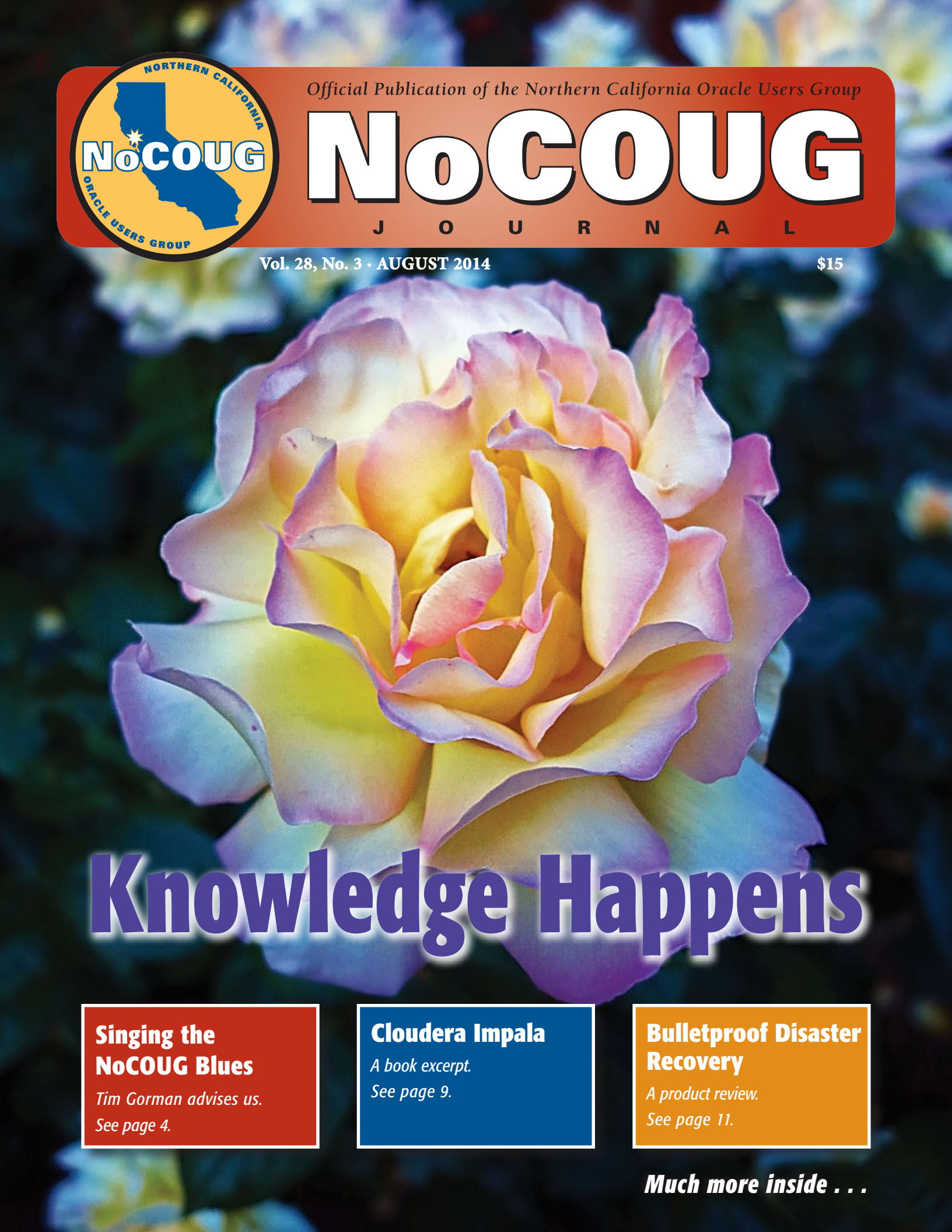
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 28, No. 3 · AUGUST 2014 $15



Knowledge Happens

Singing the NoCOUG Blues
Tim Gorman advises us.
See page 4.

Cloudera Impala
A book excerpt.
See page 9.

Bulletproof Disaster Recovery
A product review.
See page 11.

Much more inside . . .

Professionals at Work

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

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

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

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

This is the 111th issue of the *NoCOUG Journal*. Enjoy! ▲

—*NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

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.

Singing the NoCOUG Blues

with Tim Gorman

Tim Gorman

You are old, father Gorman (as the young man said) and your hair has become very white. You must have lots of stories. Tell us a story!

Well, in the first place, it is not my hair that is white. In point of fact, I'm as bald as a cue ball, and it is my skin that is pale from a youth misspent in data centers and fluorescent-lit office centers.

It is a mistake to think of wisdom as something that simply accumulates over time. Wisdom accumulates due to one's passages through the world, and no wisdom accumulates if one remains stationary. It has been said that experience is what one receives soon after they need it, and experience includes both success and failure. So wisdom accumulates with experience, but it accumulates fastest as a result of failure.

About four years ago, or 26 years into my IT career, I dropped an index on a 60 TB table with 24,000 hourly partitions; the index was over 15 TB in size. It was the main table in that production application, of course.

Over a quarter-century of industry experience as a developer, production support, systems administrator, and database administrator: if that's not enough time to have important lessons pounded into one's head, then how much time *is* needed?

My supervisor at the time was amazing. After the shock of watching it all happen and still not quite *believing* it had happened, I called him at about 9:00 p.m. local time and told him what occurred. I finished speaking and waited for the axe to fall—for the entirely justified anger to crash down on my head. He was silent for about 3 seconds, and then said calmly, "Well, I guess we need to fix it."

And that was it.

No anger, no recriminations, no humiliating micro-management. We launched straight into planning what needed to happen to fix it.

He got to work notifying the organization about what had happened, and I got started on the rebuild, which eventually took almost 2 weeks to complete.

It truly happens to all of us. And anyone who pretends otherwise simply hasn't been doing anything important.

How did I come to drop this index? Well, I wasn't *trying* to drop it; it resulted from an accident. I was processing an approved change during an approved production outage. I was trying to disable a unique constraint that was supported by the index. I wanted to do this so that a system-maintenance package I had written could perform partition exchange operations (which were blocked by an enabled constraint) on the table. When I tested the disabling of the constraint in the development environment, I used the command ALTER TABLE . . . DISABLE CONSTRAINT and it indeed disabled the unique constraint without affecting the unique index. Then I tested the same operation again in the QA/Test environment successfully. But when it came time to do so in production, it dropped the index as well.

Surprise!

I later learned that the unique constraint and the supporting unique index had been created out of line in the development and QA/test environments. That is, first the table was created, then the unique index was created, and finally the table was al-

"You are old, Father William," the young man said,
"And your hair has become very white;
And yet you incessantly stand on your head—
Do you think, at your age, it is right?"

"In my youth," Father William replied to his son,
"I feared it might injure the brain;
But now that I'm perfectly sure I have none,
Why, I do it again and again."

tered to create the unique constraint on the already-existing unique index.

But in production, the unique constraint and the supporting unique index had been created in-line. When the table was created, the CREATE TABLE statement had the unique constraint within, along with the USING INDEX clause to create the unique index.

So when I altered the table in production, disabling the constraint also caused the index to be dropped.

After the mishap, I found the additional syntax for KEEP INDEX, which could have been added to the end of the ALTER TABLE . . . DISABLE CONSTRAINT command because Oracle recognized the difference in default behaviors.

But that was a discovery I experienced after I needed it.

As to why my supervisor was so calm and matter-of-fact throughout this disaster, I was not surprised; he was always that way, it seemed. What I learned over beers long after this incident is that, in his early life, he learned the true meaning of the words "emergency" and "catastrophe." He was born in Afghanistan, and he was a young child during the 1980s after the Soviets invaded. His family decided to take refuge in Pakistan, so they sought the help of professional smugglers, similar to what we call "coyotes" on the Mexican-American border. These smugglers moved through the mountains bordering Afghanistan and Pakistan at night on foot, using camels to carry baggage and the very old, the sick and injured, and the very young.

My supervisor was about 9 years old at the time, so the smugglers put him on a camel so he would not slow them down. During the night, as they were crossing a ridge, they were spotted by the Soviets, who opened fire on them using machine guns with tracer bullets. Everyone in the caravan dove to the ground to take cover. Unfortunately, they all forgot about the 9-year-old boy on top of the 8-foot-high camel. My supervisor said he saw the bright tracer bullets arching up toward him from down below in the valley, passing over his head so close that he felt he could just reach up and grab them. He wanted to jump down, but he was so high off the ground he was terrified. Finally, someone realized that he was exposed and they pulled him down off the camel.

As he told this story, he laughed and commented that practically nothing he encountered in IT rose to the level of what he defined as an emergency. The worst that could happen was no more catastrophic than changing a tire on a car.

I've not yet been able to reach this level of serenity, but it is still something to which I aspire.

We love stories! Tell us another story!

A little over 10 years ago, I was working in downtown L.A. and arrived in the office early (5:00 a.m.) to start a batch job. I had a key card that got me into the building and into the office during the day, but I was unaware that at night they were locking doors in the elevator lobby. I banged on the doors and tried calling people, to no avail. Finally, after a half-hour, out of frustration, I grabbed one of the door handles and just *yanked* hard.

It popped open.

I looked at it in surprise, thought "sweet!", walked in to the cubicle farm, sat down, and started my batch job. All was good.

Around 7:00 a.m., the LAPD showed up. There were about a dozen people in the office now, so the two officers began questioning folks nearest the door. From the opposite side of the room, I stood up, called out "Over here," and 'fessed up.

They told me that if I hadn't called them over immediately, they would have arrested me by the time they got to me. Have a nice day, sir.

The NoCOUG Blues

NoCOUG membership and conference attendance have been declining for years. Are user groups still relevant in the age of Google? Do you see the same trends in other user groups? What are we doing wrong? What can we do to reverse the dismal trend? Give away free stuff like T-shirts and baseball caps? Bigger raffles? Better food?

Yes, the same trends are occurring in other users groups. IT organizations are lean and can't spare people to go to training. The IT industry is trending older as more and more entry-level functions are sent offshore.

Users groups are about education. Education in general has changed over the past 20 years as online searches, blogs, and webinars have become readily available.

The key to users groups is the quality of educational content that is offered during live events as opposed to online events or written articles. Although online events are convenient, we all know that we, as attendees, get less from them than we do from face-to-face live events. They're better than nothing, but communities like NoCOUG have the ability to provide the face-to-face live events that are so effective.

One of the difficulties users groups face is fatigue. It is difficult to organize events month after month, quarter after quarter, year after year. There is a great deal of satisfaction in running such an organization, especially one with the long and rich history enjoyed by NoCOUG. But it is exhausting. Current volun-

"You are old," said the youth, "As I mentioned before,
And have grown most uncommonly fat;
Yet you turned a back-somersault in at the door—
Pray, what is the reason of that?"

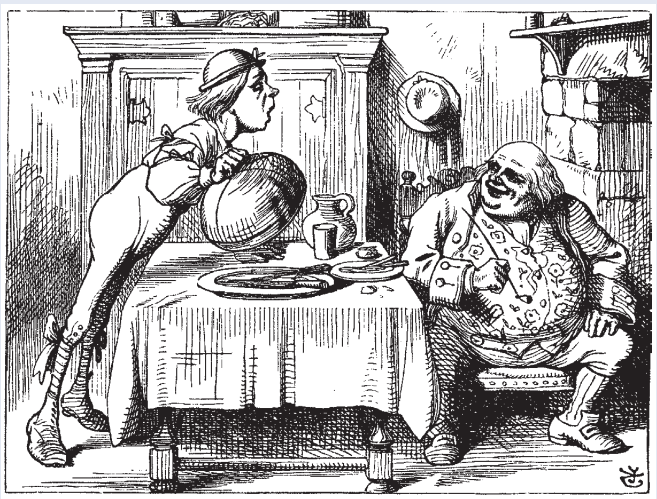
"In my youth," said the sage, as he shook his grey locks,
"I kept all my limbs very supple
By the use of this ointment—one shilling the box—
Allow me to sell you a couple?"

teers have overriding work and life conflicts. New volunteers are slow to come forward.

One thing to consider is reaching out to the larger national and international Oracle users groups, such as ODTUG, IOUG, OAUG, Quest, and OHUG. These groups have similar missions and most have outreach programs. ODTUG and IOUG in particular organize live onsite events in some cities, and have webinar programs as well. They have content, and NoCOUG has the membership and audience. NoCOUG members should encourage the board to contact these larger Oracle users groups for opportunities to partner locally.

Another growing trend is meet-ups, specifically through Meetup.com. This is a resource that has been embraced by all manner of tech-savvy people, from all points on the spectrum of the IT industry. I strongly urge all NoCOUG members to join Meetup.com, indicate your interests, and watch the flow of announcements visit your inbox. The meet-ups run the gamut from Hadoop to Android to Oracle Exadata to In-Memory to Big Data to Raspberry Pi to vintage Commodore. I think the future of local technical education lies in the intersection of online organization of local face-to-face interaction facilitated by Meetup.com.

Four conferences per year puts a huge burden on volunteers. There have been suggestions from multiple quarters that we organize just one big conference a year like some other user groups. That would involve changing our model from an annual membership fee of less than $100 for four single-day conferences (quarterly) to more than $300 for a single multiple-day conference (annual), but change is scary and success is not guaranteed. What are your thoughts on the quarterly vs. annual models?



"You are old," said the youth, "And your jaws are too weak
For anything tougher than suet;
Yet you finished the goose, with the bones and the beak—
Pray, how did you manage to do it?"

"In my youth," said his father, "I took to the law,
And argued each case with my wife;
And the muscular strength which it gave to my jaw,
Has lasted the rest of my life."

I disagree with the idea that changing the conference format requires increasing annual dues. For example, RMOUG in Colorado (**http://rmoug.org/**) has one large annual conference with three smaller quarterly meetings, and annual dues are $75 and have been so for years. RMOUG uses the annual dues to pay for the three smaller quarterly education workshops (a.k.a. quarterly meetings) and the quarterly newsletter; the single large annual "Training Days" conference pays for itself with its own separate registration fees, which of course are discounted for members.

Think of a large annual event as a self-sufficient, self-sustaining organization within the organization, open to the public with a discount for dues-paying members.

Other Oracle users groups, such as UTOUG in Utah (**http://utoug.org/**), hold two large conferences annually (in March and November), and this is another way to distribute scarce volunteer resources. This offers a chance for experimentation as well, by hiring one conference-coordinator company to handle one event and another to handle the other, so that not all eggs are in one basket.

The primary goal of larger conferences is ongoing technical education of course, but a secondary goal is to raise funds for the continued existence of the users group and to help subsidize and augment the website, the smaller events, and the newsletter, if necessary.

It costs a fortune to produce and print the* NoCOUG Journal*, but we take a lot of pride in our unbroken 28-year history, in our tradition of original content, and in being one of the last printed publications by Oracle user groups. Needless to say it also takes a lot of effort. But is there enough value to show for the effort and the cost? We've been called a dinosaur. Should we follow the other dinosaurs into oblivion?

I don't think so. There are all kinds of formats for publication, from tweets to LinkedIn posts to blogs to magazines to books. Magazines like the *NoCOUG Journal* are an important piece of the educational ecosystem. I don't think that any of the Oracle users groups who no longer produce newsletters planned to end up this way. They ceased publishing because the organization could no longer sustain them.

I think today the hurdle is that newsletters can no longer be confined within the users group. Both NoCOUG and RMOUG have independently come to the realization that the newsletter must be searchable and findable online by the world, which provides the incentive for authors to submit content. Today, if it cannot be verified online, it isn't real. If it isn't real, then there is little incentive for authors to publish.

So making the *NoCOUG Journal* available online has been key to its own viability, and NoCOUG membership entitles one to a real hard-copy issue, which is a rare and precious bonus in this day and age.

Oracle Database 12*c*

Mogens Norgaard (the co-founder of the Oak Table Network) claims that* "since Oracle 7.3, that fantastic database has had pretty much everything normal customers need," *but the rest of us are not confirmed Luddites. What are the must-have features of Oracle 12c that give customers the incentive to upgrade from 11g to 12c? We've heard about pluggable databases and the in-memory option, but they are extra-cost options aren't they?

I know for a fact that the Automatic Data Optimization (ADO) feature obsolesces about 3,000 lines of complex PL/SQL code that I had written for Oracle 8*i*, 9*i*, 10*g*, and 11*g* databases. The killer feature within ADO is the ability to move partitions online, without interrupting query operations. Prior to Oracle 12*c*, accomplishing that alone consumed hundreds of hours of code development, testing, debugging, and release management. Combining ADO with existing features like OLTP compression and HCC compression truly makes transparent "tiers" of storage within an Oracle database feasible and practical. The ADO feature alone is worth the effort of upgrading to Oracle 12*c* for an organization with longer data retention requirements for historical analytics or regulatory compliance.

What's not to love about pluggable databases? How different is the pluggable database architecture from the architecture of SQL Server, DB2, and MySQL?

I think that first, in trying to explain Oracle pluggable databases, most people make it seem more confusing than it should be.

Stop thinking of an Oracle database as consisting of software, a set of processes, and a set of database files.

Instead, think of a database server as consisting of an operating system (OS) and an Oracle 12*c* container database software; a set of Oracle processes; and the basic control files, log files, and a minimal set of data files. When "gold images" of Oracle database servers are created, whether for jumpstart servers or for virtual machines, the Oracle 12*c* CDB should be considered part of that base operating system image.

Pluggable databases (PDBs) then are the data files installed along with the application software they support. PDBs are just tablespaces that plug into the working processes and infrastructure of the CDBs.

When PDBs are plugged in, all operational activities involving data protection—such as backups or redundancy like Data Guard replication—are performed at the higher CDB level.

Thus, all operational concerns are handled at the CDBs and the operational infrastructure from the PDBs and the applications.

Once the discussion is shifted at that high level, then the similarities are more visible between the Oracle 12*c* database and other multitenant databases, such as SQL Server and MySQL. Of course there will always be syntactic and formatting differences, but functionally Oracle 12*c* has been heavily influenced by its predecessors, such as SQL Server and MySQL.

Bonus Question

Do you have any career advice for the younger people reading this interview so that they can be like you some day? Other than actively participating in user groups!

This sounds corny and trite, but there is no such thing as a useless experience, and while it may be frustrating, it presents the opportunity to build. Understand that everyone starts at the bottom, and enjoy the climb.

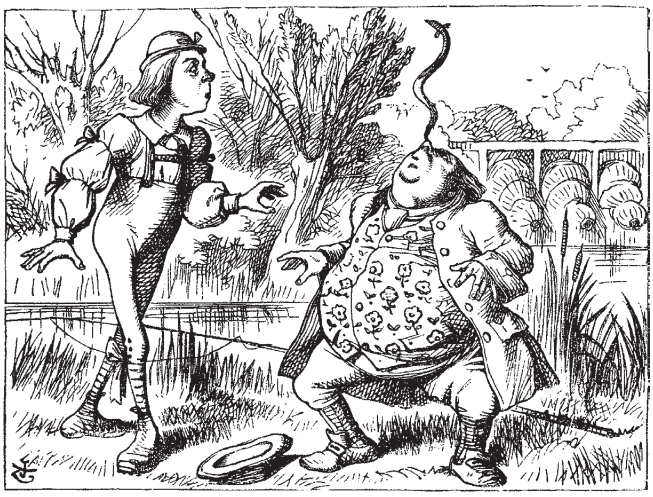
Understand that learning causes stress. Stress is stress and too much can be unhealthy, but if it is a result of learning something new, then recognize it for what it is, know it is temporary and transitory, tough it out, and enjoy knowing the outcome when it arrives.

Also, don't voice a complaint unless you are prepared to present at least one viable solution, if not several. Understand what makes each solution truly viable and what makes it infeasible. If you can't present a solution to go with the complaint, then more introspection is needed. The term "introspection" is used deliberately, as it implies looking within rather than around.

Help people. Make an impact. Can we go wrong in pursuing either of those as goals? Sometimes I wish I had done more along these lines. Never do I wish I had done less. ▲

---

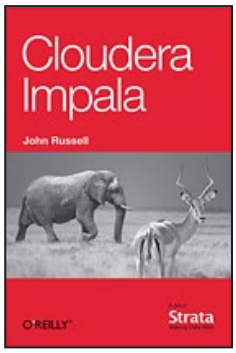
*Tim Gorman is a technical consultant for Delphix (**http://www.Delphix.com**), who enable database and storage virtualization to increase the agility of IT development and testing operations. He has co-authored six books, tech-reviewed eight more, and written articles for RMOUG SQL_Update and IOUG SELECT magazines. He has been an Oracle ACE since 2007, an Oracle ACE Director since 2012, a member of the Oak Table Network since 2002, and has presented at Oracle OpenWorld, Collaborate, KScope, Hotsos, and local Oracle users groups in a lot of wonderful places around the world. Tim lives in Westminster, Colo., with his partner, Kellyn Pot'Vin, and their children.*



"You are old," said the youth, "one would hardly suppose
That your eye was as steady as ever;
Yet you balanced an eel on the end of your nose—
What made you so awfully clever?"

"I have answered three questions, and that is enough,"
Said his father; "don't give yourself airs!
Do you think I can listen all day to such stuff?
Be off, or I'll kick you down stairs!"

Cloudera Impala



by John Russell

This is a short excerpt from Cloudera Impala *by John Russell, reprinted with permission. Cloudera Impala is an open-source relational query engine included in the Cloudera Hadoop distribution on the Oracle Big Data Appliance. You can download the book at* **http://goo.gl/xkVskl** *for free.*

The Cloudera Impala project arrives in the Big Data world at just the right moment. Data volume is growing fast, outstripping what can be realistically stored or processed on a single server. Some of the original practices for Big Data are evolving to open that field up to a larger audience of users and developers.

Impala brings a high degree of flexibility to the familiar database ETL process. You can query data that you already have in various standard Apache Hadoop file formats. You can access the same data with a combination of Impala, Apache Hive, and other Hadoop components such as Apache Pig or Cloudera search, without needing to duplicate or convert the data. When query speed is critical, the new Parquet columnar file format makes it simple to reorganize data for maximum performance of data warehouse-style queries.

Traditionally, Big Data processing has been like batch jobs from mainframe days, where unexpected or tough questions required running jobs overnight or all weekend. The goal of Impala is to express even complicated queries directly with familiar SQL syntax, running fast enough that you can get an answer to an unexpected question while a meeting or phone call is in progress. (We refer to this degree of responsiveness as "interactive.")

For users and business intelligence tools that speak SQL, Impala brings a more effective development model than writing a new Java program to handle each new kind of analysis. Although the SQL language has a long history in the computer industry, with the combination of Big Data and Impala, it is once again cool. Now you can write sophisticated analysis queries using natural expressive notation, the same way Perl mongers do with text-processing scripts. You can traverse large data sets and data structures interactively like a Pythonista inside the Python shell. You can avoid memorizing verbose specialized APIs; SQL is like a RISC instruction set that focuses on a standard set of powerful commands. When you do need access to API libraries for capabilities such as visualization and graphing, you can access Impala data from programs written in languages such as Java and C++ through the standard JDBC and ODBC protocols.

Flexibility

Impala integrates with existing Hadoop components, security, metadata, storage management, and file formats. You keep the flexibility you already have with these Hadoop strong points and add capabilities that make SQL queries much easier and faster than before.

With SQL, you can turn complicated analysis programs into simple, straightforward queries. To help answer questions and solve problems, you can enlist a wide audience of analysts who already know SQL or the standard business intelligence tools built on top of SQL. They know how to use SQL or BI tools to analyze large data sets and how to quickly get accurate answers for many kinds of business questions and "what if " scenarios. They know how to design data structures and abstractions that let you perform this kind of analysis both for common use cases and unique, unplanned scenarios.

The filtering, calculating, sorting, and formatting capabilities of SQL let you delegate those operations to the Impala query engine, rather than generating a large volume of raw results and coding client-side logic to organize the final results for presentation.

Impala embodies the Big Data philosophy that large data sets should be just as easy and economical to work with as small ones. Large volumes of data can be imported instantaneously, without any changes to the underlying data files. You have the flexibility to query data in its raw original form, or convert frequently queried data to a more compact, optimized form. Either way, you do not need to guess which data is worth saving; you preserve the original values, rather than condensing the data and keeping only the summarized form. There is no required step to reorganize the data and impose structure and rules, such as you might find in a traditional data warehouse environment.

Performance

The Impala architecture provides such a speed boost to SQL queries on Hadoop data that it will change the way you work. Whether you currently use MapReduce jobs or even other SQL-on-Hadoop technologies such as Hive, the fast turnaround for Impala queries opens up whole new categories of problems that you can solve. Instead of treating Hadoop data analysis as a batch process that requires extensive planning and scheduling, you can get results any time you want them.

Instead of doing a mental context switch as you kick off a batch query and later discover that it has finished, you can run a query, evaluate the results immediately, and fine-tune the query

if necessary. This fast iteration helps you zero in on the best solution without disrupting your workflow. Instead of trying to shrink your data down to the most important or representative subset, you can analyze everything you have, producing the most accurate answers and discovering new trends.

Perhaps you have had the experience of using software or a slow computer where after every command or operation, you waited so long that you had to take a coffee break or switch to another task. Then when you switched to faster software or upgraded to a faster computer, the system became so responsive that it lifted your mood, reengaged your intellect, and sparked creative new ideas. That is the type of reaction Impala aims to inspire in Hadoop users.

Coming to Impala from an RDBMS Background

When you come to Impala from a background with a traditional relational database product, you find the same familiar SQL query language and DDL statements. Data warehouse experts will already be familiar with the notion of partitioning. If you have only dealt with smaller OLTP-style databases, the emphasis on large data volumes will expand your horizons.

Standard SQL

The great thing about coming to Impala with relational database experience is that the query language is completely familiar: it's just SQL! The SELECT syntax works like you are used to, with joins, views, relational operators, aggregate functions, ORDER BY and GROUP BY, casts, column aliases, built-in functions, and so on.

Because Impala is focused on analytic workloads, it currently doesn't have OLTP-style operations such as DELETE, UPDATE, or COMMIT / ROLLBACK. It also does not have indexes, constraints, or foreign keys; data warehousing experts traditionally minimize their reliance on these relational features because they involve performance overhead that can be too much when dealing with large amounts of data.

The initial Impala release supports a set of core column data types: STRING instead of VARCHAR or VARCHAR2; INT and FLOAT instead of NUMBER; and no BLOB type.

The CREATE TABLE and INSERT statements incorporate some of the format clauses that you might expect to be part of a separate data-loading utility, because Impala is all about the shortest path to ingest and analyze data.

The EXPLAIN statement provides a logical overview of statement execution. Instead of showing how a query uses indexes, the Impala EXPLAIN output illustrates how parts of the query are distributed among the nodes in a cluster, and how intermediate results are combined at the end to produce the final result set.

Impala implements SQL-92 standard features with some enhancements from later SQL standards. It does not yet have does not yet have the SQL-99 and SQL-2003 analytic functions, although those items are on the product roadmap.

Storage, Storage, Storage

Several aspects of the Apache Hadoop workflow, with Impala in particular, are very freeing to a longtime database user:

The data volumes are so big that you start out with a large pool of storage to work with. This reality tends to reduce the bureaucracy and other headaches associated with a large and fast-growing database.

The flexibility of Impala schemas means there is less chance of going back and reorganizing old data based on recent changes to table structures.

The HDFS storage layer means that replication and backup are handled at the level of an entire cluster rather than for each individual database or table.

The key is to store the data in some form as quickly, conveniently, and scalably as possible through the flexible Hadoop software stack and file formats. You can come back later and define an Impala schema for existing data files. The data loading process for Impala is very lightweight; you can even leave the data files in their original locations and query them there.

Billions and Billions of Rows

Although Impala can work with data of any volume, its performance and scalability shine when the data is large enough to be impractical to produce, manipulate, and analyze on a single server. Therefore, after you do your initial experiments to learn how all the pieces fit together, you very quickly scale up to working with tables containing billions of rows and gigabytes, terabytes, or larger of total volume. The toy problems you tinker with might involve data sets bigger than you ever used before. You might have to rethink your benchmarking techniques if you are used to using smaller volumes—meaning millions of rows or a few tens of gigabytes. You will start relying on the results of analytic queries because the scale will be bigger than you can grasp through your intuition.

For problems that do not tax the capabilities of a single machine, many alternative techniques offer about the same performance. After all, if all you want to do is sort or search through a few files, you can do that plenty fast with Perl scripts or Unix commands such as grep. The Big Data issues come into play when the files are too large to fit on a single machine, or when you want to run hundreds of such operations concurrently, or when an operation that takes only a few seconds for megabytes of data takes hours when the data volume is scaled up to gigabytes or petabytes.

You can learn the basics of Impala SQL and confirm that all the prerequisite software is configured correctly using tiny data sets, as in the examples throughout this article. That's what we call a "canary test," to make sure all the pieces of the system are hooked up properly. To start exploring scenarios involving performance testing, scalability, and multi-node cluster configurations, you typically use much, much larger data sets. Try generating a billion rows of representative data, then once the raw data is in Impala, experiment with different combinations of file formats, compression codecs, and partitioning schemes.

Don't put too much faith in performance results involving only a few gigabytes of data. Only when you blow past the data volume that a single server could reasonably handle or saturate the I/O channels of your storage array can you fully appreciate the performance speedup of Impala over competing solutions and the effects of the various tuning techniques. To really be sure, do trials using volumes of data similar to your real-world system.

If today your data volume is not at this level, next year it might be. You should not wait until your storage is almost full (or even half full) to set up a big pool of HDFS storage on cheap commodity hardware.

Whether or not your organization has already adopted the Apache Hadoop software stack, experimenting with Cloudera Impala is a valuable exercise to future-proof your enterprise. ▲

Bulletproof Disaster Recovery

by Liat Malki

Liat Malki

Axxana's newest member of the Phoenix System family provides a "bulletproof" disaster recovery solution for Oracle environments. This new solution from Axxana ensures that in case of disaster, no transaction data is lost and all applications and databases are recovered with full consistency, guaranteeing the quickest recovery time possible.

Phoenix System for Oracle increases critical application availability by delivering zero transaction loss and enabling up to zero recovery time in a cost-effective DR solution.

Prior to implementing Axxana's Phoenix System for Oracle, an existing asynchronous replication topology should be in place between the primary production site and a secondary DR site. The two sites can leverage any type of storage as well as Exadata; any type of network, such as SAN, InfiniBand, or IP-based; and storage-based or Data Guard replication.

When using the asynchronous replication mode (e.g., Data Guard in Max Performance mode), there is an inherent lag between the primary and DR sites. This lag creates a "delta" between the real-time committed transaction data at the primary site and the remote database at the DR site. In case a disaster hits the primary data center, the data that comprises this lag will be lost. Applications and databases will lose consistency, which will put the recovery efforts at a significant risk and slow down the failover of the business to the remote site.

Phoenix System for Oracle by Axxana combines best-of-breed technologies to provide cost-effective continuous data protection for mission-critical applications with zero data and application loss, and up to zero recovery time, ensuring application-level consistency at restart.

This article describes Axxana's solution to overcome these challenges by using the Phoenix System for Oracle, ensuring that no transaction data is ever lost, maintaining database consistency and shortening recovery time.

Current Challenges in Database Disaster Recovery

When replicating asynchronously to a remote site, there is always a lag between the main and the DR site. In case of a disaster, this lag means the following:

- **Transaction data is lost:** All recent transaction data that constitutes the lag—the data that has not yet been replicated to the DR site—is lost.
- **Recovered applications and databases will be inconsistent:** When multiple databases of different, dependent applications are being replicated in an asynchronous man-

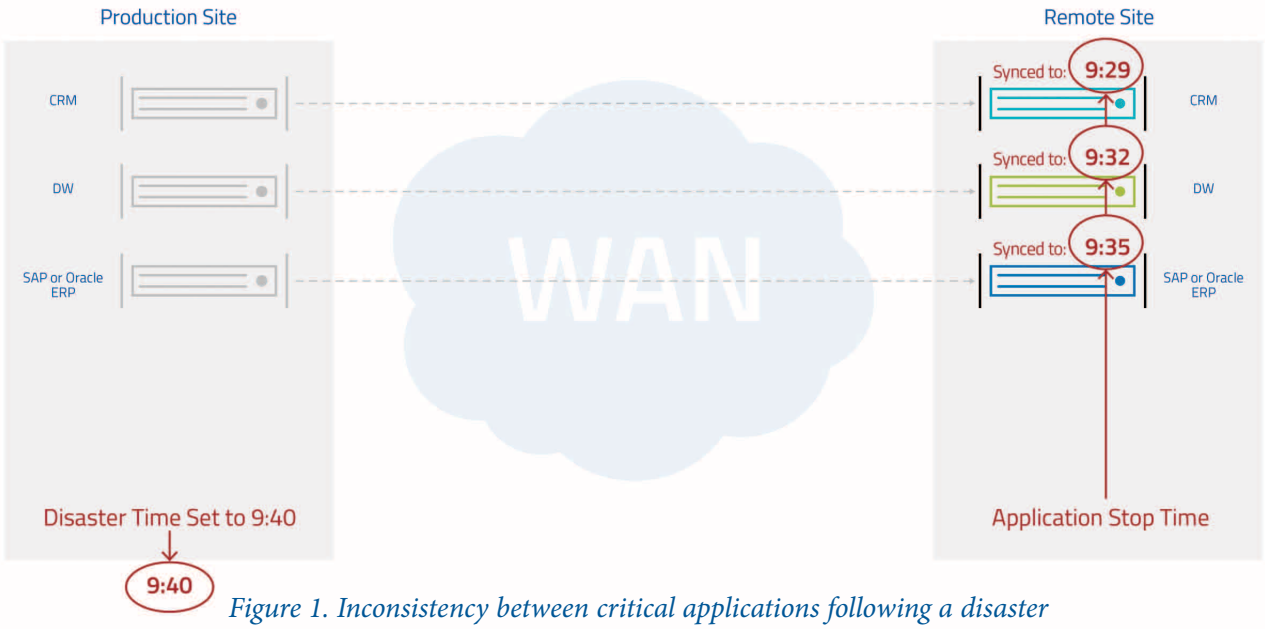


Figure 1. Inconsistency between critical applications following a disaster

ner, the replicated data of each database at the remote site will be current to a different point in time. This means that when there is data loss, there will be no consistency between the various databases and applications, in which case, in the best scenario, an extensive manual labor process will be involved in order to recover operation to one consistent point in time, across all databases and applications. Figure 1 illustrates how, following a disaster, the applications cease to operate at different points in time.

- **Performance degradation:** When replicating in the Data Guard Maximum Protection or Maximum Availability modes, application performance becomes an issue. Maximum Protection will slow down the application, which then will not perform properly when replicating remotely. The application and database will be affected by network latency. Maximum Availability, when replicating across long distances, will replicate asynchronously the majority of the time, with the same inherent risk to the data and applications.

How the Phoenix System Protects Oracle Databases

Axxana's Phoenix System for Oracle provides a solution to the above challenges, ensuring that in case of a disaster, no transaction data is lost and all applications and databases are kept consistent during recovery to the latest state of the business right when the disaster hit. The result: zero transactions are lost, and the recovery process is significantly shortened since there are no consistency issues.

At the heart of the Phoenix System for Oracle is Axxana's Phoenix Black Box. Similar to aviation flight data recorders (AKA airplane black boxes), the Axxana Black Box is a sturdy data-protection storage system, designed to withstand a wide variety of extreme conditions that may occur during a disaster. The Axxana Black Box is built to survive various scenarios, including fire; flood; natural disasters, such as earthquakes and extreme weather; and strong mechanical forces resulting from drops, shocks, heavy loads, and piercing.

The Phoenix Black Box is installed at the primary site, protecting the Oracle Redo and Archive log files. The Black Box has built-in capabilities to enable it to automatically identify a disaster, right when it happens. Post disaster, the Black Box will transfer the delta data that it protects to the secondary data center, so that this data can be used in the recovery process. The Oracle database can thus be recovered to the exact state it was in when the disaster hit the primary data center. All databases and applications are recovered to that exact same point in time, ensuring a complete, lossless, and consistent state of the business across all databases and applications.

Figure 2 shows an example of a successful recovery process with Axxana. All databases and applications are recovered successfully to the exact same point in time.

During its normal operation, the Oracle database creates three types of files:

1. **Data files:** This is the database itself, replicated on a continuous basis to the DR site.
2. **Redo Log files:** These are buffer files between the Oracle server and the database. The Oracle Log Writer (LGWR) writes first to the Redo Log files and then to the database. The Redo Log files contain, at any given moment, the newest and most current transaction data that had occurred; they are therefore extremely important for a complete and full recovery in case of disaster.
3. **Archived Log files:** Archived Log files are old Redo Logs. Archived Log files are accumulated on a continuous basis and up to a certain capacity (or data age, such as 24 hours) that is predetermined by the Oracle DBA. Once this capacity or data age is reached, the oldest Archive Log files are deleted.

The missing information that constitutes the lag resides in the Redo Log files and, if the lag is large, also in the Archive Log files. Using the complete set of Redo logs and Archives, the Oracle database knows how to reconstruct its data files up to the exact point in time in which the disaster occurred, and with full consistency between all applications and databases.

The Phoenix System for Oracle is built to maintain a synchronized, protected copy of both the Redo Logs and the Archives. When a disaster occurs and the primary storage is destroyed, the

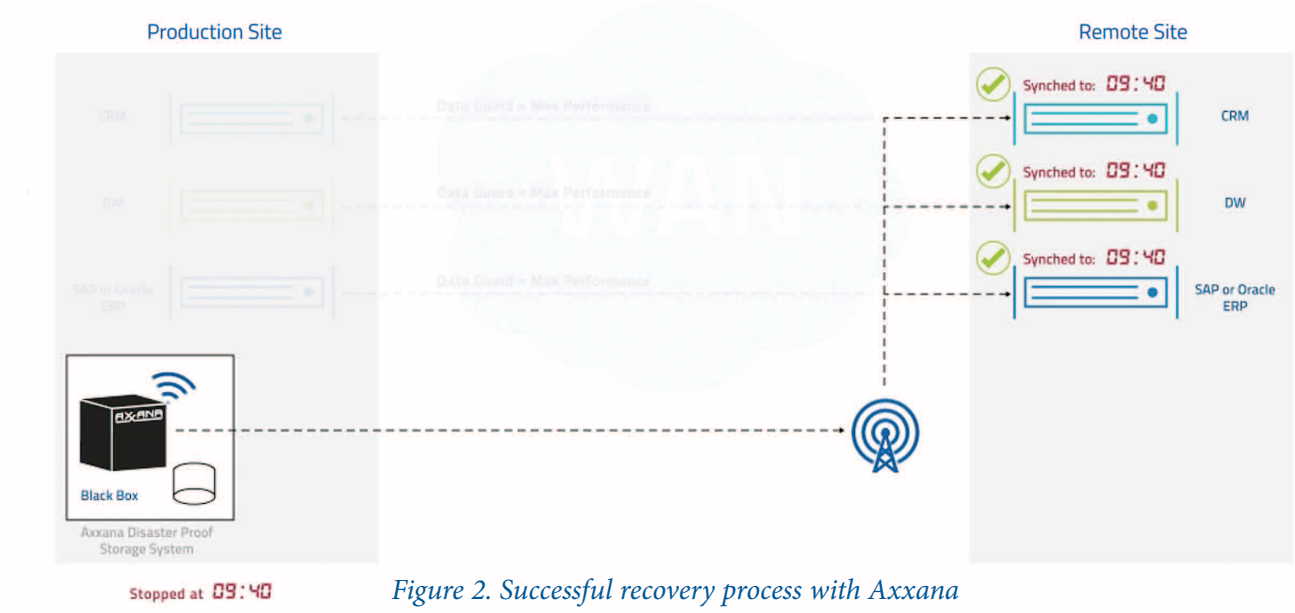


Figure 2. Successful recovery process with Axxana

Phoenix Black Box will transfer the Redo logs and Archives it holds within—only the missing data lag—to the secondary site, to allow for a quick and full recovery.

Axxana also supports recovery from a link failure between the primary data center and the DR site in a synchronous replication scenario. When the WAN connection between the two sites fails, the Axxana solution enables the primary data center to continue production and not come to a halt, while the Axxana Black Box maintains a protected copy of all accumulated, unreplicated transaction data.

Axxana leverages Oracle's native multiplexing functionality—an internal mechanism Oracle uses that allows it to write its logs to two or more destinations in parallel, both to the primary storage and to the Axxana protected storage, inside the Black Box. The Axxana protected storage is defined as "out of data path," such that it does not affect production. Since Axxana leverages the internal Oracle multiplexing mechanism, it does not matter whether the organization uses an Oracle-based replication tool (e.g., Data Guard), or any type of storage-based replication. The Phoenix Black Box will receive and maintain a copy of the Redo Logs and Archive Logs in both replication scenarios.

Figure 3 shows The Phoenix System for Oracle infrastructure.

Phoenix System Black Box

Installed at the primary site, the Black Box protects the transactions data (delta of data) it holds within. Similar to aviation flight data recorders (black boxes), the Axxana Black Box is a sturdy data-protection storage system, designed to withstand a wide variety of extreme conditions that may occur during a disaster.

In addition to its sturdiness and survival features, the Black Box has built-in capabilities to enable it to (a) automatically detect a disaster, in most cases much earlier than an operator would detect it, and (b) post disaster, transfer the data it protects to the secondary data center, so that this data can be used in the recovery process.

Most disaster scenarios will result in the loss of external power supply. The Black Box is equipped with an internal battery, enabling it to continue operation for up to 36 hours post disaster. This is needed to enable the Black Box to autonomously communicate with the secondary data center and transfer the protected data.

Several built-in data communication mechanisms: In addition to the WAN connection that may be down as a result of the disaster, the Black Box offers the following options for transferring the data post disaster:

- **Cellular:** The Black Box features three protected cellular antennas, enabling it to broadcast at 360 degrees. These enable transferring the data over the cellular network—in an encrypted and authenticated way—to the Recoverer software located at the secondary site. The cellular network is the most reliable network in case of disasters and is the U.S. government's communications network of choice for emergency services, even during large-scale disasters, because it is never physically destroyed. Axxana's automatic mechanisms for the early detection of the disaster enable it to fully and quickly utilize the cellular networks, even before potential congestion occurs.
- **Laptop:** The Black Box structure has a secure port that enables connecting a laptop equipped with the Axxana software and downloading the transaction data to that laptop. Later, the laptop can be connected to the Internet to securely transfer the encrypted data to the Recoverer software at the remote site.
- **Disk Extraction:** As a last resort, and if all other transfer options are unavailable, it is possible to extract the solid state disks from inside the box and transfer them to the remote site, where they can be connected to the Recoverer to continue the protected data-retrieval process.

Phoenix System Collector

Installed at the production site, the Phoenix System Collector acts as a storage controller that receives transaction data from the Oracle databases at the production site. It is a hardware unit that connects directly to the Black Box as well as to the Oracle database servers, over the SAN, or to an Exadata system.

The Collector performs the following tasks:

- Provisions volumes on the Black Box Solid State Drives, performs LUN masking, and exposes the volumes it created to the local Oracle host.

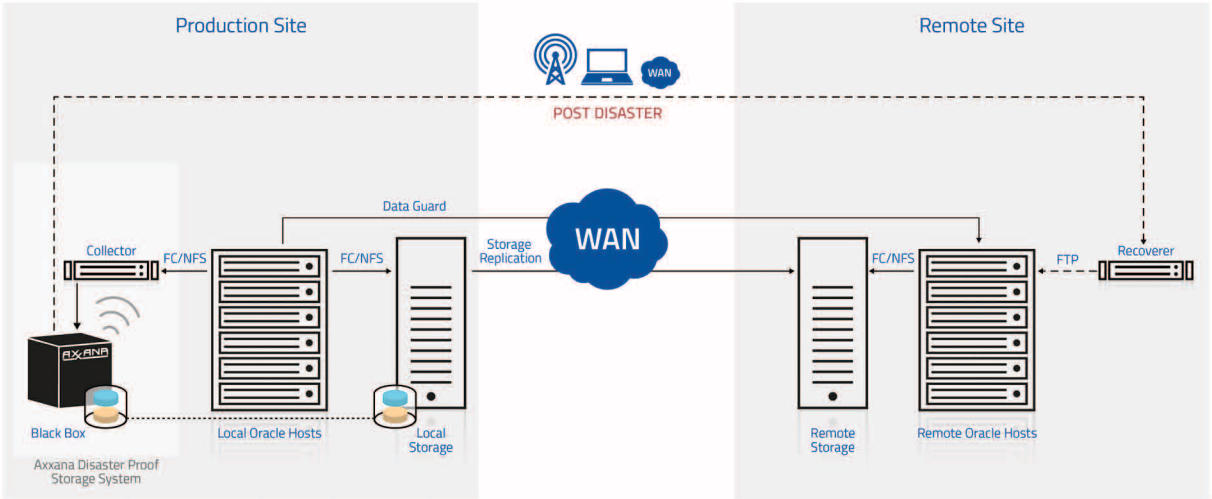


Figure 3. The Phoenix System for Oracle

Figure 4. The Phoenix System for Oracle—Black Box internal view

- Provides different interfaces to the hosts. The Collector comes in one of the following three configurations, depending on the type of network deployed by the customer:

1. **Fibre Channel (SAN):** The Collector connects to the FC SAN switch, through which the Oracle servers access their respective Axxana protected volumes.
2. **InfiniBand:** This configuration is used for Exadata. The Collector then connects directly to the Exadata array.
3. **IP Connectivity:** The Collector can connect through 10-gigabit Ethernet to support IP-based NAS and SAN networks that use the NFS or iSCSI file protocols.

The Collector also supports hybrid topologies for supporting, for instance, both FC SAN and Exadata connectivity simultaneously.

Note: The Collector is external to the Black Box and does not need to survive a disaster—the Collector's role ends once a disaster has occurred.

Phoenix System Recoverer

Installed at the remote site, the Phoenix System Recoverer manages the transaction data recovery process. The Recoverer maintains constant heartbeat connection with the Black Box via both WAN and cellular connections. In case of disaster, the protected logs are transferred from the Black Box to the Recoverer, where they await the Oracle DBA, who will use them for the Oracle Recovery process. The Recoverer is connected to the servers at the DR site via an Ethernet connection.

Axxana Phoenix System Installation and Protection Process

Adding the Phoenix System to the network is easy. It is performed in a nonintrusive way, without stopping production, having any effect on the production environment, or requiring any changes done to it.

The installation is performed by adding the Axxana Collector to the storage network, no matter what type of network is used by the organization (SAN, NAS, or InfiniBand). Once the Collector is connected to the network, the next step is to begin the Axxana Protection process.

The Axxana Protection process includes the following steps:

1. Perform sizing to find out the extent of the data lag, the size of the Redo Log files, and the size of the Archive files.
2. Provision a Protection Volume of a specific size inside the Black Box.
3. Execute LUN Masking to assign that volume to the specific Oracle database.

Once the above process is done, the Oracle server has a new Axxana volume available.

Note: Axxana's Phoenix System supports Oracle RAC. When attaching to Oracle RAC, the Axxana Protection Volume needs to be exposed to all nodes of the Oracle RAC.

The next stage in setting the Axxana Protection is performed by the Oracle DBA, who needs to define multiplexing for his Redo log files. This means adding a member to each Redo log group, in order to perform multiplexing. Once this stage is completed, the Oracle database will write to both its primary Redo logs and also to the secondary redo logs that are configured on Axxana.

Should Axxana fail for any reason, production and replication will not be affected. The Oracle database will continue to operate when the redo logs, saved within Axxana, are accessible.

Oracle will perform a Drop Member step, and from that moment on, the database will continue to use and access only the primary storage for writing Redo Log files. Once the connection with Axxana is resolved, the Oracle system recognizes that automatically and resumes multiplexing to the Axxana system.

Axxana also maintains a copy of the Archive Logs. This is done through adding another destination to the Oracle database, so that it writes the Archive logs to two directories—both to the primary storage and to the Axxana system. The Axxana destination is defined as optional, so that if it is not available, the Oracle server will ignore it and will not incur any performance degradation.

Archive Deletion Module

The Axxana volume is set, sizewise, to hold a specific number of hours of logs data, according to what was determined during the initial setup. As part of its ongoing operation, the Oracle server is continuously adding Archive Logs (the Redo Logs are cyclical and overlap themselves). In order to prevent a case in which the Axxana storage capacity is fully consumed, Axxana has a mechanism for deleting older Archive Logs while maximizing the use of protected storage, to maintain as many Archive Logs as possible. For example, if the organization has decided to keep 5 hours' worth of data, then once the Axxana storage fills up 5 hours of data, it will delete the oldest Archive Log to allow for a new Archive Log to be written.

This feature is critical in a case of a rolling disaster, examples of which are earthquakes, loss of power, and floods. A rolling disaster is often characterized by first losing the WAN connection between the primary and DR data centers, while the primary data center continues to function for an additional period of time, until it is also hit by the disaster and ceases production.

The challenge in such a rolling disaster scenario is that the data lag between the primary and secondary data centers continues to grow significantly, due to losing the WAN connection.

Axxana provides a solution to this rolling disaster scenario by keeping a protected copy of all the Redo Log files and Archived Logs that have not been replicated to the secondary site—up to the number of hours set in its setup process.

Phoenix System with Exadata and Data Guard

Phoenix System for Oracle has the ability to connect to Oracle's Exadata system through the InfiniBand and 10 GigE connections. The Oracle DBA performs the identical installation and setup procedures of the Phoenix System for Oracle, including setting up LUNs on Axxana's protected storage. The result is the same: the Exadata sees the Axxana LUNs and uses multiplexing to write the Redo Log files and Archive Logs to Axxana. This is exactly the same process shown earlier in this article, but in this case using InfiniBand connectivity, instead of Fibre Channel.

Exadata customers leverage Oracle's Data Guard replication software to replicate the transaction data to another Exadata machine, located at the remote site. Data Guard replication is done at the transaction level, as opposed to storage-based replication that replicates at the data level. Phoenix System for Oracle complements Data Guard, ensuring that no transactions are ever lost, including those that haven't yet been replicated by Data Guard.

Data Guard comes in three flavors:

- **Max Protection:** Data Guard's Synchronous mode of replication. This mode is usually not used for remote replication.
- **Max Performance:** Data Guard's Asynchronous replication mode.
- **Max Availability:** a combination of the above two modes. This mode begins with the Max Protection mode of Synchronous replication, but when performance slows down, it moves automatically to the Max Performance Asynchronous mode.

Most Oracle customers use the Max Availability or Max Performance modes, since Max Protection has a significant impact on performance. These two modes have an inherent lag between the production and secondary data centers, which means that transaction data will always be lost, in a case of disaster.

Adding Axxana's Phoenix System bridges this gap and provides the ultimate protection. Phoenix System enables customers to use Max Availability in its Asynchronous mode and still get full, Synchronous protection through the Axxana Black Box. In essence, Data Guard's Max Availability or Max Performance, combined with Axxana's Phoenix System, is as though the customer uses Max Protection.

The Recovery Process

Phoenix System for Oracle improves recovery and increases efficiency of an organization's DR by shortening the overall recovery time and providing application consistency with an aggressive zero data loss guarantee and a recovery time equal or very close to zero.

Data extraction, which is the first step in the recovery process, can be performed in two ways: automatic and manual. To shorten the recovery process, the Phoenix System for Oracle uses a unique, intelligent feature called Logical Transfer. The Axxana Phoenix solution knows exactly what data has been replicated to the disaster site and what hasn't, and therefore knows to transfer only the missing transactions and not the full log files.

The transaction data in the Phoenix Black Box that has already been replicated to the DR site will not be transferred again post disaster.

The Axxana Black Box, built to survive a wide range of disasters with the log files stored inside it, will survive the disaster while the Collector that connects between the Black Box and the Oracle database will be destroyed.

The Axxana Recoverer at the DR site keeps constant communication with the Black Box through heartbeats, both over WAN and over cellular connections, to verify connectivity with the primary site. When declaring disaster, an authenticated and encrypted data transfer is initiated from the Black Box, either manually or automatically:

Manual Recovery

The Admin at the DR site initiates a manual Failover command from the Recoverer. The Recoverer waits for the next heartbeat from the Black Box; once received, the Recoverer piggybacks the Failover command on the heartbeat ACK via the same communication method that was used to send the heartbeat itself (WAN or cellular). When the Black Box receives the Failover command, it moves from Normal mode to Failover mode in a two-step process:

1. The Black Box enters a "Failover" state in which it stops accepting writes from the Oracle LGWR.
2. The Recoverer at the DR site gets access to the log files inside the Black Box. The Recoverer then asks the Black Box to transfer these log files through whatever communication means are available. Phoenix System knows exactly what data is missing at the DR site, so only the missing data is transferred, in an encrypted and compressed way, to shorten transfer time.

Automatic Data Extraction

Transfer of the missing logs is triggered by the Black Box upon sensing the lack of power and/or communication to the box (i.e., a potential disaster). The advantage is that this entire process can be performed by the Phoenix System even before the organization has digested the fact that a disaster has occurred and before a high-level executive decision is made to failover production to the DR site.

The actual failover and database recovery is not performed without specific instruction and action of the DBA at the DR site.

This automatic process significantly shortens the recovery time, as the lag will most likely be transferred to the DR site before the organization has initiated such failover process.

On the other hand, if a disaster is not declared, then the transferred files are ignored, with no effect on the Oracle environment.

The next major step is for the organization to reach a decision to failover. Once such a decision is reached, the organization performs a failover either at the Data Guard level or at the storage level, turning the secondary environment in the DR site into the primary environment.

The Oracle hosts at the secondary site see all the replicated data: databases, data files, redo logs, archives, etc. The only caveat is that the replicated data is not current to the exact point in time of the disaster; it is from some previous point in time, de-

pending on the extent of the lag when the disaster occurred. It is impossible to know in real time how big the lag was and how much information is missing. The missing data, the delta that is equivalent to the lag, most likely already exists at the Axxana Recoverer in the DR site.

The AxxRecovery Tool

For the next steps in the recovery process, Axxana provides the Oracle DBA with a tool, called The AxxRecovery tool, that automates the entire recovery process. The tool asks the DBA for various acknowledgements during the process.

The AxxRecovery tool is used to retrieve the log files from the Axxana Recoverer, via an FTP connection. The AxxRecovery tool then places the specific log files for each database in the corresponding Oracle directory for that database at the local Oracle host. Once AxxRecovery finishes its work, all the log files from the Axxana Recoverer have been moved to the relevant directory at the Oracle host, and the Oracle database can begin the Recovery process.

The final step is for the Oracle DBA to start the Oracle database and perform a Recovery process. This Recovery process is performed from within the Oracle database, using the log files that were protected by the Phoenix System and transmitted to the DR location. Although the DBA can use standard Oracle tools to complete the Recovery process, AxxRecovery helps the DBA to perform this final task by semi-automating it—interactively asking for the DBA's acknowledgment in various recovery steps.

This Oracle Recovery process retrieves the Oracle database to the exact point in time where the production servers were when the disaster struck. The result is no loss of transaction data and achieving full consistency between all applications and databases in a relatively short period (minimal RTO).

Monitoring and Troubleshooting

Phoenix System for Oracle comes with integrated monitoring that supports the SNMP protocol and interfaces with various control systems that the organization may use through that protocol. The monitoring system closely watches overall performance of the Phoenix System and includes the following features in case it recognizes a problem:

- **Email Notifications:** The user can define various events, for which the system will generate an email notification.
- **Call Home:** If a user allows this option during the initial setup, Axxana Global Services will receive both an SNMP alert and an email notification when the system identifies a problem, so that an Axxana support team can immediately address the issue.

Summary

The Axxana Phoenix System for Oracle provides a complete data-protection solution for Oracle environments. The system augments existing Asynchronous replication products, whether Storage based or Data Guard, and adds a layer of protection to ensure that no transaction data is lost and all applications and databases are recovered with full consistency, guaranteeing the quickest possible recovery time.

The Axxana Black Box protects the inherent lag between the primary and DR sites. This lag represents lost transactions in case of disaster and, as a result of that, a very real and significant risk of losing the consistency of—and between—the applications running the business of the organization. Prolonged downtime, permanent loss of transactions, and loss of reputation are only a few of the risks incurred by the organization as result of replicating asynchronously.

Axxana's Phoenix System for Oracle provides a solution to the above challenges, ensuring that no transaction data is ever lost and that all applications and databases are kept consistent during recovery to the latest state of the business right when the disaster hit. The result is that zero transactions are lost and the recovery process is significantly shortened, since there are no consistency issues.

The uniqueness of the Axxana solution for Oracle is that it is effective across any geographical distance between primary and secondary data centers; it works with any storage and any replication over any communication lines; and it can just as easily protect Exadata environments. ▲

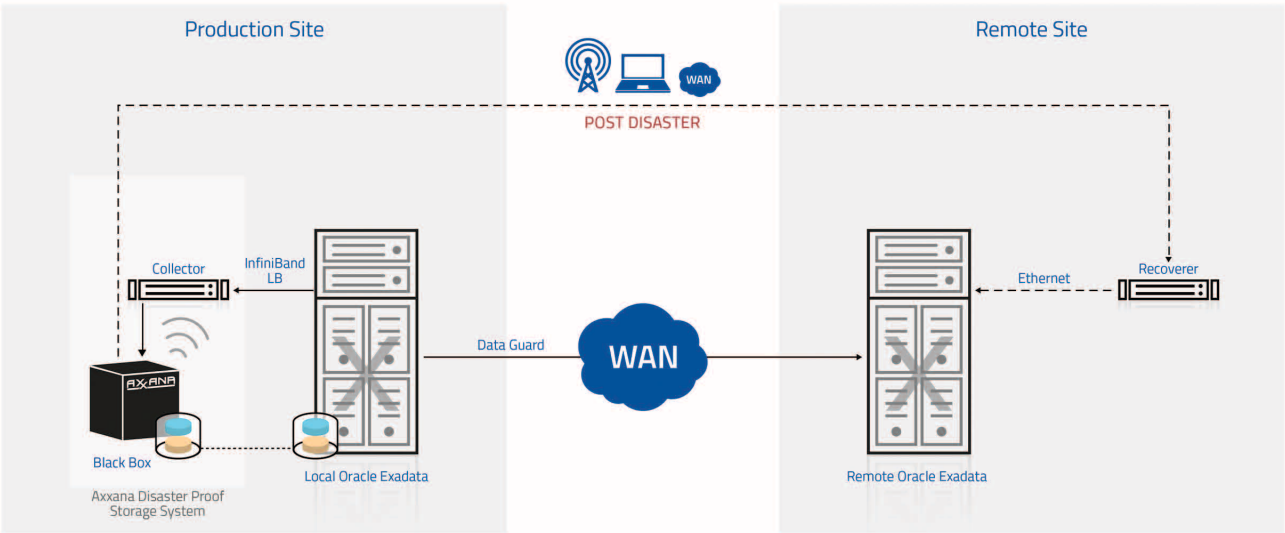


Figure 5. The Phoenix System for Exadata

Explaining the EXPLAIN PLAN

by Iggy Fernandez

Iggy Fernandez

Excerpted from blog posts on ToadWorld. You can read the entire series at **http://goo.gl/PCBKoR**.

A couple of years ago, I came across the following honest admission on AskTom:

"After 15 years development, implementation and administration of Oracle Databases I'm going to be totally honest.

1. *I have used Oracle 6-11*g
2. *I do understand the concepts of Oracle RDBMS databases*
3. *I do find PL/SQL 'relatively easy'*
4. *I have been following Tom Kyte in web and print since the mid 90s from his comp.databases.oracle days*
5. *I've always been a proponent of modular well written and well documented code*
6. *I've always been a big believer in as simple and logical a common sense approach as possible*

BUT

1. *I still find SQL very difficult*
2. *I still can't easily think in sets*
3. *I still can't really read an EXPLAIN PLAN with more than a few lines of indentation*
4. *I think I have used 'real' analytics about 3 times – after seeing it done in a specific example on this site*
5. *I would still bet that the majority of SQL writers out there would have no idea about or concept of query execution plans"*

The above comment motivated me to start collecting notes for a book on SQL that puts the query execution plan front and center. My plan is now to turn my notes into a series of blog posts with the goal of assembling them into a self-published book at some future date. Hop along for the ride; you'll soon be master of the EXPLAIN PLAN.

A Long Time Ago

Why do we read EXPLAIN PLANs? A history lesson is called for, because history connects us to our past and explains why we are what we are and why we do what we do. Imagine, then, a time before most of us were born, when magnetic spinning disks were becoming mainstream and online databases first became a reality. Prior to that time, data only existed on magnetic tapes. Either you read a tape sequentially—making changes as you went—or you read the entire contents of a tape into memory. It wasn't fun. Then magnetic spinning disks became mainstream and a whole world of fun possibilities opened up.

Charles Bachman enters from the left. "People," he says "Listen up! The days of sequential tapes are gone. You can still retrieve Employee records sequentially from an Oracle database if you need to do so or want to do so but now you can also retrieve a single Employee record by its primary key EMPLOYEE_ID or a unique key such as EMAIL, or you can retrieve multiple records using a non-unique key such as LAST_NAME. You can also retrieve a single Employee record by its ROWID if you know it."

Mr. Bachman must have had a time machine if he could predict the future name of Larry Ellison's company and the exact table and column names in the HR sample schema.

Mr. Bachman continues: "But wait! There's more! We can connect records in chains. Each chain has an owner record and zero or more member records linked together in a list. Every member record has a pointer to the previous and next member records—if any—and also a pointer to the owner record! For example, all the Employee records corresponding to a single DEPARTMENT_ID can be linked to the corrresponding Department record and all the Employee records corresponding to a single JOB_ID can be linked to the corresponding Job record."

Mr. Bachman walks to the chalk board—white boards and dry-erase markers had not yet been invented—and draws the diagram in Figure 1. Notice especially that records can be members of more than one chain; for example, the Employee record with EMPLOYEE_ID = 1 is a member of the chain owned by the Department record with DEPARTMENT_ID = 1 as well as the chain owned by the Job record with JOB_ID = 1. Every Depart-

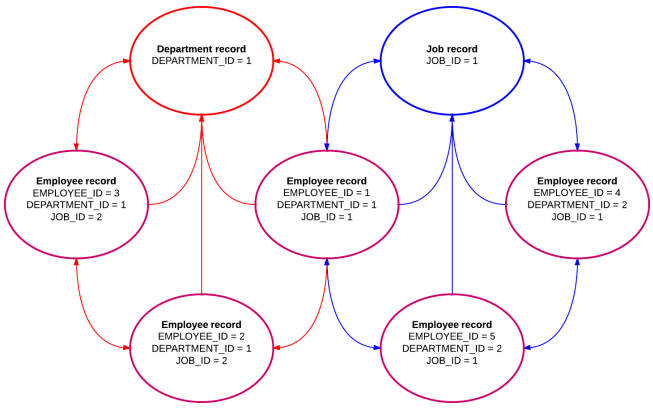


Figure 1

ment record would own a chain—possibly empty—of Employee records. Similarly, every Job record would own a chain—possibly empty—of Employee records. Every Employee record is therefore a member of two chains.

Mr. Bachman continues: "Therefore, in addition to the four retrieval methods I have already enumerated—sequential access, primary key access, non-unique key access, and ROWID access—there are three more. Having navigated to any record, one can navigate through all the records in a chain owned by that record, one can also navigate to the previous or next record in a chain in which the record participates, or one can navigate to the owner record of one of those chains. The programmer is therefore a navigator, navigating through a forest of records of different types."

The world cheered and Mr. Bachman received the ACM Turing Award, the equivalent of the Nobel Prize in the world of computers. His Turing Award lecture was titled "The Programmer as Navigator."

Suddenly, Dr. Edgar Codd enters from the right. "I object," he cries. "If you follow Mr. Bachman, you won't be able to see the forest for the trees! All you will see is individual records! Do you realize that Mr. Bachman did not use the word 'table' even once? He wants you to focus on individual records and the ways in which these records are connected! I say unto you that you are better off focusing on tables—that is, sets of records of the same type! You shouldn't have to remember what indexes exist and how records are connected. Focus on the tables, people!"

Dr. Codd must have been really passionate about relational theory if he used so many exclamation points.

The Impossible Dream

Dr. Codd was an academician, a scientist, a mathematician, a theoretician, and, above all, an idealist. Charles Bachman on the other hand was simply a manager of software development. Bachman created the very first database management system for his employer, General Electric (GE). Dr. Codd only created a theory that would take many years to be put into practice and, even today, has not yet been fully implemented. It seems that the relational model is truly an impossible dream.

Dr. Codd saw two uses for this theory. The first was "derivability and consistency of relations stored in large data banks," which he used as the title of his 1969 paper, the first paper on relational theory. Nobody understood what he meant, since—being a mathematician by training—he was prone to using gobbledygook that English-speaking people don't understand. He then changed the title to "A Relational Model of Data for Large Shared Data Banks" (1970) and focused on user convenience and productivity instead. That paper was reprinted in the 100th issue of the *NoCOUG Journal*; the NoCOUG volunteers lovingly retyped every word from photocopies, taking care even to reproduce typographical errors like the misspelling "Pheonix" in the References section.

Dr. Codd pressed his point in "Normalized Data Base Structure: A Brief Tutorial" (1971), in which he said, *"What is less understandable is the trend toward more and more complexity in the data structures with which application programmers and terminal users directly interact. Surely, in the choice of logical data structures that a system is to support, there is one consideration of absolutely paramount importance – and that is* ***the convenience of the majority of users.*** *[emphasis added] . . . To make formatted data bases readily accessible to users (especially casual users) who have little or no training in programming we must provide the simplest possible data structures and almost natural language. . . . What could be a simpler, more universally needed, and more universally understood data structure than a table?"*

And in his acceptance speech for the 1981 ACM Turing Award (the equivalent of the Nobel Prize in the field of computing), Dr Codd said: *"It is well known that the growth in demands from end users for new applications is outstripping the capability of data processing departments to implement the corresponding application programs. There are two complementary approaches to attacking this problem (and both approaches are needed): one is to put end users into direct touch with the information stored in computers; the other is to increase the productivity of data processing professionals in the development of application programs. It is less well known that a single technology, relational database management, provides a practical foundation to both approaches."*

Accordingly, the goals of the SQL language were user convenience and productivity. As explained by the creators of SQL in their 1974 paper "SEQUEL: A Structured English Query Language," there is *"a large class of users who, while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language. Examples of such users are* ***accountants, engineers, architects, and urban planners*** *[emphasis added]. It is for this class of users that SEQUEL is intended. For this reason, SEQUEL emphasizes simple data structures and operations."*

Historical note: The acronym SEQUEL was shortened to SQL because—as recounted by Chamberlin in "The 1995 SQL Reunion: People, Projects, and Politics"—SEQUEL was a trademarked name. This means that the correct pronunciation of SQL is "sequel" not the more popular "es-que-el."

And that's precisely why we need EXPLAIN PLAN. Pre-relational database management systems did not need an equivalent of EXPLAIN PLAN, because they used procedural languages: the programmer decided what the database management system should do and the database management system diligently followed suit. On the other hand, the SQL language used by relational database management systems is a non-procedural language. A SQL query specifies what data is needed but does not specify how to obtain it. The "query optimizer" then automagically constructs a query execution plan for us: the recipe, if you will. EXPLAIN PLAN is a listing of that recipe.

Secret Sauce

Dr. Codd's theories were an immediate sensation even though relational database management systems were still years away. It would be ten years before Larry Ellison created Oracle Database, and the first version did not even support transactions. That would take another five years—until 1983 to be precise. As IBM researcher Donald Chamberlin recalled later: *"[Codd] gave a seminar and a lot of us went to listen to him. This was as I say a revelation for me because Codd had a bunch of queries that were fairly complicated queries and since I'd been studying CODASYL, I could imagine how those queries would have been represented in CODASYL by programs that were five pages long that would navigate through this labyrinth of pointers and stuff. Codd would sort of write them down as one-liners. These would be queries like, 'Find the employees who earn more than their managers.' He just*

whacked them out and you could sort of read them, and they weren't complicated at all, and I said, 'Wow.' This was kind of a conversion experience for me, that I understood what the relational thing was about after that" (The 1995 SQL Reunion: People, Projects, and Politics).

The secret sauce of relational database management systems is composed of what the inventor of the relational model, Dr. Edgar Codd, referred to as "relational calculus" and "relational algebra." Once again, we have to forgive him for using complex-sounding mathematical terms that normal folk don't use around the water cooler. Relational calculus has nothing to do with the calculus you encountered in high school and college. A relational calculus expression is an English-like non-procedural specification of the user's query requirements: for example, "employees who have worked in all accounting job classifications." Relational algebra, on the other hand, is a step-by-step procedural recipe that details how to meet the user's requirements.

More precisely, relational algebra is a collection of operations that could be used to combine tables. Just as you can combine numbers using the operations of addition, subtraction, multiplication, and division, you can combine tables using operations like join, semi-join, anti-join, minus, intersection, and so on.

In fact, we can define a relational database as *"a database in which the data is perceived by the user as tables (and nothing but tables), and the operators available to the user for (for example) retrieval are operators that derive "new" tables from "old" ones"* (*An Introduction to Database Systems*, by Chris Date)

And, the whole reason that you and I are so interested in EXPLAIN PLAN is because it documents the sequence of relational algebra operations that Oracle Database used at run-time to execute any particular query; that is, it documents the query execution plan used by Oracle Database.

Relational Algebra and Relational Calculus

Just as you can combine numbers using the operations of addition, subtraction, multiplication, and division, you can combine tables using operations like "Selection," "Projection," "Union," "Difference," and "Join." Refer to Table 1 for the definitions.

We need an example. Let's use the HR sample schema that comes with Oracle Database. Let's use the above five operations to answer the question: Which employees have worked in all accounting job classifications, that is, those for which the job_id starts with the characters AC? The current job of each employee is stored in the job_id column of the employees table. Any previous jobs are held in the job_history table. Here is one of the ways in which this requirement can be expressed in SQL:

```
SELECT employee_id FROM employees
MINUS
SELECT employee_id
FROM
  (SELECT employees.employee_id,
    jobs.job_id
  FROM employees
  CROSS JOIN jobs
  WHERE jobs.job_id LIKE 'AC%'
  MINUS
    ( SELECT employee_id, job_id FROM job_history
    UNION
    SELECT employee_id, job_id FROM employees
    )
  )
```

Operation	Definition	SQL Implementation
Selection	Form another table by extracting a subset of the rows of the table of interest using some criteria.	SELECT * FROM [table] WHERE [criteria]
Projection	Form another table by extracting a subset of the columns of the table of interest. Any duplicate rows that are formed as a result of the projection operation are eliminated. This requires the use of the DISTINCT clause if the column list does not include columns that uniquely identify each row. A strict theoretical approach requires that SELECT DISTINCT always be used in place of SELECT but this would cause older versions of Oracle Database to perform unnecessary sorting so it was advisable to use the DISTINCT clause only when absolutely necessary. Newer versions of Oracle Database make an effort to avoid unnecessary sorting but the old advice still stands.	SELECT DISTINCT [column list] FROM [table]
Union	Form another table by selecting all rows from two tables of interest. If the first table has M1 rows and the second table has M2 rows, then the resulting table will have at most M1 + M2 rows. Duplicates are eliminated from the result.	SELECT * FROM [first table] UNION SELECT * FROM [second table]
Difference	Form another table by extracting only those rows from one table of interest that do not occur in a second table.	SELECT * FROM [first table] MINUS SELECT * FROM [second table]
Join (Cartesian)	Form another table by concatenating records from two tables of interest. If the first table has M1 rows and the second table has M2 rows, then the resulting table will have M1 * M2 rows, and, if the first table has N1 columns and the second table has N2 columns, then the resulting table will have N1 + N2 columns.	SELECT * FROM [first table] CROSS JOIN [second table]

Table 1. Five relational algebra operations

It's not very obvious. but the above SQL statement specifies a total of 11 relational algebra operations: one Selection operation, six Projection operations, one Union operation, two Difference operations, and one Join operation. This is clear in the following version, which uses "common table expressions" to express the requirement. Note that we do not need to use the DISTINCT clause in any of the projection operations.

```
WITH
  -- Step 1: Projection
  all_employees_1 AS
  ( SELECT employee_id FROM employees
  ),
  -- Step 2: Projection
  all_employees_2 AS
  ( SELECT employee_id FROM employees
  ),
  -- Step 3: Projection
  all_jobs AS
  ( SELECT job_id FROM jobs
  ),
  -- Step 4: Selection
  selected_jobs AS
  ( SELECT * FROM all_jobs WHERE job_id LIKE 'AC%'
  ),
  -- Step 5: Join
  selected_pairings AS
  ( SELECT * FROM all_employees_2 CROSS JOIN selected_jobs
  ),
  -- Step 6: Projection
  current_job_titles AS
  ( SELECT employee_id, job_id FROM employees
  ),
  -- Step 7: Projection
  previous_job_titles AS
  ( SELECT employee_id, job_id FROM job_history
  ),
  -- Step 8: Union
  complete_job_history AS
  ( SELECT * FROM current_job_titles
  UNION
  SELECT * FROM previous_job_titles
  ),
  -- Step 9: Difference
  nonexistent_pairings AS
  ( SELECT * FROM selected_pairings
  MINUS
  SELECT * FROM complete_job_history
  ),
  -- Step 10: Projection
  ineligible_employees AS
  ( SELECT employee_id FROM nonexistent_pairings
  )
-- Step 11: Difference
SELECT * FROM all_employees_1
MINUS
SELECT * FROM ineligible_employees
```

The sequence of subqueries in the above SQL statement hopefully doesn't need much explanation. An employee_id that occurs in a nonexistent pairing of an employee_id with a job_id is an ineligible employee. Eliminating all such ineligible employees from the list of all employees, therefore, yields the desired result: employees who have worked in all accounting job classifications.

The five relational operations in Table 1 are not only sufficient—as proved by Dr. Codd—to implement any requirement expressed in relational calculus but they are also "primitive," meaning that none of them can be eliminated without compromising sufficiency. In other words, none of them is a combination of the others. Composite relational operations can be devised by combining these five operations; examples include inner join, intersection, outer join, semi-join, anti-join, and division. Inner join is obviously a combination of Cartesian join and selection. Intersection can also be constructed using the operations in Table 1 as follows: A INTERSECTION B = A MINUS (A MINUS B). The Venn diagram in Figure 2 below makes it clear. I leave the rest as a research exercise for you.

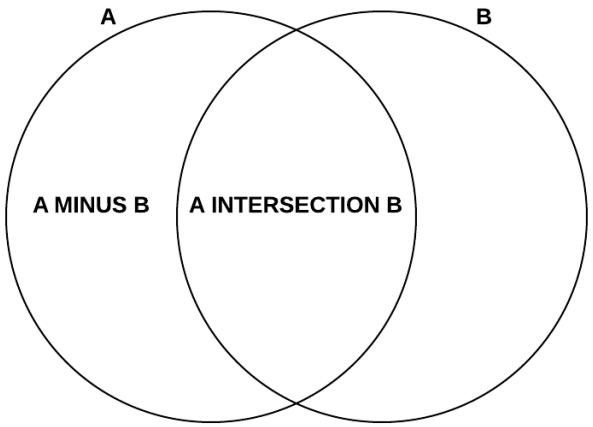


Figure 2

When writing SQL, we don't pay attention to relational algebra, and that's by design. Codd was of the opinion that *"[r]equesting data by its properties is far more natural than devising a particular algorithm or sequence of operations for its retrieval. Thus, a calculus-oriented language provides a good target language for a more user-oriented source language"* ("Relational Completeness of Data Base Sublanguages"). With the exception of the union operation, the original version of SQL was based on relational calculus, though; over time, other elements of relational algebra such as difference (minus), intersection, and outer join crept in. The following SQL statement is the more traditional way of stating the previous query. Note the use of the EXISTS keyword, which is the hallmark of relational calculus. This version can be paraphrased as "find employees such that there does not exist an accounting job classification which these employees have not held."

```
SELECT employee_id
FROM employees e
WHERE NOT EXISTS
  (SELECT job_id
  FROM jobs j
  WHERE job_id LIKE 'AC%'
  AND NOT EXISTS
    (SELECT *
    FROM
      ( SELECT employee_id, job_id FROM job_history
      UNION
      SELECT employee_id, job_id FROM employees
      )
    WHERE employee_id = e.employee_id
    AND job_id        = j.job_id
    )
  )
```

Trees Rule

An Oracle EXPLAIN PLAN is a "tree" structure corresponding to a relational algebra expression. It is printed in "pre-order" sequence (visit the root of the tree, then traverse each subtree—if any—in pre-order sequence) but should be read in "post-order" sequence (first traverse each subtree—if any—in post-order sequence; only then visit the root of the tree).

Note the recursive nature of the pre-order and post-order procedures: the phrase "pre-order" is used in the definition of the pre-order procedure and the phrase "post-order" is used in the

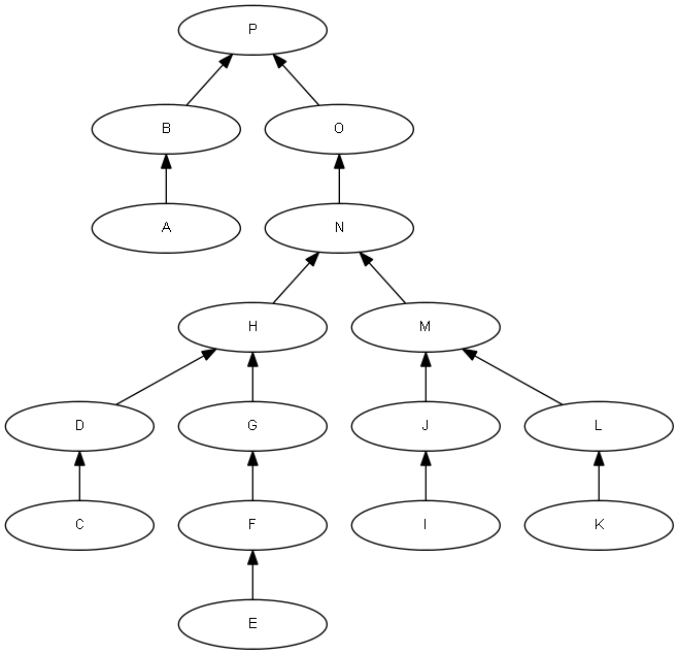


Figure 3

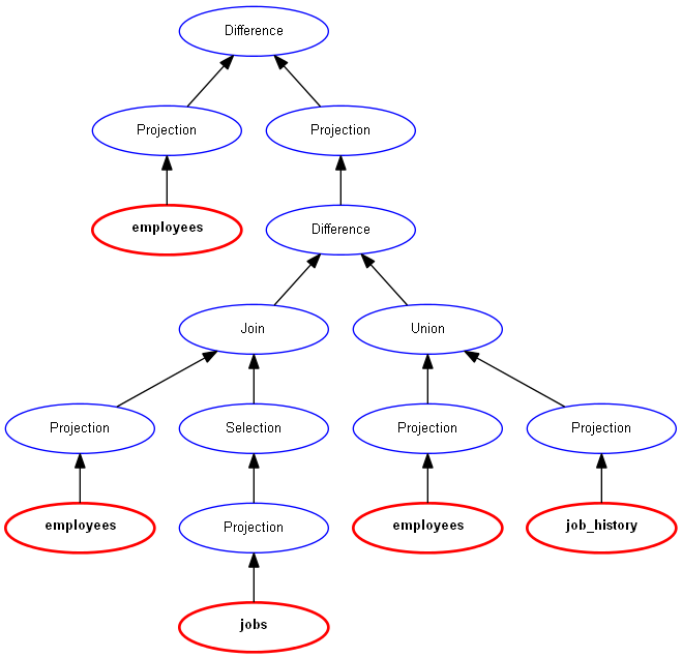


Figure 4

definition of the post-order procedure. In other words, the procedures are defined in terms of themselves.

Figure 3 shows an example of a tree structure.

The root of the above tree is labeled P. It has two subtrees whose roots are labeled B and O respectively. The subtree whose root is labeled B has a subtree whose root is labeled A. And so on. Read and re-read the definition of post-order traversal and convince yourself that the nodes should be visited in alphabetical order; the node labeled A should be visited first, the node labeled B should be visited next, and so on. The root note should be visited last.

Now look at the tree in Figure 4. The single-character labels have been replaced with the names of tables and operations, but this does not affect the order in which the nodes should be visited. The red-colored terminal nodes ("leaf nodes") represent data, while the blue-colored non-terminal nodes (non-leaf nodes) represent operations. Non-terminal nodes with just one subtree represent "unary" operations such as Selection and Projection, while those with left and right subtrees represent "binary" operations like Union, Difference, and Join. The fact that the subtrees are trees in their own right means that operations can be nested. In other words, a query execution plan corresponds to a nested relational algebra expression.

The tree in Figure 4 is a hypothetical query execution plan for the common table expression (CTE) solution to our teaching example, "Which employees have worked in all accounting job classifications: those for which the job_id starts with the characters AC." Count the blue-colored non-terminal nodes in the above tree; you should find exactly eleven of them. That's because there are exactly eleven steps in the CTE solution. Figure 5 illustrates how the desired result is produced. Note especially that each blue-colored terminal node represents an intermediate table. In fact, we can define a relational database as "a database in which: the data is perceived by the user as tables (and nothing but tables) and the operators available to the user for (for example) retrieval are operators that derive 'new' tables from 'old' ones" (*An Introduction to Database Systems*, by Chris Date).

Every query execution plan corresponds to a nested relational algebra expression, because relational algebra operations are the basic procedures used by Oracle Database after all. The following expression corresponds to the trees shown in Figures 2 and 3. The uppercase abbreviations S, P, U, D, and J correspond to the operations Selection, Projection, Union, Difference, and Join respectively, while the lowercase abbreviations e, j, and jh correspond to the tables employees, jobs, and job_history respectively. The outermost operation in the expression corresponds to the root of the tree.

```
D(P(e),P(D(J(P(e),S(P(j))),U(P(e),P(jh)))))
```

The relational algebra expression corresponding to the trees shown in Figures 4 and 5 can also be written as follows, though the correspondence with the trees gets obfuscated. As used here,

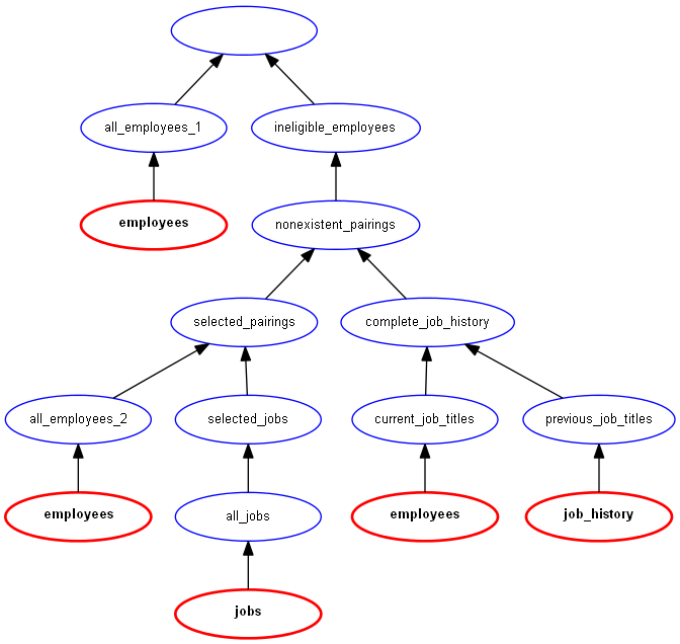


Figure 5

the symbols "-", "*", and "U" represent the Difference, Join, and Union operations respectively.

```
P(e)-P(P(e)*S(P(j))-(P(e)UP(jh))
```

Summary

This article is excerpted from blog posts on ToadWorld. You can read the entire series at **http://goo.gl/PCBKoR**.

- A relational database is *"a database in which: the data is perceived by the user as tables (and nothing but tables) and the operators available to the user for (for example) retrieval are operators that derive 'new' tables from 'old' ones"* (*An Introduction to Database Systems* by Chris Date).
- SQL is a non-procedural language: a SQL query specifies what data is needed but does not specify how to obtain it. The "query optimizer" automagically constructs a query execution plan for us.
- The query execution plan can and does change when the inputs (values of bind variables, data distribution statistics, etc.) change. This comes as a great surprise to almost everybody, but that's how it was always intended to work.
- A huge problem with relational databases is that semantically equivalent statements do not result in the same runtime query execution plan. That's not how it was ever intended to work.
- The EXPLAIN PLAN documents the query execution plan used by Oracle Database; that is, it documents the sequence of relational algebra operations that Oracle Database uses at run time to execute any particular SQL query.
- An EXPLAIN PLAN is a "tree" structure corresponding to a relational algebra expression. It is printed in "pre-order" sequence (visit the root of the tree, then traverse each subtree—if any—in pre-order sequence), but it should be read in "post-order" sequence (first traverse each subtree—if any—in post-order sequence, and only then visit the root of the tree). ▲

The statements and opinions expressed here are the author's and do not necessarily represent those of Oracle Corporation.

SQL Mini-Challenge

The inventor of the relational model, Dr. Edgar Codd, was of the opinion that *"[r]equesting data by its properties is far more natural than devising a particular algorithm or sequence of operations for its retrieval. Thus, a calculus-oriented language provides a good target language for a more user-oriented source language"* (Relational Completeness of Data Base Sublanguages). Therefore, with the exception of the Union operation, the original version of SQL was based on relational calculus, although, over time, other elements of relational algebra like difference (minus), intersection, and outer join crept in.

Testing of existence in a set using subqueries is the hallmark of relational calculus. The following example has nested subqueries. It lists the locations containing a department that either contains an employee named Steven King or an employee who holds the title of President or or employee who has previously held the title of President.

```
SELECT l.location_id, l.city
FROM locations l
WHERE EXISTS (
  SELECT * FROM departments d, employees e, jobs j
  WHERE d.location_id = l.location_id
  AND e.department_id = d.department_id
  AND j.job_id = e.job_id
  AND (
    (e.first_name = 'Steven' AND e.last_name = 'King')
    OR j.job_title  = 'President'
    OR EXISTS (
      SELECT * FROM job_history jh, jobs j2
      WHERE jh.employee_id = e.employee_id
      AND j2.job_id = jh.job_id
      AND j2.job_title = 'President'
    )
  )
);
```

The challenge is to rewrite the above query without subqueries and in the most efficient way possible, as measured by the number of consistent gets; that is, the Buffers column in the query execution plan. Answers must be posted on the NoCOUG blog **http://nocoug.wordpress.com**. Answers must be tested in the HR sample schema to prove that they actually work and must be accompanied by a query execution plan showing the number of consistent gets used by the rewritten query. The winner will receive a $75 Amazon gift certificate or a prize of equal value. The contest will close when a sufficient number of entries have been received. NoCOUG's decisions are final. ▲

YesSQL!

At Oracle OpenWorld 2014, Oracle Technology Network will host a special event titled "YesSQL! A Celebration of SQL and PL/SQL." Co-hosted by Tom Kyte and Steven Feuerstein, YesSQL! celebrates SQL, PL/SQL, and both the people who make the technology and those who use it. At YesSQL!, special guests Andy Mendelsohn, Maria Colgan, Andrew Holdsworth, Graham Wood, and others will share their stories of how SQL affected their lives and invite you to share yours. ▲

Simplicity Is *Good*!

by James Morle

James Morle

This article is about the importance of *appropriately simplistic* architectures. I frequently get involved with the creation of full-stack architectures—in particular the architecture of the database platform. There are some golden rules when designing such systems, but one of the most important ones is to keep the design as simple as possible. This isn't a performance enhancement; this is an *availability* enhancement. Complexity, after all, is the enemy of availability.

Despite it being a sensible goal, it is incredibly common to come up against quite stubborn resistance to simplicity. Frequently, the objections will be based upon the principles of the complex solution being a "better way" to do things. I have two closely linked examples of this in action.

Case 1: Real Application Cluster Interconnects

A cluster interconnect is an incredibly important component of the architecture. The cluster exists, after all, as an availability feature (and possibly a scalability feature), and so the foundations of the cluster must be robust in order for it to deliver that availability. The cluster interconnect is the lifeblood of the cluster. And yet, it has such a very simple set of requirements:

- Point-to-point communication between all nodes of the cluster
- Low latency
- n+1 availability of network paths
- Multicast support between the nodes
- (optionally) Jumbo frames support

It explicitly does not need any of the more "fancy" networking features, such as:

- Routing of any kind
- Spanning tree support
- VLANs
- *Access to any other networks*

It just needs a dedicated pair (or more) of discrete layer-2 networks. They don't need to be bonded; the networks do not even need to be aware of each other—they are completely independent (at least, that is certainly the case since the HAIP functionality of Oracle 11*g* Release 2). They *do* need real switches, though—crossover cables fail ungracefully in the event of a peer host losing power. But they don't need anything high end—just something better than crossover cables and with enough bandwidth for the required traffic rates. The latency difference between the majority of switches is barely a consideration. The switches don't really even need redundant power supplies, though it's not a terrible idea to insulate yourself from this type of failure, and it brings no detriment apart from a marginal cost increase.

So, something like a pair of *unmanaged* layer-2 GbE Ethernet switches are the perfect solution. Something like a Netgear JGS516 would probably do the job, from a brief scan of the specification. They are about $166 (£100) each, net cost of $332 (£200) for a nice, robust solution. Or if you wanted to really push the boat out, something like a fully managed L2 switch with redundant power such as the HP E2810-24G will set you back all of $1160 (£700) each. Cisco shops might spend a bit more and go for something like a 3750 G for about $4650 (£2800) each.

But . . . somebody will always push back on this. They will plumb the cluster nodes into the full core/edge corporate dream stack topology with fully active failover between a pair of core switches. Surely, at a cost of more than four orders of magnitude more than the bargain basement Netgear solution, this must be better, right? Wrong.

There are numerous aspects that are incorrect in this assumption:

a. Higher cost means better.
b. There will be an increase in availability.
c. Every networking requirement is the same as every other one.

First of all, these network topologies are not designed for cluster interconnects. They are designed for corporate networks, connecting thousands of ports into a flexible and secure network. RAC interconnects are tiny closed networks and need none of that functionality. More precisely, they need none of that *complexity*. Corporate networks also have a different level of failure sensitivity to cluster interconnects; if a user's PC goes offline for a couple of minutes, or even half an hour, the recovery from that failure is instant once the fault is rectified—the user is immediately back in action. Cluster interconnects are not so forgiving; if a cluster's networks go AWOL for a few minutes, the best you can hope for is a single node of the cluster still standing when the fault is rectified. That is how clusters are designed to operate: if the network disappears, the cluster must assume it is unsafe to allow multiple nodes to access the shared storage. The net result of this failure behavior is that a relatively short network outage can result in a potentially lengthy full (and manual) restart of the whole cluster, restart of the application, balancing of services, warming of caches, and so on. It would not be an exaggeration for this to be a one-hour or greater outage. Not terrific for a highly available cluster. *(continued on page 26)*

Many Thanks to Our Sponsors

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

Long-term event sponsorship:

Chevron

Oracle Corp.

Thank you! Gold Vendors:

- Axxana
- Confio
- Database Specialists
- Delphix
- Embarcadero Technologies
- Kaminario

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

TREASURER'S REPORT

Eric Hutchinson, *Treasurer*

Beginning Balance		
January 1, 2013		**$ 56,473.82**
Revenue		
Membership Dues	42,707.70	
Training Day Receipts	23,254.36	
Vendor Receipts	22,558.73	
Conference Sponsorships	3,000.00	
Interest	7.72	
Total Revenue		**$ 91,528.51**
Expenses		
Administration	7,371.73	
Regional Meeting/Conferences	34,487.55	
Board Meeting	4,050.02	
Journal	15,600.65	
Membership	3,875.29	
Training Day Expenses	18,693.49	
Insurance	507.00	
Vendor Expenses	3,488.18	
Servers/Software Hosting	3,011.69	
Total Expenses		**$ 91,085.60**
Ending Balance		
December 31, 2013		**$ 56,916.73**

(continued from page 24)

But hang on a minute—this über-expensive networking technology never goes down, right? Not true. What exactly is this active/active core switch topology? Think about it. It's a kind of cluster itself, with each switch running complex software to determine the health of its peer and managing a ton of state information between them. The magic word in that sentence was the word "software"— anything that is running software has a great deal of failure potential. Not only that, but clustered software has a great deal of potential to fail on *all nodes concurrently*. This is a unique attribute of distributed software and one that does not exist in discrete hardware designs. In discrete hardware designs it is incredibly unlikely that more than one component will fail concurrently. Software is great at catastrophic failure, most particularly when it is combined with some element of human error during upgrades, reconfiguration, or just plain tinkering. Not even humans can make two independent hardware switches fail concurrently, unless they are being creative with power supply.

Just to highlight this point, I should state here that I have personally witnessed failures of entire core/edge switch topologies on three occasions in the last five years. It does not matter that the cluster nodes are connected to the same edge switches when this kind of failure occurs, because every component in the network is a logical contributor to the larger entity and will become unavailable as part of a larger meltdown. If you are a Blackberry user, you have experienced one yourself recently. The Blackberry issue proves the potential, but in their case the topology was at least appropriate—they have a requirement to interconnect thousands of devices. In our clusters, we have *no such requirement*, and we should not be implementing overly complex and thus unreliable network topologies accordingly.

Case 2: The Great SAN Splurge

Now let's think about Storage Area Networking. And let's not restrict this thought to Fibre Channel, because the same principles apply to an Ethernet-based SAN. In fact, let me just clear off the Ethernet SAN piece first: Don't use your corporate network for storage connectivity. It's the wrong thing to do for all the reasons stated in the first case on this page.

So, now we can focus on Fibre Channel SANs. Fibre Channel has become the backbone of the data center, allowing storage devices to be located in sensible locations, perhaps in different rooms to the servers, and for everything to be able to be connected to everything else with optimized structured cabling. The zoning of the fabric then determines which devices are allowed to see other devices. All very well and good, but how is this implemented? Unsurprisingly, it is implemented using an exactly analogous solution to the core/edge Ethernet network design in the previous case. Two active core switches lie at the heart of a multi-tier network and provide failover capability for each other. A cluster. This cluster can (and does) fail for exactly the same reasons given in the former case, and yes, I have also seen this occur in real life—twice in the last five years.

The failure implications for a SAN meltdown can be even more serious than a cluster meltdown. *All* I/O will stop and, if the outage goes on long enough, *all databases in the data center* will crash and need to be restarted.

There are a few other implications with this topology in large data centers. Notably, it is common for the storage arrays to be connected via different physical switches than the servers, implying that there are a number of Inter-Switch Links (ISLs) to go through. These ISLs can become congested and cause severe bottlenecks in throughput that can be extremely tricky to track down. In extreme cases, ISLs can be the cause of *multi-minute* I/O response times, which will also cause clusters and databases to crash.

So that preamble paints the SAN picture and sets the stage for the following questions:

Why are all devices in the SAN connected to all other devices? Why are the handful of nodes that make up your critical database part of a SAN of thousands of other devices? Why are they not just connected via simple switches to the storage array?

There is only one reason and that is data-center cabling. But it doesn't really follow: If your database servers are in a rack, or a few racks next to each other, put a pair of physically and logically discrete switches into the top of the rack, attach all the nodes, and then connect the storage array using the same number ports that you would have connected to the switches if they had been edge switches. The destination of those cables would be the storage array rather than the core switches, but the number of cable runs is pretty much the same and results in a more robust solution. There is no exposure to catastrophic loss of service in the SAN, because there are two completely discrete SANs between the servers and the storage.

Fibre Channel networks are *vertical* in nature: server nodes do not communicate with other server nodes over the SAN; they only communicate with the storage array. Server nodes do not need to be connected to thousands of storage arrays, either. The connectivity requirement for a given platform is actually rather simple.

Note: I am writing from the viewpoint of a typical RDBMS implementation, not from the viewpoint of massively parallel HPC or big data systems. Clearly, if there truly are thousands of devices that *do* need to be connected, this argument does not apply.

Conclusion

The common theme between these two cases is this: Don't connect things that don't need to be connected. Yes, it is easier to cable up, and arguably easier to manage, but it has a knock-on effect of dictating an implementation that does not suit the requirement. It results in a less reliable, more complex solution, with the cart very much before the horse. Don't trade off administrative simplicity against architectural simplicity: it will sneak up and bite you.

As Albert Einstein said, *"Make things as simple as possible, but not simpler."* Wise words indeed. ▲

James Morle is a specialist large-scale Oracle-based systems consultant with over 20 years experience in IT consulting. His success is based on two key factors: deep domain expertise in the full Oracle stack (he speaks every language) together with huge credibility with senior IT and business leaders. James is one of the co-founders of the OakTable Network and an Oracle ACE Director. He is also the founder of Scale Abilities Ltd., based in the UK and serving clients across the world. Scale Abilities solves implementation, storage, and performance problems with large, complex Oracle-based systems. International companies rely on its extensive full-stack expertise, which successfully solves the wide range of cross-boundary problems that single-discipline consultancies and experts struggle to handle. Scale Abilities consultants will tell you what others are afraid to share, and they understand how to unpick politics from performance.

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.

DatabaseSpecialists

CALL 1-888-648-0500 TO DISCUSS A SERVICE PLAN

NoCOUG
P.O. Box 3282
Danville, CA 94526

RETURN SERVICE REQUESTED

NoCOUG Summer Conference Schedule

Thursday, August 21, 2014—Chevron, 6101 Bollinger Canyon Road, San Ramon, CA

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

8:00–9:00 a.m.	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Hanan Hit, NoCOUG president
9:30–10:30	**Keynote:** *The Internet of Things*—Shyam Varan Nath, General Electric
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Room 1220: *Oracle Clusterware Deep Drive*—Amit Das, PayPal
	Room 1240: *OMG! Identifying and Refactoring Common SQL Performance Anti-patterns*—Jeffrey Jacobs
	Room 1150: *Surviving Data Corruption*—Lorrie Yang, Bank of America
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Room 1220: *Unit of Work Time Based Analysis*—Craig Shallahamer, OraPub
	Room 1240: *Introduction to Vertica Column Store for Data Warehouse Developers*—Dave Abercrombie, Tapjoy
	Room 1150: *MapReduce by Example – Hands-on Lab*—Arijit Das, Naval Postgraduate School
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Room 1220: *Creative Oracle Database 12c Redo Maneuvers*—Craig Shallahamer, OraPub
	Room 1240: *Twelve Things About SQL Every Oracle Professional Should Know – Part I*—Iggy Fernandez
	Room 1150: *MapReduce by Example – Hands-on Lab*—Arijit Das, Naval Postgraduate School
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Room 1240: *Twelve Things About SQL Every Oracle Professional Should Know – Part II*—Iggy Fernandez
4:00–	**NoCOUG Networking and No-Host Happy Hour at Hopyard American Alehouse & Grill, San Ramon**

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

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo: Rose at twilight by Dave Abercrombie.