be that storage is relatively inexpensive and the trend is to lower cost, that doesn't directly relate to performance. Whatever the cost of storage in general, there will still be more expensive, faster storage and slower, cheaper storage. ILM seeks to optimize whatever you spend on storage to get the most important data on the fastest storage.

ILM improvements in 12c include In-Database Archiving (IDA), Temporal Validity, and Automatic Data Optimization (ADO). IDA marks data as archived or active, and the archived data becomes invisible to the application and is compressed. Temporal Validity adds hidden columns to a table to track when the data is valid and when it is not. ADO generates a heat map of activity at the segment and row levels. Based on this heat map, ADO will move data to storage that has been identified as faster or slower.

Temporal History (TH) is similar to the 11g feature Total Recall (insert Terminator joke here), which was a Flashback Data Archive (FDA). This is flashback, configured for a specified retention time for a portion of a tablespace or for one or more tables.

There are many built-in performance enhancements relevant to BI and DW, too many to be discussed in detail in the book, but they fall into two categories: Materialized Views and Parallelism. For MV we have out-of-place and synchronized refresh, both of which support faster refreshes. Parallelism has better statement queuing and an improved automatic degree of parallelism (Auto DOP).

Chapter 9: Security New Features

Since I recently read the Oracle Database 12c Security book, I was curious about which of the many new features would make the cut for this chapter. Auditing is discussed first. The Unified Audit Trail (UAT) is new, and the SGA has two additional named queues that are used one at a time, switching when one is full. When the first queue fills, the new background process GEN0 (General Task Execution Process) flushes the records in memory

to the AUDSYS table. Note that UAT is not enabled by default, which I find surprising after all the talk about how important auditing is.

UAT supports separation of duties using two new roles, AUDIT_ADMIN and AUDIT_VIEWER. UAT is part of both container and pluggable databases. An example is given showing all the SQL needed to set up UAT. Extended Audit Information (EAI) provides auditing for Fine Grained Auditing (FGA), Data Pump, RMAN, Oracle Label Security, Database Vault, and Real Application Security (RAS).

Privilege Analysis (PA) is provided to show which users or roles have excessive privileges. All too often, many database users are given full access just because it is easy and saves time. PA works by creating a capture process that will examine all or some of the database, executing that process and generating the results. A lengthy example shows all the SQL and steps needed.

Data Redaction is new in 12c and is different from Data Masking. Data Masking would be used to change the data as you copy a production environment to a test environment. Data Redaction changes only the way the data is displayed and not the actual data itself. There are multiple options available to determine exactly how the data is displayed. For example, using the PARTIAL redaction method and working on a column that contains a last name, you can specify the start and end position and the redaction character. Using this you could display only the last four digits of the actual credit card number. Note that this does not protect sensitive data from authorized users; it only affects how the data is displayed on screen or in printed reports. A lengthy scenario is presented showing how to set up and use Data Masking.

The next section covers miscellaneous security improvements like the new encryption options. The RESOURCE role no longer has the UNLIMITED TABLESPACE privilege, which is a real improvement. The SELECT ANY DICTIONARY privilege no longer has access to several security-related data dictionary tables. When users connect to the database, the banner will show the time of their last login. While we can agree that these changes have been needed for some time from a security point of view, they could also cause problems for your existing scripts and batch jobs. You need to make sure you are aware of these changes and are prepared to deal with the problems that may occur. It probably isn't practical to tell you to review all the scripts and batch jobs in your environment, but you do need to realize that these changes could cause trouble.

The last section of this chapter is titled "Mixing and Matching Security Features." We learn that choosing the right combination of the available features to meet all the needs of a company is a daunting task. Just keeping up with all the acronyms is daunting!

Chapter 10: Oracle Database 12c Manageability New Features

We learn about several ways 12c makes database management easier. If you are trying to execute some administrative operation, you may get an error if the database is busy, perhaps due to a locking issue. With 12c Online Operations, the database will handle these issues for you. This section covers the enhanced online DDL capabilities for various DDL commands, lock timeout for online redefinitions, moving a datafile, and table/partition redefinition.

For database monitoring and administration, 12c provides some new features and some changes to existing features.

Enterprise Manager Database Express is new, replacing Database Control, and is easier to manage because it doesn't rely on any middleware components. Note that some new features, such as container databases, and some existing features, such as ASM, are not present in EM Database Express. This doesn't seem like an improvement to me, but the author tells us we really should be using EM Cloud Control 12c, so it doesn't matter.

Real-time database operations monitoring has been improved to monitor parallel SQL execution, to monitor any SQL that consumes more than five seconds of CPU or I/O. It will also monitor composite operations, which include scripts and batch jobs.

The last section of this chapter is "Miscellaneous Manageability Features," which covers a lot of new stuff. A new database parameter PGA_AGGREGATE_LIMIT can be set to provide a hard limit on the size of the PGA for the instance.

New administrator's privileges, SYSBACKUP, SYSDG, and SYSKM—used for administering backups, Data Guard, and data encryption respectively—have been created to help us move away from every DBA having SYSDBA privilege.

The last section covers changes that affect many of the features, new and existing, that have already been discussed in other chapters. This section is really long. Among the many areas covered, I like Queryable Patch Inventory. When we install Oracle, the inventory is created and we use the opatch utility to see what patches have been installed. With 12c, we have the dbms_qopatch package so we can query the inventory. I can see this being useful in the real world. Among the other topics are changes to Data Masking; Database Replay; Direct NFS (dNFS); database cloning; Advanced Network Compression; large network buffers; multiprocess, multithreaded Oracle; and new scheduler script jobs.

Chapter 11: Oracle Database 12c Performance New Features

With 12c, we have new features to improve database performance, SQL tuning, and overall database monitoring. These are grouped into those that are statistics related and those that are optimizer related.

Statistics-related new features start with automatic column group detection, where statistics are collected on two or more columns to help with cardinality estimates. The improvement is that you don't have to guess which columns should be used. Concurrent statistics gathering collects statistics on multiple tables in a schema or a table with multiple partitions. Incremental statistics uses the statistics from those partitions unaffected by a partition maintenance operation to reduce the time needed to compute new statistics (which has been enhanced to support nonpartitioned tables), and my favorite, incremental staleness, is discussed again. After a bulk load operation—which includes create table as select, insert into using the APPEND hint, and parallel inserts—12c automatically collects statistics. Global temporary tables can have their own statistics for each session using the new database parameter GLOBAL_TEMP_TABLE_STATS. When you are preparing to generate statistics, you can choose to use the new reporting mode, which shows which objects will be analyzed but doesn't actually generate the statistics.

The section on optimizer-related new features includes coverage of adaptive query optimization where the optimizer can make run-time adjustments to execution plans to help make them better. This section covers how adaptive plans get config-

ured and created—and how you can view them—and includes a detailed example of all the steps.

Next is adaptive statistics, a term for many new and improved features that help the optimizer generate better statistics for complex queries. The discussion covers automatic reoptimization, which means statistics are gathered as SQL is executed that are compared to those used to generate the execution plan. This process of optimization is iterative: the execution plan will change as the SQL is executed over and over. Another incredibly detailed example follows.

Histograms have issues when the data has many distinct values. In 12c, we have improved histograms that can look at popular, unpopular, and (my favorite) almost popular data values. I went through high school being almost popular. We have top-frequency histograms, where the most unpopular values are ignored to limit the number of buckets, and hybrid histograms, which combine features of height-based and frequency-based histograms.

This chapter is very long and at the end, the author tells us it might be the best chapter for the developer, the report writer, and the architect, as well as the DBA.

Chapter 12: Other Oracle Database 12c New Features

This is the last chapter and it covers all sorts of odds and ends. The new features and enhancements covered are Data Pump export and import, SQL*Loader, External Tables, Log Miner, ADR DDL and debug logs, SecureFiles, Database R, Hadoop, MapReduce, CloneDB, and the SQL Translation Framework. There is a separate section for each of these; some are very long (Data Pump and SQL*Loader) and some are very short (External Tables and Log Miner). There should be something in this chapter for everyone.

Appendix A: Deprecated and Desupported Features in Oracle Database 12c

This appendix starts with an explanation of what “deprecation” and “desupport” really mean. Then we have sections for deprecated features, desupported features, and deprecated views as well as parameters. I didn’t think I’d find much of interest in this appendix, but I was wrong. Specifically, I found several things that surprised me. First, if you are using Oracle Streams, you need to migrate to Oracle GoldenGate, because Oracle Streams has been deprecated. Second, the same story goes for Oracle Advanced Replication: you need to migrate to Oracle GoldenGate. Support for raw devices has been eliminated altogether.

Instead of listing all the deprecated database parameters, we are given the SQL statement to generate the list.

Appendix B: New Parameters and Views in Oracle Database 12c

In the section covering new parameters I found the first and only mention of the new database parameter `ENABLE_PLUGGABLE_DATABASE`. If you want to create a container database and one or more pluggable databases, you must have this database parameter set. I was surprised to find the only mention of this parameter in Appendix B; I’d expect that it would have been described in Chapter 3, when creating the CDB and PDBs was covered.

The section that covers the new views in 12c contains the

statement, “There are a ton of new views included in Oracle Database 12c,” and based on the listing provided, I quite agree.

Conclusion

This book was well worth my time. I didn’t fully understand many of the new features that are discussed because I don’t regularly use them. For example, I’m not a PL/SQL developer, so I can’t comment on how useful the new 12c features for PL/SQL are. On the other hand, I now have a good feel for all of the major areas where 12c provides new functionality. At the same time, for core DBA—the area where I have the most interest—I would have liked more detail on the 12c multitenant database.

A single volume can’t cover all of the new features in great detail, but this book does a great job of providing as much detail as possible. Even if you aren’t planning to migrate to 12c anytime soon, I recommend that you look at this book, perhaps reading only those chapters that relate to your work, because 12c is a big change and will affect how we all work with the Oracle database in the future. ▲

Brian Hitchcock works for Oracle Corporation where he has been supporting Fusion Middleware since 2013. Before that, he supported Fusion Applications and the Federal OnDemand group. He was with Sun Microsystems for 15 years (before it was acquired by Oracle Corporation) where he supported Oracle databases and Oracle Applications. His contact information and all his book reviews and presentations are available at www.brianhitchcock.net/oracle-dbafmw/. The statements and opinions expressed here are the author’s and do not necessarily represent those of Oracle Corporation.

Copyright © 2016, Brian Hitchcock

DATABASE RECOVERY HAS NEVER BEEN SO SIMPLE!

Axxana’s award winning **Phoenix System** offering unprecedented Data Protection, Cross-application consistency, Storage and Replication agnostic.



info@axxana.com • www.axxana.com

The Rise and Fall of the NoSQL Empire

by Iggy Fernandez



Iggy Fernandez

Editor's Note: This article is a reprint from the February 2015 issue of the NoCOUG Journal. Chris Date provided detailed comments (keyed to the section headings in this article) which were published in the May 2015 issue of the NoCOUG Journal. Date's comments are also reprinted here, as footnotes to the sections of this article.

NoSQL is a “disruptive innovation” in the sense used by Harvard professor Clayton M. Christensen. In *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*, Professor Christensen defines disruptive innovations and explains why it is dangerous to ignore them:

“Generally, disruptive innovations were technologically straightforward, consisting of off-the-shelf components put together in a product architecture that was often simpler than prior approaches. They offered less of what customers in established markets wanted and so could rarely be initially employed there. They offered a different package of attributes valued only in emerging markets remote from, and unimportant to, the mainstream.”

The personal computer was an example of a disruptive innovation. It was initially targeted only at the home computing segment of the market. Established manufacturers of mainframe computers and minicomputers did not see PC technology as a threat to their bottom lines. Eventually, however, PC technology came to dominate the market, and established computer manufacturers such as Digital Equipment Corporation, Prime, Wang, Nixdorf, Apollo, and Silicon Graphics went out of business.

So where lies the dilemma? Professor Christensen explains:

“In every company, every day, every year, people are going into senior management, knocking on the door saying: ‘I got a new product for us.’ And some of those entail making better products that you can sell for higher prices to your best customers. A disruptive innovation generally has to cause you to go after new markets, people who aren't your customers. And the product that you want to sell them is something that is just so much more affordable and simple that your current customers can't buy it. And so the choice that you have to make is: Should we make better products that we can sell for better profits to our best customers. Or maybe we ought to make worse products that none of our customers would buy that would ruin our margins. What should we do? And that really is the dilemma.”

Exactly in the manner that Christensen described, the e-commerce pioneer Amazon created an in-house product called Dynamo in 2007 to meet the performance, scalability, and availability needs of its own e-commerce platform after it concluded that mainstream database management systems were not capable of satisfying those needs. The most notable aspect of Dynamo was the apparent break with the relational model; there was no mention of relations, relational algebra, or SQL.

Dynamo Requirements and Assumptions

Amazon started out by using Oracle Database for its e-commerce platform but later switched to a proprietary database management system called Dynamo that it built in-house. Dynamo is the archetypal NoSQL product; it embodies all the innovations of the NoSQL camp. The Dynamo requirements and assumptions are documented in the paper “Dynamo: Amazon's Highly Available Key-value Store” (<http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf>), published in 2007. Here are some excerpts from that paper:

“Customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.”

“There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.”

“Experience at Amazon has shown that data stores that provide ACID guarantees tend to have poor availability.”

“Dynamo targets applications that operate with weaker consistency (the “C” in ACID) if this results in high availability.”

“... since each service uses its distinct instance of Dynamo, its initial design targets a scale of up to hundreds of storage hosts.”

To paraphrase, Amazon's requirements were extreme performance, extreme scalability, and extreme availability.

Functional Segmentation¹

Amazon's *pivotal* design decision was to break its monolithic enterprise-wide database service into simpler component services, such as a best-seller list service, a shopping cart service, a customer preferences service, a sales rank service, and a product catalog service. This avoided a single point of failure. In an interview for the *NoCOUG Journal*, Amazon's first database administrator, Jeremiah Wilton explains the rationale behind Amazon's approach:

"The best availability in the industry comes from application software that is predicated upon a surprising assumption: The databases upon which the software relies will inevitably fail. The better the software's ability to continue operating in such a situation, the higher the overall service's availability will be. But isn't Oracle unbreakable? At the database level, regardless of the measures taken to improve availability, outages will occur from time to time. An outage may be from a required upgrade or a bug. Knowing this, if you engineer application software to handle this eventuality, then a database outage will have less or no impact on end users. In summary, there are many ways to improve a single database's availability. But the highest availability comes from thoughtful engineering of the entire application architecture." (http://www.nocoug.org/Journal/NoCOUG_Journal_200711.pdf#page=4)

As an example, the shopping cart service should not be affected if the checkout service is unavailable or not performing well.

I said that this was the *pivotal* design decision made by Amazon. I cannot emphasize this enough. If you resist functional segmentation, you are *not* ready for NoSQL. If you miss the point, you will not understand NoSQL.

Note that functional segmentation results in simple hierarchical schemas. Here is an example of a simple hierarchical schema from Ted Codd's 1970 paper on the relational model (http://www.nocoug.org/Journal/NoCOUG_Journal_201111.pdf#page=10). This simple schema stores information about employees, their children, their job histories, and their salary histories.

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

Functional segmentation is the underpinning of NoSQL technology, but it does not present a conflict with the relational model; it is simply a physical database design decision. Each functional segment is usually assigned its own standalone database. The collection of functional segments could be regarded as a single distributed database. However, distributed transactions are forbidden in the NoSQL world. Functional segmentation can therefore result in temporary inconsistencies if, for example, the

shopping cart data is not in the same database as the product catalog, and occasional inconsistencies result. As an Amazon customer, I occasionally leave items in my shopping cart but don't complete a purchase. When I resume shopping, I sometimes get a notification that an item in my shopping chart is no longer in stock or has been repriced. This technique is called "eventual consistency." Randy Shoup, one of the architects of eBay's ecommerce platform, explains:

"At eBay, we allow absolutely no client-side or distributed transactions of any kind—no two-phase commit. In certain well-defined situations, we will combine multiple statements on a single database into a single transactional operation. For the most part, however, individual statements are auto-committed. While this intentional relaxation of orthodox ACID properties does not guarantee immediate consistency everywhere, the reality is that most systems are available the vast majority of the time. Of course, we do employ various techniques to help the system reach eventual consistency: careful ordering of database operations, asynchronous recovery events, and reconciliation or settlement batches. We choose the technique according to the consistency demands of the particular use case." (<http://www.infoq.com/articles/eBay-scalability-best-practices>)

Sharding²

Amazon's next design decision was "sharding" or horizontal partitioning of all the tables in a hierarchical schema. Hash-partitioning is typically used. Each table is partitioned in the same way as the other tables in the schema, and each set of partitions is placed in a separate database referred to as a "shard." The shards are independent of each other; that is, there is no clustering as in Oracle RAC.

Note that the hierarchical schemas that result from functional segmentation are always shardonable; that is, hierarchical schemas are shardonable by definition.

Returning to the example from Ted Codd's 1970 paper on the relational model:

```
employee (man#, name, birthdate) with primary key (man#)
children (man#, childname, birthyear) with primary key (man#, childname)
jobhistory (man#, jobdate, title) with primary key (man#, jobdate)
salaryhistory (man#, jobdate, salarydate, salary) with primary key (man#, jobdate, salarydate)
```

Note that the jobhistory, salaryhistory, and children tables have composite keys. In each case, the leading column of the composite key is the man#. Therefore, all four tables can be partitioned using the man#.

Sharding is an essential component of NoSQL designs but it does not present a conflict with the relational model; it too is simply a physical database design decision. In the relational model, the collection of standalone databases or shards can be logically viewed as a single distributed database.

¹ Chris Date's comments: "Functional segmentation is the underpinning of NoSQL technology, but it does not present a conflict with the relational model; it is simply a physical database design decision." Absolutely correct. I couldn't agree more.

"The collection of functional segments could be regarded as a single distributed database. However, distributed transactions are forbidden in the NoSQL world. Functional segmentation can therefore result in temporary inconsistencies . . ." I agree with the first sentence here. As for distributed transactions and temporary inconsistencies, however, I have a different perspective on what's really going on with consistency in the NoSQL world (and indeed elsewhere), which I'll elaborate on in the next section but one.

² Chris Date's comments: "Sharding is an essential component of NoSQL designs but it does not present a conflict with the relational model; it too is simply a physical database design decision." Absolutely correct. I couldn't agree more.

"In the relational model, the collection of . . . shards can be logically viewed as a single distributed database." Again I basically agree, modulo my reservations (hinted at above) regarding consistency.

Replication and Eventual Consistency³

The Dynamo developers saw that one of the keys to extreme availability was data replication. Multiple copies of the shopping cart are allowed to exist and, if one of the replicas becomes unresponsive, the data can be served by one of the other replicas. However, because of network latencies, the copies may occasionally get out of sync, and the customer may occasionally encounter a stale version of the shopping cart. Once again, this can be handled appropriately by the application tier; the node that falls behind can catch up eventually, or inconsistencies can be detected and resolved at an opportune time, such as at checkout. This technique is called “eventual consistency.”

The inventor of relational theory, Dr. Codd, was acutely aware of the potential overhead of consistency checking. In his 1970 paper, he said: *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 [emphasis added]. If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently.* In other words, the inventor of relational theory would not have found a conflict between his relational model and the “eventual consistency” that is one of the hallmarks of the NoSQL products of today. However, the Dynamo developers imagined a conflict because it quite understandably conflated the relational model with the ACID guarantees of database management systems. However, ACID has *nothing* to do with the relational model per se (although relational theory does come in very handy in defining consistency constraints); pre-relational database management systems such as IMS provided ACID guarantees and so did post-relational object-oriented database management systems.

The tradeoff between consistency and performance is as important in the wired world of today as it was in Dr. Codd’s world. Synchronous replication is rarely used in the relational camp, so we cannot frown at Dynamo for not using it. Application developers in the relational camp are warned about the negative impact of consistency checking, so we cannot frown on Dynamo’s decision to permit temporary inconsistencies between functional segments.

- “Using primary and foreign keys can impact performance. Avoid using them when possible.” (http://docs.oracle.com/cd/E17904_01/core.1111/e10108/adapters.htm#BABCCCIH)
- “For performance reasons, the Oracle BPEL Process Manager, Oracle Mediator, human workflow, Oracle B2B, SOA Infrastructure, and Oracle BPM Suite schemas have no foreign key constraints to enforce integrity.” (http://docs.oracle.com/cd/E23943_01/admin.1111/e10226/soaadmin_partition.htm#CJHCJJI)
- “For database independence, applications typically do not store the primary key-foreign key relationships in the database itself; rather, the relationships are enforced in the application.” (http://docs.oracle.com/cd/E25178_01/fusionapps.1111/e14496/securing.htm#CHDDGFHH)
- “The ETL process commonly verifies that certain constraints are true. For example, it can validate all of the foreign keys in the data coming into the fact table. This means that you can trust it to provide clean data, instead of implementing constraints in the data warehouse.” (http://docs.oracle.com/cd/E24693_01/server.11203/e16579/constra.htm#i1006300)

The False Premise of NoSQL⁴

The final hurdle was extreme performance, and that’s where

³ Chris Date’s comments: I have no problem with the broad intent of this section or with the business requirement the NoSQL developers are aiming at in this connection. As indicated earlier, however, I do have a different perspective on what’s really going on here. I also disagree, somewhat, with the quote from Codd in this section. Let me try to explain:

- First, to say that a database (distributed or otherwise) is consistent merely means, formally speaking, that the database conforms to all stated integrity constraints. Now, it’s crucially important that databases *always* be consistent in this sense; indeed, a database that’s not consistent in this sense, at some particular time, is like a logical system that contains a contradiction. Well, actually, that’s exactly what it is—a logical system with a contradiction. And in a logical system with a contradiction, you can prove anything; for example, you can prove that $1 = 0$. (In fact, you can *prove* that you can prove that $1 = 0$ in such a system!) What this means in database terms is that if the database is inconsistent in the foregoing sense, you can never trust the answers you get to queries (they may be false, they may be true, and you have no way in general of knowing which they are); all bets are off. That’s why consistency in the foregoing sense is crucial.
- But consistency in the foregoing sense isn’t necessarily the same thing as consistency as conventionally understood (consistency as understood outside the world of databases in particular). Suppose there are two items *A* and *B* in the database that, in the real world, we believe should have the same value. They might, for example, both be the selling price for some given commodity, stored twice because replication is being used to improve availability. If *A* and *B* in fact have different values at some given time, we might certainly say, informally, that there’s an inconsistency in the data as stored at that time. But that “inconsistency” is an inconsistency as far as the system is concerned *only if the system has been told that A and B are supposed to be equal*—i.e., only if “ $A = B$ ” has been stated as a formal constraint. If it hasn’t, then (a) the fact that $A \neq B$ at some time doesn’t in itself constitute a consistency violation as far as the system is concerned, and (b) importantly, the system will nowhere rely on an assumption that *A* and *B* are equal.
- Thus, if all we want is for *A* and *B* to be equal “eventually”—i.e., if we’re content for that requirement to be handled in the application layer—all we have to do as far as the database system is concerned is omit any declaration of “ $A = B$ ” as a formal constraint. No problem, and in particular no violation of the relational model.

Now perhaps you can see why I don’t entirely agree with that text of Codd’s. Codd says:

There are, of course, several possible ways in which a system can . . . respond to [inconsistencies]. In one approach the system checks for possible inconsistency whenever an insertion, deletion, or key update occurs. [Incidentally, I don’t know why Codd says “key update” specifically; surely checking should be done on all pertinent updates?] Naturally, such checking will slow these operations down.

Well, the system can only “respond to” those inconsistencies it knows about, or in other words those that correspond to formally declared constraints. For such inconsistencies, it simply *must* do the checking whenever a pertinent update occurs; there’s no alternative, because not to do that checking is to

(continued on page 15)

the Dynamo developers went astray. The Dynamo developers believed that the relational model imposes a “join penalty” and therefore chose to store data as “blobs.” This objection to the relational model is colorfully summarized by the following statement attributed to Esther Dyson, the editor of the Release 1.0 newsletter, *“Using tables to store objects is like driving your car home and then disassembling it to put it in the garage. It can be assembled again in the morning, but one eventually asks whether this is the most efficient way to park a car.”* The statement dates back to 1988 and was much quoted when object-oriented databases were in vogue.

Since the shopping cart is an object, doesn’t disassembling it for storage make subsequent data retrieval and updates inefficient? The belief stems from an unfounded assumption that has found its way into every relational DBMS—that every table should map to physical storage. In reality, the relational model is a logical model and, therefore, it does not concern itself with storage details at all. It would be perfectly legitimate to store the shopping cart in a physical form that resembled a shopping cart while still offering a relational model of the data complete with SQL. In other words, the physical representation could be optimized for the most important use case—retrieving the entire shopping-cart object using its key—without affecting the relational model of the data. It would also be perfectly legitimate to provide a non-relational API for the important use cases. Dr. Codd himself gave conditional blessing to such non-relational APIs in his 1985 *Computerworld* article, “Is Your DBMS Really Relational?,” in which he says, *“If a relational system has a low-level (single-record-at-a-time) language, that low level [should not] be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).”*

The key-blob or “key-value” approach used by Dynamo and successor products would be called “zeroth” normal form in relational terminology. In his 1970 paper, “A Relational Model of Data for Large Shared Data Banks,” Dr. Codd says: *“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.”

In common parlance, a relation with non-simple domains is said to be in “zeroth” normal form or unnormalized. Dr. Codd suggested that unnormalized relations should be normalized for ease of use. Here again is the unnormalized employee relation from Dr. Codd’s paper:

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

The above unnormalized relation can be decomposed into four normalized relations as follows.

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

However, this is *not* to suggest that these normalized relations must necessarily be mapped to individual buckets of physical storage. Dr. Codd differentiated between the stored set, the named set, and the expressible set. In the above example, we have one unnormalized relation and four normalized relations. If we preferred, the unnormalized employee relation could be the only member of the stored set. Alternatively, if we preferred, *all five* relations could be part of the stored set; that is, we could legitimately store redundant representations of the data. However, the common belief blessed by current practice is that the normalized relations should be the only members of the stored set.

Even if the stored set contains only normalized relations, they need not map to different buckets of physical storage. Oracle Database is unique among mainstream database management systems in providing a convenient construct called the “table cluster” that is suitable for hierarchical schemas. In Dr. Codd’s example, employee# would be the cluster key, and rows corresponding to the same cluster key from all four tables could be stored in the same physical block on disk, thus avoiding the join penalty. If the cluster was a “hash cluster,” no indexes would be required for the use case of retrieving records belonging to a single cluster

(continued from page 14)

risk having a database for which all bets are off (see above). What’s more, I don’t agree that such checking will slow the system down. If the user has bothered to declare the constraint, presumably he or she wants it enforced—for otherwise there’s no point in declaring it in the first place. And if the user wants that constraint enforced, and if the system isn’t going to do it (by which I mean do it properly, by which I mean doing the checking on all pertinent updates), then the user is going to have to do it instead. Either way, the checking has to be done. What’s more, I would hope that the system could do the checking more efficiently than the user; thus, I think that, far from the operations being slowed down, they should be speeded up, so long as the system does the right thing and shoulders its responsibility properly.

Back to Fernandez’s paper. Fernandez goes on to say: “ACID has *nothing* to do with the relational model per se.” Well, I agree with the broad sense of this observation, though I do have some reservations regarding ACID in general which it probably isn’t appropriate to air in detail here. Let me just say that the relational model does at least tacitly require the database system never to lose information, and the A, I, and D features of transactions are aimed at satisfying this goal. In that sense, we might at least say those features of ACID are a mechanism for satisfying a certain relational requirement. (I disagree with the C feature, however, for reasons my earlier remarks on consistency should be sufficient at least to suggest. I don’t want to get into further details on this issue here.)

One final small point on this section: Fernandez uses the phrase “post-relational object-oriented database management systems.” Actually, I would say that while OODBMSs might be “postrelational” from a chronological point of view, they’re actually *prerelational* in terms of their database functionality.

key. A demonstration is available at <http://iggyfernandez.wordpress.com/2013/12/30/the-twelve-days-of-nosql-day-six-the-false-premise-of-nosql/>.

Schemaless Design

The final innovation of the NoSQL camp is “schemaless design.” In database management systems of the NoSQL kind, data is stored in “blobs” or documents; the database management system does not police their structure. In mainstream database management systems on the other hand, doctrinal purity requires that the schema be designed *before* data is inserted. Let’s do a thought experiment.

Let’s suppose that we don’t have a schema and that the following facts are known:

- ▶ Iggy Fernandez is an employee with EMPLOYEE_ID=1 and SALARY=\$1000.
- ▶ Mogens Norgaard is a commissioned employee with EMPLOYEE_ID=2, SALARY=€1000, and COMMISSION_PCT=25.
- ▶ Morten Egan is a commissioned employee with EMPLOYEE_ID=3, SALARY=€1000, and unknown COMMISSION_PCT.

Could we ask the following questions and expect to receive correct answers?

- ▶ **Question:** What is the salary of Iggy Fernandez?
Expected answer: \$1000.
- ▶ **Question:** What is the commission percentage of Iggy Fernandez?
Expected answer: Invalid question.
- ▶ **Question:** What is the commission percentage of Mogens Norgaard?
Expected answer: 25%
- ▶ **Question:** What is the commission percentage of Morten Egan?
Expected answer: Unknown.

If we humans can process the above data and correctly answer the above questions, then surely we can program computers to do so.

The above data could be modeled with the following three relations. It is certainly disruptive to suggest that this be done on the fly by the database management system, but it is not outside the realm of possibility.

```
EMPLOYEES
EMPLOYEE_ID NOT NULL NUMBER(6)
EMPLOYEE_NAME VARCHAR2(128)
```

```
UNCOMMISSIONED_EMPLOYEES
EMPLOYEE_ID NOT NULL NUMBER(6)
SALARY NUMBER(8,2)
```

```
COMMISSIONED_EMPLOYEES
EMPLOYEE_ID NOT NULL NUMBER(6)
SALARY NUMBER(8,2)
COMMISSION_PCT NUMBER(2,2)
```

A NoSQL company called Hadapt has already stepped forward with such a feature:

*“While it is true that SQL requires a schema, it is entirely untrue that the user has to define this schema in advance before query processing. **There are many data sets out there, including JSON, XML, and generic key-value data sets that are self-describing — each value is associated with some key that describes what entity attribute this value is associated with [emphasis added].** If these data sets are stored in Hadoop, there is no reason why Hadoop cannot automatically generate a virtual schema against which SQL queries can be issued. And if this is true, users should not be forced to define a schema before using a SQL-on-Hadoop solution — they should be able to effortlessly issue SQL against a schema that was automatically generated for them when data was loaded into Hadoop.”* (<http://hadapt.com/blog/2013/10/28/all-sql-on-hadoop-solutions-are-missing-the-point-of-hadoop/>)

This is not really new ground. Oracle Database provides the ability to convert XML documents into relational tables (http://docs.oracle.com/cd/E11882_01/appdev.112/e23094/xdbo1int.htm#ADXDB0120), though it ought to be possible to view XML data as tables while physically storing it in XML format in order to benefit certain use cases. It should also be possible to redundantly store data in both XML and relational formats in order to benefit other use cases.

In “Extending the Database Relational Model to Capture More Meaning,” Dr. Codd explains how a “formatted database” is created from a collection of facts:

“Suppose we think of a database initially as a set of formulas in first-order predicate logic. Further, each formula has no free variables and is in as atomic a form as possible (e.g. A & B would be replaced by the component formulas A, B). Now suppose that most of the formulas are simple assertions of the form Pab . . . z (where P is a predicate and a, b, . . . , z are constants), and that the number of distinct predicates in the database is few compared with the number of simple assertions. Such a database is usually called formatted, because the major part of it lends itself to rather regular structuring. One obvious way is to factor out the predicate common to a set of simple assertions and then treat the set as an instance of an n-ary relation and the predicate as the name of the relation.”

In other words, a collection of facts can always be organized into a collection of relations.

⁴ Chris Date’s comments: “The belief stems from an unfounded assumption that has found its way into every relational DBMS—that every table should map to physical storage.” I would replace the phrase “every relational DBMS” here by “every mainstream SQL DBMS” (there are certainly exceptions to what Fernandez is claiming here), but overall I agree with most of the paragraph in which this sentence appears.

“It would also be perfectly legitimate to provide a non-relational API [to a relational system] for the important use cases.” Legitimate, yes, but I believe it would be quite unnecessary, as I tried to explain in my contribution to the interview with Hugh Darwen and myself that appeared in a recent issue of the *NoCOUG Journal* (“No! to SQL! No! to NoSQL!,” *NoCOUG Journal* 27, No. 3, August 2013).

“The key-blob . . . approach . . . would be called ‘zeroth’ normal form in relational terminology.” No, it wouldn’t. There’s no such thing as “zeroth normal form” in relational terminology. In fact, there’s no such thing as an unnormalized relation—the very phrase is a contradiction in terms. *All* relations are normalized, in the sense that they’re in at least first normal form. To suggest otherwise is to do the cause of genuine understanding a serious disservice.

NoSQL Taxonomy

NoSQL databases can be classified into the following categories:

- **Key-value stores:** The archetype is Amazon Dynamo, of which DynamoDB is the commercial successor. Key-value stores basically allow applications to “put” and “get” values, but each product has differentiators. For example, DynamoDB supports “tables” (namespaces) while Oracle NoSQL Database offers “major” and “minor” key paths.
- **Document stores:** While key-value stores treat values as uninterpreted strings, document stores allow values to be managed using formats such as JSON (JavaScript Object Notation) that are conceptually similar to XML. This allows key-value pairs to be indexed by any component of the value just as XML data can be indexed in mainstream database management systems.
- **Column-family stores:** Column-family stores allow data associated with a single key to be spread over multiple storage nodes. Each storage node only stores a subset of the data associated with the key, hence the name “column-family.” A key is therefore composed of a “row key” and a “column key.”
- **Graph databases:** Graph databases are non-relational databases that use graph concepts such as nodes and edges to solve certain classes of problems, such as determining the shortest route between two towns on a map. The concepts of functional segmentation, sharding, replication, eventual consistency, and schemaless design do not apply to graph databases, so I will not discuss graph databases.

NoSQL Buyer's Guide⁵

NoSQL products are numerous and rapidly evolving. There is a crying need for a continuously updated encyclopedia of NoSQL products, but none exists. There is a crying need for an independent benchmarking organization, but none exists. My best advice is to do a proof of concept (POC) as well as a PSR (Performance, Scalability, and Reliability) test before committing to using a NoSQL product. Back in 1985, Dr. Codd had words of advice for those who were debating between the new relational products and the established pre-relational products of the time.

*“Any buyer confronted with the decision of which DBMS to acquire should weigh three factors heavily. The first factor is the buyer’s performance requirements, often expressed in terms of the number of transactions that must be executed per second. The average complexity of each transaction is also an important consideration. Only if the performance requirements are extremely severe should buyers rule out present relational DBMS products on this basis. **Even then buyers should design performance tests of their own, rather than rely on vendor-designed tests or vendor-declared strategies** [emphasis added]. The second factor is reduced costs for developing new databases and new application programs . . . The third factor is protecting future investments in application programs by acquiring a DBMS with a solid theoretical foundation . . . In every case, a relational DBMS wins on factors two and three. In many cases, it*

⁵ Chris Date’s comments: Here I’d just like to say that I agree very strongly with the remarks by Codd quoted in this section.

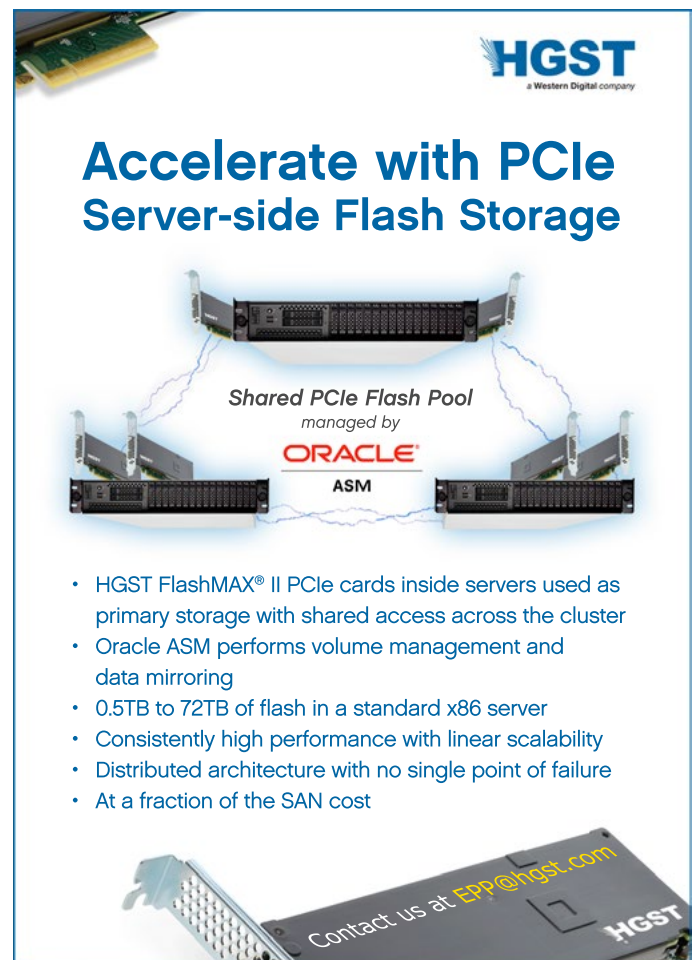
can win on factor one also—in spite of all the myths about performance.” — “An Evaluation Scheme for Database Management Systems that are claimed to be Relational”

The above advice is as solid today as it was in Dr. Codd’s day.

Oracle NoSQL Database

In May 2011, Oracle Corporation published a white paper titled “Debunking the NoSQL Hype,” the final advice being “Go for the tried and true path. Don’t be risking your data on NoSQL databases.” (<https://www.google.com/#q=%2B%22Debunking+the+NoSQL+Hype%22>) However, in September of the same year, Oracle Corporation released Oracle NoSQL Database. Oracle suggested that the NoSQL approach was well-suited for certain use-cases:

“The Oracle NoSQL Database, with its ‘No Single Point of Failure’ architecture, is the right solution when data access is “simple” in nature and application demands exceed the volume or latency capability of traditional data management solutions. For example, click-stream data from high volume web sites, high-throughput event processing and social networking communications all represent application domains that produce extraordinary volumes of simple keyed data. Monitoring online retail behavior, accessing customer profiles, pulling up appropriate customer ads and storing and forwarding real-time communication are examples of domains requiring the ultimate in low-latency access. Highly distributed applications such as real-time sensor aggregation and scalable



HGST
a Western Digital company

Accelerate with PCIe Server-side Flash Storage

Shared PCIe Flash Pool
managed by
ORACLE
ASM

- HGST FlashMAX® II PCIe cards inside servers used as primary storage with shared access across the cluster
- Oracle ASM performs volume management and data mirroring
- 0.5TB to 72TB of flash in a standard x86 server
- Consistently high performance with linear scalability
- Distributed architecture with no single point of failure
- At a fraction of the SAN cost

Contact us at Epp@hgst.com

authentication also represent domains well-suited to Oracle NoSQL Database.” (<http://www.oracle.com/technetwork/products/nosqldb/learnmore/nosql-wp-1436762.pdf>)

Oracle NoSQL Database has two features that distinguish it from other key-value stores: A key is the concatenation of a “major key path” and a “minor key path.” All records with the same “major key path” will be colocated on the same storage node. In addition, Oracle NoSQL provides transactional support for modifying multiple records with the same major key path.

Challenges to NoSQL

There are already proofs that performance, scalability, and reliability can be achieved without abandoning the relational model. For example, ScaleBase provides sharding and replication on top of MySQL storage nodes. Another good example to study is VoltDB, which claims to be the world’s fastest OLTP database (though it has never published an audited TPC benchmark). A counter-example to Amazon is eBay, which arguably has equal scale and equally high-performance, scalability, and reliability requirements. eBay uses performance segmentation, sharding, replication, and eventual consistency but continues to use Oracle (and SQL) to manage local databases. I asked Randy Shoup, one of the architects of the eBay e-commerce platform, why eBay did not abandon Oracle Database, and he answered in one word: “comfort.” Here are links to some of his presentations and articles on the eBay architecture:

- “eBay’s Scaling Odyssey: Growing and Evolving a Large eCommerce Site” (<http://francotravostino.name/papers/eBayScalingOdysseyShoupTravostino.pdf>)
- “The eBay Architecture: Striking a balance between site stability, feature velocity, performance, and cost” (<http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf>)
- “Randy Shoup Discusses the eBay Architecture” (<http://www.infoq.com/interviews/shoup-ebay-architecture>)
- “Randy Shoup on eBay’s Architectural Principles” (<http://www.infoq.com/presentations/shoup-ebay-architectural-principles>)
- “Scalability Best Practices: Lessons from eBay” (<http://www.infoq.com/articles/eBay-scalability-best-practices>)

The latest challenge to NoSQL comes from Google, which recently created a new DBMS called F1 for its business-critical AdWords application. Google implemented a version of Oracle table clusters in order to avoid the join penalty. Here is a quote from Google’s paper “F1: A Distributed SQL Database That Scales”:

“In recent years, conventional wisdom in the engineering community has been that if you need a highly scalable, high-throughput data store, the only viable option is to use a NoSQL key/value store, and to work around the lack of ACID transactional guarantees and the lack of conveniences like secondary

indexes, SQL, and so on. When we sought a replacement for Google’s MySQL data store for the AdWords product, that option was simply not feasible: the complexity of dealing with a non-ACID data store in every part of our business logic would be too great, and there was simply no way our business could function without SQL queries. Instead of going NoSQL, we built F1, a distributed relational database system that combines high availability, the throughput and scalability of NoSQL systems, and the functionality, usability and consistency of traditional relational databases, including ACID transactions and SQL queries. Google’s core AdWords business is now running completely on F1. F1 provides the SQL database functionality that our developers are used to and our business requires. Unlike our MySQL solution, F1 is trivial to scale up by simply adding machines.” (<http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/41344.pdf>)

Summary⁶

The NoSQL camp put performance, scalability, and reliability front and center but lost the opportunity to take the relational model to the next level because—just like the relational camp—it mistakenly believed that normalization dictates physical storage choices, that non-relational APIs are forbidden by the relational model, and that “relational” is synonymous with ACID (Atomicity, Consistency, Isolation, and Durability).

The NoSQL camp created a number of innovations that are disruptive in the sense used by Harvard Business School professor Clayton Christensen: functional segmentation, sharding, replication, eventual consistency, and schemaless design. Since these innovations are compatible with the relational model, they should eventually be absorbed by mainstream database management systems.

Finally, I should point out that there are very good reasons to criticize current NoSQL products, including a lack of standards; primitive feature sets, security, and management tools; unproven claims; and traps for the unwary. MongoDB uses a database-wide lock for reads and writes (<http://docs.mongodb.org/manual/faq/concurrency/#what-type-of-locking-does-mongodb-use>). #nuffsaid. ▲

Copyright © 2015, Iggy Fernandez

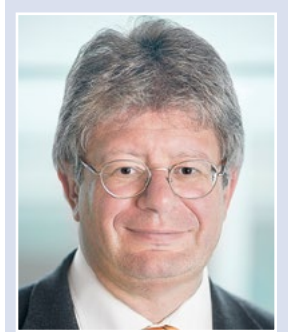
Editor’s Note 1: In November 2014, MongoDB began supporting document-level locking for writes when using the optional Wired Tiger storage engine and collection-level locking when using the default MMAPv1 storage engine (one MongoDB database can contain multiple document collections). Prior to that, MongoDB only supported database-level locking for writes. Prior to August 2012, MongoDB only supported instance-level locking for writes (one MongoDB instance can control multiple databases).

Editor’s Note 2: At OpenWorld 2015, Oracle Corporation made a presentation on native support for sharding in Oracle Database 12c Release 2 (currently in beta).

⁶ Chris Date’s comments: This isn’t a comment on the section of this name in Fernandez’s paper—I’m just using this heading as a convenient place to compliment Fernandez on a most interesting article. My comments in the foregoing, especially the ones of a critical nature, aren’t meant to detract from what I see as a most useful contribution to the ongoing NoSQL debate. Thank you, Iggy!

Raiders of the Data Dictionary III: The Quest for the Missing Index

by Lothar Flatz



Lothar Flatz

In rule number four of his rules defining requirements for a database management system to be considered “relational,” Edgar F. Codd states, “*The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language.*” That is, users must be able to access the database’s structure (catalog) using the same query language that they use to access the database’s data.¹

I have used the exhaustive knowledge that the database holds about its own structure in many ways during my ten years with Oracle Consulting. As you know, there is one category of SQL statements that is among the most difficult to write: statements that check for something that does not exist. Sometimes, however, this is exactly what you need to do when searching for a missing index.

Those in the State of Innocence

It all started with a project for a good customer of mine who was suffering from severe locking issues. The project leader called me in to help. When I arrived, another external consultant was already working on the solution (well, not really—as we will see). He was sent by the software vendor to prove that it was not the fault of the software.

I swiftly spotted that the issue was related to foreign key constraints that were declared, but the foreign key was not indexed. I needed to do some fighting with the other consultant to prove my case. “No, this cannot possibly be the reason; I have never heard of it.” I had to write some queries against v\$lock and pull out the SQL statements that were waiting and locking.

When I produced the evidence to my customer, the case was immediately decided. I told the project leader that I could solve the case in a few minutes by writing a query that would find all non-indexed foreign keys. Using this query I would generate a script that would create the missing indexes. However, I added, at that point we would not yet be done. Since right then only primary key indexes were present, we would also need to create indexes to speed up searches.

The project leader blinked at me and asked, “Can’t you just write another query to find those?” “Gosh!” I said to myself, picturing myself banging my head against the wall, “He really has no clue what he is asking for. Just write another query . . .” Quite often, the technically naïve questions will give you the biggest headache.

If It Could Be Done

I normally find it hard to reject a challenge. When I don’t know how to do a task, I ask myself, “If it *could* be done, how would I do it?” That way I focus on the solution rather than on the difficulties of the task. Following this procedure, I began to forge the queries. By the end of the day I got something working. It was not perfect or pretty (even though you will see the improved version here), but it did the job. Working hard I didn’t realize how the time was passing. By the end of the day I was completely exhausted. I felt more like a vegetable than a human being. Unfortunately, I had promised my wife that we would go out that night. I’m still sorry that I was a rather dull partner that evening.

Some General Advice on Indexing

Before we actually start discussing the queries, here’s a bit of advice: I do not spend a lot of time making my scripts pretty. I just want to get something working as quickly as possible. Please keep this in mind if you want to use them as the basis for developing something better.

When I wrote my scripts I sometimes had to make up my mind to include educated guesses in order to make the scripts work. Feel free to experiment with your own numbers if you like.

Always consider that an index is also an obligation for the database to slow down DML. An index might speed up one query, but it might slow dozens of INSERT, UPDATE, or DELETE statements, so always keep your eye on the big picture.

Keep asking yourself questions like, “Is this query executed often enough?” or “Will other queries benefit from that index too?” or “Can a slight modification in the index structure allow other queries to benefit from that index?”

The Index Is Missing

I started to think about how I normally find missing indexes. I usually pick them up from runtime statistics. I can illustrate the approach with a typical example (Figure 1):

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.81	21157
1	NESTED LOOPS		1	25	1	00:00:00.81	21157
2	NESTED LOOPS		1	25	4	00:00:00.81	21153
* 3	TABLE ACCESS FULL	PATIENT	1	25	4	00:00:00.81	21143
* 4	INDEX UNIQUE SCAN	TMT_INDEX	4	1	4	00:00:00.01	10
* 5	TABLE ACCESS BY INDEX ROWID	TREATMANT	4	1	1	00:00:00.01	4

Predicate Information (identified by operation id):


```

3 - filter((lower("P"."FIRSTNAME") LIKE :1 AND lower("P"."LASTNAME")
      LIKE :2 AND TRUNC(INTERNAL_FUNCTION("P"."DATEOFBIRTH"))=TRUNC(:3)))
4 - access("T"."PATIENT_ID"="P"." PATIENT_ID")
5 - filter("T"."SERID"=:4)

```

Figure 1: Runtime statistics for a simple query

Everything falls into place. We have a full table scan (shown above as “TABLE ACCESS FULL”) on a table where only a few rows get selected. These four rows represent a tiny fraction of the table. We don’t know how big the table is, but by the number of buffers processed we can guess that it is far bigger than four rows.

The full table scan takes the majority of the time to process the query; therefore, it is good to eliminate it.

The next step is to look at the filter columns for operation 3. We can see that we have a LOWER() function applied to the name columns and a TRUNC() function applied to the date column. This is classic. Probably there is an existing index. These columns are obviously search columns; therefore, it is likely that they are indexed already. However, these indexes can’t be used because of the functions applied. The obvious solution is to create a function-based index.

The nice aspect in this example is that the optimizer is estimating about right. Well, the estimate (E-Rows) is 25 rows, but the actual (A-Rows) is four rows. However, even 25 rows are a small fraction of the table and a good reason to use the index. Indeed, the optimizer used the function-based index right after it was created. To improve the estimate we could consider column group statistics.

Generalizing the above example, we can now try to deduce how a query that checks for potential missing indexes should look. Obviously we will search for a full table scan as a starting point. Then we have to check if the full table scan returns a small enough proportion of rows implying a missing index. Obviously “small enough” is a bit of a vague definition, but that’s pretty much all we have. So we have to define a value.

As a general rule, you will you are not likely to be able to fully automate things. Instead of a program that automatically creates indexes, I prefer to write a query that gives background information for a conscious decision. Creating an index needs a bit more consideration than just finding out that an index is missing.

This approach allows me to be a bit more liberal when defining “small enough.” If I wanted automatic index creation I would have probably gone for 2% of the rows in the table. Since the query will just present me with index *candidates* rather than *creating* indexes, I set my limit to 5% of the number of rows in the table.

Normally I am looking at one specific application, which means that I want to limit my search to a specific schema.

I decided to write two versions of the script (Figures 2 and 3). I prefer to use runtime statistics, just like in the example above. However, runtime statistics will only be generated if the parameter statistics_level is set to “all” for all relevant queries.

The impact on performance of setting the statistics_level to “all” is a bit unclear. Some customers might object to changing this parameter. Since I cannot guarantee that runtime statistics will always be available, I will provide a second version of my script. Instead of using actual values, this version uses the optimizer’s estimates. This is certainly less desirable, but it’s better than nothing, don’t you think?

```

SELECT p.sql_id,
       ROUND(ps.output_rows / ps.executions,2) rows_exe,
       t.num_rows,
       t.blocks,
       ROUND(disk_reads/executions,2) disk_exe,
       object_name,
       filter_predicates, p.child_number, p.plan_hash_value
FROM   v$sql_plan p,
       v$sql_plan_statistics ps,
       all_tables t
WHERE  p.operation          = 'TABLE ACCESS'
      AND OPTIONS          = 'FULL'
      AND p.sql_id         = ps.sql_id
      AND p.ID             = ps.operation_id
      AND p.plan_hash_value = ps.plan_hash_value
      AND p.child_number   = ps.child_number
      AND object_name       = t.table_name
      AND ps.executions     > 0
      AND ps.output_rows   < t.num_rows / 100 * 5 -- 5% of the rows
      AND t.blocks         > 10 -- is it worthwhile to have an index at all?
choose your own number
AND filter_predicates IS NOT NULL
AND cr_buffer_gets     > 0 -- some sanity check
AND object_name        IN
  (SELECT table_name FROM dba_tables WHERE owner='&myschema'
   )
ORDER BY disk_reads DESC, cr_buffer_gets DESC

```

Figure 2: Finding a missing index using runtime statistics

```

SELECT v.sql_id,
       executions,
       v.rows_processed / v.executions,
       t.num_rows,
       t.blocks,
       disk_reads/executions,
       object_name,
       filter_predicates
FROM   v$sql_plan vp,
       v$sql v,
       all_tables t
WHERE  operation          = 'TABLE ACCESS'
      AND v.hash_value    = vp.hash_value
      AND v.address        = vp.address
      AND OPTIONS          = 'FULL'
      AND object_name      = t.table_name
      AND v.executions     > 0
      AND vp.cardinality   <= t.num_rows / 100 * 5 -- 5% of the rows
      AND filter_predicates IS NOT NULL -- is it worthwhile to have an index at all?
      AND t.blocks         > 10
choose your own number
AND buffer_gets          > 0 -- some sanity check
AND object_name          IN
  (SELECT table_name FROM dba_tables WHERE owner='&myschema'
   )
ORDER BY disk_reads, buffer_gets

```

Figure 3: Finding a missing index using optimizer estimates

An Existing Index Can Be Improved

It is certainly possible that a missing index does not result in a full table scan. In such a case, the execution plan could start with an operation on another table and join in the table where the index is missing. Or, we could have an index, but it is not as good as it could be because filter criteria are missing.

In either case the application of good filter criteria will result in a drop in the number of rows.

As you can see in Figure 4, the execution plan starts with an INDEX RANGE SCAN in step 3 that returns 4577 rows. In step 2 during the filtering operation, the actual number of rows is only 10. That is quite a high factor: 457:1!

	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads
	0	SELECT STATEMENT		1	1	1	00:00:15.42	3540	3262
	1	SORT AGGREGATE		1	1	1	00:00:15.42	3540	3262
*	2	TABLE ACCESS BY INDEX ROWID	MM_SALES_DELIVERIES	1	52695	10	00:00:15.42	3540	3262
*	3	INDEX RANGE SCAN	MM_SALE_DEL_FK4	1	17M	4577	00:00:00.16	11	9

Predicate Information (identified by operation id):

```

2 - filter(("MM_FROM_LOCATION_ID"=:B3 AND NVL("MM_DISTRIBUTION_STATUS",'H')<>'U' AND
      NVL("MM_DIRECT_DEL_STATUS",'H')<>'D' AND NVL("MM_DISTRIBUTION_DEL_STATUS",'H')<>'D' AND
      NVL("MM_OUTBOUND_DEL_STATUS",'H')<>'D' AND NVL("MM_OUTBOUND_STATUS",'H')<>'U'))
3 - access("MM_ITEM_ID"=:B2 AND "MM_WAREHOUSE_ID"=:B4 AND "MM_FIRM_ID"=:B5)An Investigation behind Segment Statistics

```

Figure 4: Example of missing filter fields in an existing index

The number of rows drops so drastically because of the extra filter that is applied in operation 2. So, what difference does it make if the filter is applied in operation 2 or operation 3? It makes a whole lot of difference!

To do the index range scan on the index in step 3, only 11 buffers were needed. To retrieve and filter the rows from the table in step 2, 3540 buffers were utilized. This is because (a) the index is much smaller than the table, and (b) the index is clustered.

If we add the filter columns from step 2 to the end of the MM_SALE_DEL_FK index, the index will still be small, but the additional filter will eliminate all but ten rows that want to retrieve from the table.

I systematically tested which of the filter conditions in operation 2 were good filters and added just those columns to the newly built index. By compressing the index I could achieve a reduction in the number of buffers accessed in the index scan (Figure 5).

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1		1	00:00:00.02	12	4
1	SORT AGGREGATE		1	1	1	00:00:00.02	12	4
* 2	TABLE ACCESS BY INDEX ROWID	MM_SALES_DELIVERIES	1	52695	10	00:00:00.02	12	4
* 3	INDEX RANGE SCAN	PK_SALE_DEL_03	1	52695	10	00:00:00.01	6	0

Predicate Information (identified by operation id):

2 - filter((NVL("MM_DISTRIBUTION_DEL_STATUS", 'H') <> 'D' AND NVL("MM_OUTBOUND_DEL_STATUS", 'H') <> 'D' AND NVL("MM_OUTBOUND_STATUS", 'H') <> 'U'))

3 - access("MM_WAREHOUSE_ID"=:B4 AND "MM_FIRM_ID"=:B1 AND "MM_ITEM_ID"=:B2 AND "MM_FROM_LOCATION_ID"=:B3) filter((NVL("MM_DISTRIBUTION_STATUS", 'H') <> 'U' AND NVL("MM_DIRECT_DEL_STATUS", 'H') <> 'D'))

Figure 5: A newly built index contains extra filter columns

The time has dropped by a factor of 700:1 and the “buffer gets” by a factor of 295:1. Thus, adding the additional columns was a big success. In this case, we did not need to think about the alternative, which would have been to create a new index composed of the filter columns.

If those columns filter so well, why don’t we create an index for them and start the search with it? (What is superior certainly also depends on other queries and their search criteria.) In our example above, such a thing would not have been possible, since the fields added to the index are compared with a non-equal operator. Because of that non-equivalence operation, accessing an index via the tree structure is not possible. However, it’s possible to use these fields as filters during the range scan, as clearly demonstrated in Figure 5. There is no reason why this should not work.

Strangely, the developer who wrote that code insisted that he wanted the new index dropped, because it was “against the rules.” For me, the primary rule is that what works is good—provided there are no other circumstances to consider, just like in one of my favorite quotes: “There are no ‘mitigating circumstances’ when it comes to rebellion against a sovereign lord.” “Unless you win.”[2]

When I wrote my query to search for improvable indexes, I decided to write only the version with runtime statistics (Figure 6). I thought it was better to have precise information. By the way, you might decide to take out the RULE hint. I used it because I was too impatient to fiddle with dictionary or fixed-table statistics in the event that the query is running on a database without them.

```
SELECT *
FROM
  (SELECT /*+ RULE */
    table_name,
    index_name,
    filter_predicates,
    ps_executions executions,
    ps_cr_buffer_gets "bg_exec",
    ROUND ((pas_output_rows - ps_output_rows) / pas_output_rows * 100, 0) "potential %"
  FROM
    (SELECT SUM (pas.executions) EXEC,
      p.object_name table_name,
      pa.object_name index_name,
      p.filter_predicates,
      SUM (pas.output_rows) pas_output_rows,
      SUM (ps.output_rows) ps_output_rows,
      SUM (ps.cr_buffer_gets) ps_cr_buffer_gets,
      SUM (ps.executions) ps_executions
    FROM v$sql_plan p,
      v$sql_plan_statistics ps,
      v$sql_plan pa,
      v$sql_plan_statistics pas
    WHERE p.sql_id = ps.sql_id
      AND p.id = ps.operation_id
      AND p.plan_hash_value = ps.plan_hash_value
      AND p.child_number = ps.child_number
      AND p.sql_id = pa.sql_id
      AND p.plan_hash_value = pa.plan_hash_value
      AND p.child_number = pa.child_number
      AND p.id = pa.parent_id
      AND pa.operation LIKE 'INDEX%'
      AND p.operation = 'TABLE ACCESS'
      AND p.options LIKE 'BY INDEX ROWID%'
      AND pa.sql_id = pas.sql_id
      AND pa.plan_hash_value = pas.plan_hash_value
      AND pa.child_number = pas.child_number
      AND pa.id = pas.operation_id
      AND p.filter_predicates IS NOT NULL
      AND ps.output_rows < pas.output_rows * 0.95
      AND p.OBJECT_NAME IN
        (SELECT table_name FROM dba_tables WHERE owner='&myschema')
      AND pa.OBJECT_NAME IN
        (SELECT index_name
          FROM dba_indexes
          WHERE owner = '&myschema'
            and table_name=p.OBJECT_NAME)
    GROUP BY p.object_name,
      pa.object_name,
      p.filter_predicates
  )
ORDER BY ps_executions * ps_cr_buffer_gets * ((pas_output_rows - ps_output_rows) / pas_output_rows) DESC
)
WHERE rownum < 11
```

Figure 6: Finding an improvable index using runtime statistics

Conclusion

What I wrote here I tried myself, and I consider it the most important work I did in this respect.

However, it doesn’t begin to be all that we can find when querying the dictionary. I wish you good luck with your own queries!

References

- ¹ Codd, Edgar Frank, “Is Your DBMS Really Relational?” *ComputerWorld*, October 14, 1985.
- ² Clavell, James, *Shogun*, dialogue between fictional characters Yoshi Toranaga and John Blackthorne, discussing the American Revolution, 1975.
- ³ Flatz, Lothar, “Raiders of the Data Dictionary, The Curse of the Buffer Cache,” *NoCOUG Journal*, August 2015. ▲

Lothar Flatz started working with Oracle Database in 1989, in the days of Version 5. He worked for Oracle Corporation for 15 years and was a member of the Real-World Performance Group for two years. He is a member of the Oak Table network of Oracle scientists and specializes in performance tuning.

Copyright © 2016, Lothar Flatz

Many Things Oracle

by Biju Thomas



Biju Thomas

Editor's Note: Biju publishes daily Oracle tidbits on Facebook (fb/oraclenotes) and on Twitter (@biju_thomas). In this issue, Biju reminds us of a few interesting V\$ views from Oracle Database 12c and lower versions. He also shares a few tips about the MGMTDB database.

More V\$ Views

There are certain v\$ and DBA views that we use every day, some we never use, and some we do not even know existed. Here are a few data dictionary views that we do not use every day but that might fall into the never-knew-existed category.

The **V\$OBSOLETE_PARAMETER** view has been there since Oracle 10gR2 and displays information about obsolete initialization parameters. If any row of the view contains TRUE in the ISSPECIFIED column, then you should examine why the parameter is still specified in the initialization file. On my Linux 12.10.2 database, there are 133 rows in this view.

There is a difference between an obsolete and a deprecated parameter. My alert log has this entry when the database starts (it's an EBS database; hence, I need this parameter).

```
Deprecated system parameters with specified values:
sec_case_sensitive_logon
End of deprecated system parameter listing
```

V\$OBSOLETE_PARAMETER does not list sec_case_sensitive_logon. Deprecated parameters used in the instance can be found using query

```
SQL> SELECT name FROM v$parameter
       WHERE isdeprecated = 'TRUE'
       AND isdefault = 'FALSE';
```

```
NAME
-----
sec_case_sensitive_logon
```

V\$PATCHES shows the patches applied on an Oracle ASM instance and the list of patches applied to an Oracle Grid infrastructure home directory. This view is empty in a database instance. Only in an ASM instance does it show the patches applied. On the 12c database, you can use DBMS_QOPATCH package to query the patches applied to RDBMS home.

```
SQL> SELECT xmltransform(dbms_qopatch.get_opatch_lsinventory, dbms_
qopatch.get_opatch_xslt) X FROM dual;
```

DBMS_QOPATCH has several subprograms. To find the Oracle Home location and the inventory location for the instance, use

```
SQL> SELECT xmltransform(dbms_qopatch.get_opatch_lsinventory, dbms_
qopatch.get_opatch_xslt) X FROM dual;
```

V\$SESSION_BLOCKERS introduced in 11gR2 displays the blocker sessions for each blocked session. Each row represents a blocked and blocker session pair. If a session is blocked by multiple sessions, there will be multiple rows for that blocked session. The SID, SESS_SERIAL# combination identifies the blocked session, and BLOCKER_SID and BLOCKER_SESS_SERIAL# identifies the blocker session. These pairs are handy to join with V\$SESSION to find more information on the session.

Other related views providing blocking session information include V\$SESSION and DBA_WAITERS. The BLOCKING_SESSION column in V\$SESSION identifies the blocking session. DBA_WAITERS shows all the sessions that are waiting on a lock. DBA_BLOCKERS displays a session if it is not waiting for a locked object but is holding a lock on an object for which another session is waiting. GV\$SESSION and GV\$SESSION_BLOCKERS show information from all instances in RAC, whereas DBA_WAITERS and DBA_BLOCKERS only show information from the current instance—meaning that the blocker and waiter have to be from the same instance. This is a good reason not to use these views when V\$SESSION_BLOCKERS and V\$SESSION are more useful.

V\$BACKUP_NONLOGGED, new in 12c, displays information about nonlogged block ranges in data file backups recorded in the control file. A related view is V\$NONLOGGED_BLOCK. It displays ranges of nonlogged datafile blocks recorded in the control file. In version 12c and forward, RMAN validate no longer populates view V\$DATABASE_BLOCK_CORRUPTION; instead V\$NONLOGGED_BLOCK is updated. V\$COPY_NONLOGGED displays information about nonlogged block ranges in datafile copy blocks recorded in the control file.

V\$CON_SYSSTAT is similar to the V\$SYSSTAT view but is new in 12c and useful for container database (CDB) architecture. From 12c onwards, you might want to start using V\$CON_SYSSTAT instead of V\$SYSSTAT, so the scripts can be used on both CDB and non-CDB databases. One nice thing about V\$CON_SYSSTAT is that it includes both NAME and CLASS columns, so there's no need to join with the V\$STATNAME

view. Usually we join V\$SYSSTAT with V\$STATNAME to get a meaningful query result.

V\$DEAD_CLEANUP has been available since 11.2.0.4. This view is useful to know the status of dead processes and killed sessions in the instance. The STATE column, per the documentation, could have the following values (taken from Oracle documentation): UNSAFE TO ATTEMPT - Occurs for a killed session that has not been moved, so no cleanup can occur on it yet. CLEANUP PENDING - Occurs for a dead process/killed session that can be cleaned up, but PMON has not yet made an attempt. RESOURCES FREED - Occurs for a dead process/killed session where all children have been freed, but the process/killed session itself is not yet freed. RESOURCES FREED - PENDING ACK - Occurs for a killed session where all children have been freed, but the session itself cannot be freed until the owner has acked it. PARTIAL CLEANUP - Occurs if some of the children have been cleaned up.

V\$SESSIONS_COUNT is new in 12c and shows the session count in each PDB. The view always returns 254 rows, one for each PDB. The same information can be easily obtained from V\$SESSION as well. For non-CDB databases, this view has no meaningful information.

V\$INSTANCE in 12c includes an interesting new column FAMILY that is not documented. The EDITION column identifies the edition of the install. See MOS Note “V\$instance in 12c, What is family, What is the meaning of CORE EE, CORE SE? (Doc ID 1922322.1).”

V\$RESERVED_WORDS is a handy reference for SQL and PL/SQL developers. This view displays a list of all SQL reserved keywords. To determine whether a particular keyword is reserved in any way, check the RESERVED, RES_TYPE, RES_ATTR, and RES_SEMI columns. The Database SQL Language Reference manual also includes an appendix with a few of the reserved words listed.

The MGMTDB Database in the 12c Cluster

When you install Oracle Clusterware 12c, the MGMTDB is automatically installed and configured. MGMTDB is the Oracle Grid Infrastructure Management Repository (GIMR) database. It is an Oracle database that stores real-time operating system metrics collected by the Cluster Health Monitor (CHM). The instance name is -MGMTDB (notice the minus at the beginning, which indicates a special database, similar to the plus you have for ASM instances), and the database name is _MGMTDB. It is a single-tenant container database with just one pluggable database. The GIMR database instance runs on one node in the cluster and must support failover to another node in case of node or storage failure.

When you install Oracle Grid Infrastructure 12c, the GI management repository database MGMTDB is created by default on the same disk group as the OCR. A disk group having both OCR and voting files together with the Grid Infrastructure Management Repository (GIMR) database would require at least 6 GB when external redundancy is used. Chapter 7 of the GIMR installation guide provides details on the space requirement. In my 12.1.0.2 installation, which has been running for a few months, the CDB\$ROOT size is around 1 GB and the PDB size is around 2.5 GB. The PDB has three tablespaces—SYSGRIDHOMEDATA, SYSMGMTDATA, and SYSMGMTDATADB—apart from the usual SYSAUX, SYSTEM, TEMP, and USERS tablespaces.


The GIMR database is managed using the “srvctl” commands, including “srvctl config mgmtdb”, “srvctl start mgmtdb”, “srvctl stop mgmtdb”, and “srvctl status mgmtdb”. The “mgmtca” tool is used to manage passwords in the GIMR.

If you would like to move the MGMTDB data files to another disk group, use an MDBUtil tool downloaded using MOS note 2065175.1. MDBUtil can also be used to delete and recreate the GIMR.

You might have an obvious question: If the GIMR database goes down for some reason, will my cluster or databases running on the cluster be affected? The answer is no: GIMR clients cache data locally. If the GIMR database goes down for some reason, CSR will restart or failover the database to another node. ▲


Biju Thomas is an Oracle ACE Director, an Oracle Certified Professional, and a Certified Oracle Database SQL Expert. He is a principal solutions architect at OneNeck IT Solutions, with more than 20 years of Oracle DBA experience. He is the author of Oracle 12c and 11g OCA, and co-author of Oracle 10g, 9i, and 8i OCP certification guides published by Sybex/Wiley. He is a frequent presenter at Oracle conferences and publishes articles for technical journals. Twitter @biju_thomas. Blog www.bijoos.com.

Copyright © 2016, Biju Thomas



Database Virtualization Software

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



www.delphix.com

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. ▲

Long-term event sponsorship:

CHEVRON

ORACLE CORP.

Thank you! Gold Vendors:

- Axxana
- Database Specialists
- Dell Software
- Delphix
- EMC
- HGST
- SolarWinds

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



TREASURER'S REPORT

Sri Rajan, Treasurer

Beginning Balance

January 1, 2015

\$ 56,512.02

Revenue

Corporate Membership	11,000.00
Individual Membership	9,480.00
Conference Walk-in Fees	2,149.00
Training Day Receipts	1,044.00
Gold Vendor Fees	10,000.00
Silver Vendor Fees	5,000.00
Conference Sponsorships	4,500.00
Journal Advertising	5,500.00
Charitable Contributions	450.00
Interest	5.48

Total Revenue

\$ 49,128.48

Expenses

Conference Expenses	27,585.09
Journal Expenses	16,759.89
Training Day Expenses	0.00
Board Expenses	2,234.19
PayPal Expenses	1,176.00
Software Dues	4,212.71
Insurance	500.00
Office Expenses	441.28
Meetup Expenses	89.94
Taxes and Filings	2,369.80
Marketing Expenses	0.00

Total Expenses

\$ 55,368.90

Ending Balance

December 31, 2015

\$ 50,271.60



Dr. DR

By Rich Parsons



Dr. DR is brought to you by Axxana.

TRAINING WHEN YOU WANT IT

TRAINING MEMBERSHIP INCLUDES

- Online training by Craig Shallahamer
- Mastering The Material email follow-up
- How-to webinars
- 24/7 unlimited access
- Priority response
- Discounts



Find out more at orapub.com.
Questions? Contact support@orapub.com.

YesSQL Summit 2016

by Stephen van Linge

I am the new membership director of NoCOUG, taking over from Joel Rosingana—the longest-serving NoCOUG volunteer—who retired from the NoCOUG board after 15 years of service, including two terms as NoCOUG president. To recognize his service, the NoCOUG board of directors made Joel the first-ever lifetime member of NoCOUG. Here is a picture of Joel and me during the Fall Conference at the PayPal Town Hall in San Jose in November.



I live in Escondido near San Diego in Southern California—about 500 miles away from the San Francisco Bay Area—so I log 1000 miles on the odometer for every conference. I've been told that I travel farther than anybody else on a regular basis to attend NoCOUG conferences. But there is no other Oracle user group between San Diego and San Francisco. Those who live in the San Francisco Bay Area are incredibly lucky to have such a vibrant Oracle user group in their own backyard.

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 every quarter. We have very few resources compared to the national and international user groups, but the NoCOUG volunteers have always managed to pull it off, quarter after quarter, for 30 years.

Yes—NoCOUG is now in its 30th year. But it is still steaming along as strong as ever. We've organized the first-ever YesSQL Summit on January 26–27 at the Oracle Conference Center in Redwood City. YesSQL Summit has an incredible slate of speakers, including Andy Mendelsohn, who started out at Oracle as a young C programmer and wrote the B-tree indexing code in Oracle Database that is still in use to this day. Andy is now the senior vice president of server technologies at Oracle. The NoCOUG Winter Conference is scheduled for the first day (January 26) of YesSQL Summit, but NoCOUG members may attend either day (January 26 or January 27) of YesSQL Summit for free as their quarterly membership entitlement. In addition to the presentations at YesSQL Summit, conference attendees can choose from a smorgasbord of business intelligence, data warehouse, analytics, graph, and spatial presentations at BIWA Sum-

mit and Spatial Summit, which are being held in conjunction with YesSQL Summit.

NoCOUG is not just a place where you expand your technology horizons but a place where you meet old friends and make new ones (sometimes you want to go where everybody knows your name and they're always glad you came). I'll be leading a tour of the Oracle campus during the lunch break. Here are two pictures from the campus tour at the last winter conference. I'm third from the right in the first picture. Also in the first picture are NoCOUG vice president Jeff Mahe (extreme right), NoCOUG book reviewer Brian Hitchcock (fifth from the right), NoCOUG webmaster Jimmy Brock (sixth from the right), and conference attendees from Sacramento and China (much farther away than San Diego).



I hope to see you at the conference. First-time NoCOUG conference attendees can attend the winter conference for free, so please invite your colleagues to request a free pass. ▲

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

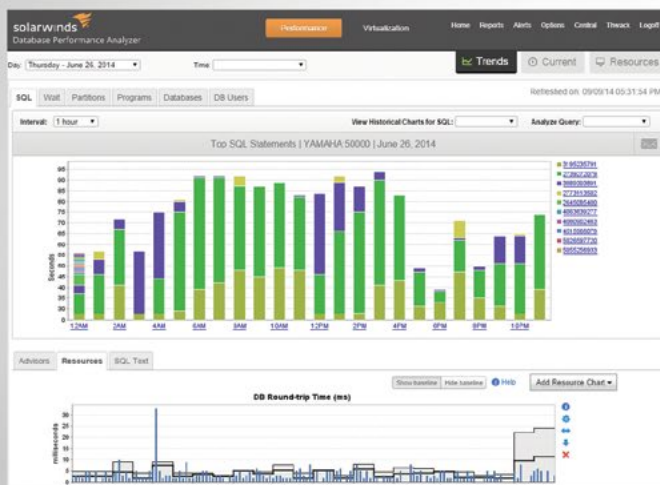
P.O. Box 3282

Danville, CA 94526

RETURN SERVICE REQUESTED



See what's new in Database Performance Analyzer 9.0



- Storage I/O analysis for better understanding of storage performance
- Resource metric baselines to identify normal operating thresholds
- Resource Alerts for full-alert coverage
- SQL statement analysis with expert tuning advice

Download free trial at: solarwinds.com/dpa-download