Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 28, No. 1 · FEBRUARY 2014 $15

Knowledge Happens

One Graph to Rule Them All

We interview Neo4j CEO Emil Eifrem.

See page 4.

Graph Databases

Relational warhorse Joe Celko expounds.

See page 8.

Oracle Spatial *and* Graph

Oracle Database 12c has it all.

See page 15.

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 109th issue of the *NoCOUG Journal*. Enjoy! ▲

—*NoCOUG Journal* Editor

Table of Contents

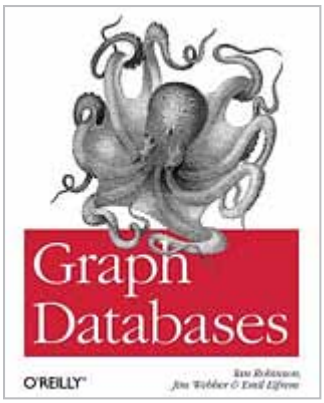
ADVERTISERS

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.



One Graph to Rule Them All

with Emil Eifrem

Emil Eifrem

Emil Eifrem is co-founder and CEO of Neo Technology and the Neo4j open-source graph database project.

From the relational fortress in which NoCOUG members live, NoSQL doesn't seem very threatening or even compelling. Are we living in a dream world? These walls seem pretty solid though.

That's a great question. Exponential growth looks very flat in the beginning. The NoSQL numbers are still small compared to the relational world, much in the same way that the relational numbers were very small compared to IMS and CODASYL when relational first came into the arena. One thing that makes today's trend different of course is that now we're really talking about multiple database models being used in tandem with existing RDBMSs, to address particular types of data problems.

We rarely see people moving entirely off of one system and onto another just because they have a new requirement. Normally that doesn't make sense. On the other hand, if you look at unconstrained greenfield projects where technology choices are made based purely on the technical and business requirements—and where you have no technology inhibitions—I will bet that the numbers will lean very favorably towards solutions that include some alternatives to the relational model. Of course in those cases as well, relational databases are often a part of the solution.

So, should developers feel threatened? No, I don't think there's any sense of threat. Rather, I think there's an opportunity to leverage new technologies alongside the familiar ones, to create more value for the business. There is a huge wealth of options available today when you look at data and database management. By understanding the strengths and weaknesses of each tool, you will see what makes the most sense for a given and use that knowledge to your advantage.

Some of the criticisms of NoSQL databases seem quite valid. Lack of management and monitoring tools, limited feature sets, limited security, and traps for the unwary. I just discovered that MongoDB uses a database-wide lock for every write . . .

Many of the NoSQL databases are really new to the scene, and it's taken relational databases many years, if not decades, to get to where they are today. The RDBMS is a really mature, well-understood, battle-proven technology. So I generally agree with that statement. We have been working on Neo4j since 2000, which is why we have a completely ACID transactional core, unusual in NoSQL land. This is one reason we don't have a global write lock! We have the equivalent of row-level locking, which means that if you write to two different nodes in the graph, you can run those write operations in parallel. We do this through a variant of MVCC, allowing you to read an old version of a property on a node while another write transaction is modifying it.

Our database kernel, granted, is more mature in such areas than most other NoSQL databases (including other graph databases) because unlike most of those others, we've been at this for the past 13 to 14 years. That said, the ecosystem is still very young, and there are certain areas that haven't yet been as much of an R&D focus, such as matching the elaborate and fine-grained security models you get with RDBMSs. The main takeaway here is that while they're different and maybe not as mature in all areas, they are already far enough along to have a solid production track record in some of the most mission-critical applications you can imagine (such as routing trains and packages in real time, for example). I actually think one of the bigger challenges is that the industry as a whole is largely still getting its head around how to use graph databases, and so sometimes there is confusion when people assume graph databases behave like others in the NoSQL categories. We are quite different in fact. Slowly, though, people are learning, and word is getting out. Graph databases are quickly becoming recognized as being their own battle-proven technology. I think that's just the nature of a new technology.

Graph databases seem very different from other NoSQL categories. There is no sharding, replication, eventual consistency, or schemaless design. Or is there?

Great question. I agree with the overall statement that graph databases are highly differentiated. There is some overlap between key value stores and document databases, and there is some overlap between document databases and column family, as well as key value store and column family, etc. They all solve broadly similar problems. But graph databases by and large solve a very different problem from all of the other categories in NoSQL: they're built to help applications quickly and easily navigate complex and evolving systems of densely connected data, as opposed to simply managing chunks of discrete data.

Having said that, Neo4j does include some of the things that you mentioned up front. It has a high-availability clustering mode that is based on replication. It has a schema-free model, which means that you can add an arbitrary amount of properties to one node. And you can also add and remove relationships arbitrarily between nodes, to reflect whatever data is coming in—without having to predefine or guess ahead of time how your data is really going to connect. Therefore graph databases as a category differentiate themselves from the other NoSQL categories first and foremost by their data model.

However Neo4j specifically has also been a bit of a NoSQL contrarian in terms of our architecture choices. The fact that Neo4j is built on an ACID transactional core is really important—and contrarian. In Neo4j 2.0, we started adding optional schema, which is again contrarian. We play a lot of tricks to ensure that queries never get bogged down by network hops—culminating in a scale-out approach that lets you shard data in cache without sharding it on disk. This again is quite contrarian in the NoSQL world. The last area in which we're contrarian actually makes us closer in a sense to relational databases. It's the fact that Neo4j not only acknowledges relatedness between data items—which by the way is an anti-pattern in the other NoSQL databases—but also goes a step beyond even relational databases by lifting up the connections to be first-class citizens. It does this first by allowing the relationships (not just the nodes) to have an arbitrary number of properties. This lets you ascribe weights, for example, to friendships as well as start and end dates, etc. This is quite powerful. The other thing that's different is that relationships are database objects that each connect two actual pieces of data. So rather than defining how types of data relate, you create a relationship between the actual bits of data, with pointers on either end linking the nodes, and a database engine that's capable of traversing it in either direction. In this way, every piece of data that has a link to another piece of data has a relationship between them. The query-time benefits are astounding. This lets queries move very quickly across the graph, following connections through in-memory pointer chasing and entirely bypassing join operations. There's no need for the database to traverse down the index's b-tree or sideways via a range scan with each hop to determine whether data is related. The result is queries that are 1000x faster in many cases compared to their SQL equivalent, with much more compact queries.

What was the specific situation or interesting customer story that led you to create Neo4j?

I was working at an enterprise content management startup in Sweden. I was the CTO and had 20 or so engineers working on the product, half of which spent the vast majority of their time just fighting with the relational database. It took us a while to figure out why. But when we did the root-cause analysis to determine why we were struggling so much, we saw that the real reason things were going so slowly was that there was a mismatch between the shape of the data that we were dealing with in the application and the abstraction that the relational database exposed. The relational database exposed a tabular view of the world. But our enterprise content management system was filled with hierarchical and connected data. Content was stored in a big tree, with symbolic links or shortcuts pointing from item to item, all of which led to a big connected messy data structure, which was very difficult to squeeze down in a relational database. The mismatch was not unlike the object-relational mismatch that we all know about, only worse, because all of the objects were connected in a highly unpredictable way. After a while we realized that we could represent all of it as a network or a graph. At the time we thought that there must be a database that persists data in the same way as a relational database but just exposes a different data model. We wanted a database that exposed a graph data model rather than a tabular data model. So we started looking around for that, but couldn't find anything. Ultimately we decided to build it. How hard could it be, we thought! Now, 14 years later, we're still working to evolve the graph database, which has many of the properties of a relational database, married with a graph data model.

How does Neo4j work under the covers? Is "j" for Java?

The "j" is for JVM. Neo4j written on top of the Java Virtual Machine, which has a bad rep in some camps but in my book is one of the finest pieces of engineering that the software industry has ever produced. It's deployed virtually everywhere in the world. Wherever we run software, the Java Virtual Machine is running. Neo4j is not Java-exclusive in any way: you can use it from any language. We have a declarative query language called Cypher, which you, of course, can execute from .NET, Ruby, PHP, Java, or the Unix console, as well as from our browser-based query tool. However, internally it's written primarily in Java with a bit of Scala. Neo4j has native drivers for all of the popular languages and an HTTP endpoint that can be used to run queries in Cypher.

What are some of the ways that your customers have been using Neo4j?

Let me count the ways! Certainly, the original use case is one that we see a good deal of. In that example you have the graph of identities: users, groups, groups of groups, and so on. Colloquially we call this a "hierarchy." (Actually a hierarchy is just a specialized type of graph where the relationships all go up and down.) But often in the real world a hierarchy is not just a hierarchy. Data can often relate through sideways connections, and nodes can sometimes connect upward to multiple parents. Any such line breaks the strict hierarchy, turning it into a generalized graph. Going back to the use case, content looks very much the same. You have the individual piece of content and then collections of content and then collections of collections (which can often also contain content directly), and so on. Now you interleave those two mostly hierarchical graphs with relationships between the identity graph and the content graph, which is how you express permissions. This gives you a tangled web, which is hard to model and even harder to query. This pattern is the perfect example of where graph databases are a great fit, and it comes up in a lot of different applications.

For example, let's now take network and data center management. The nodes in the network are the physical and virtual machines, the software assets, and the processes running on those machines. These are each different graphs, which relate to each other . . . and at the lowest level they are tied together by physical and virtual networks. Moving on, there's the original graph use case, which is geo. Leonhard Euler, who invented graph theory back in the early 1700s, first used his new theory to solve a geo-routing problem. You would therefore expect there to be (and you would be right) a number of logistics companies that have discovered graph databases and are using them

to complement their existing relational infrastructure to handle routing, from packages to trains and so on. Facebook of course popularized the social graph, and that's quite a common use as well: both for declared relationships, and inferred ones, that you arrive at by hopping through, say, common interests. Master data management (of "hierarchical" data) comes up a lot: organizational hierarchy, product master, etc. I can show you a typical HR query that we got from a consultant that takes up more than one page in SQL and just four lines in Neo4j's Cypher query language.

> ***"It is very likely that there are vastly more graph problems out there than most people think. We now see valid, actual uses across most verticals."***

Generally speaking, if you're finding yourself doing lots of recursive joins or joining across lots of join tables or running queries of unknown/arbitrary path-length, like ShortestPath (aka the "Kevin Bacon" query) or chasing up or down a graph to find all of the dependencies, then graph databases can help relieve some pain. Besides the performance advantages, many developers just find that graphs are an elegant way to frame their problems when dealing with connected data. So to summarize: adoption has been in a wide range of verticals, mostly alongside relational databases and sometimes alongside other NoSQL databases, to tackle the various challenges one bumps up against with connected queries and connected data.

You're surely not suggesting that graph databases be considered for non-graph problems? In* Normalized Data Base Structure: A Brief Tutorial *(1971), Dr. Codd carved out a special exemption for what he called "network problems," but for the general case, 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. . . . 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?"*

Correct, that would certainly be ridiculous. However, it is very likely that there are vastly more graph problems out there than most people think. We now see valid, actual uses across most verticals where people use software: anywhere from financial services to telecom to media and publishing to health care to automotive—horizontally across the board. Businesses are faced with the need to understand intricately connected real-world systems across a variety of domains: social networks, biological networks, logistics networks, and so on. The ability to leverage connected data is a horizontal concern, and I believe that by the end of this decade most systems that use software will touch a graph database in some way.

NoCOUG users already have it pretty good with Oracle Spatial and Graph. How is the Neo4j approach different from the Oracle approach?

I'm afraid I can't comment, as I don't know the inner workings of Oracle Spatial and Graph. I have to say however, it did put a smile on my face when I read Oracle's blog post stating that Oracle Spatial was being renamed to Oracle Spatial and Graph because of the growing popularity of graph databases! What I can tell you about Neo4j is that it's written from the ground up to be optimized for storing and querying graphs, and it is a native graph database. This means that every layer in the stack was designed to make storing and retrieving graph data highly performant and scalable.

What new in Neo4j? What's on the roadmap?

We started working on Neo4j in 2000, and it took us a decade to release Neo4j 1.0 in 2010. We're Swedes and have some pretty high standards when it comes to building a system that you can trust with your data. The majority of our focus in the Neo4j 1.x series was to continue to improve performance, scalability, and robustness. Clustering, for example, was added very early on in the 1.x series. Fast-forward to last year: all of 2013 was spent working on Neo4j 2.0, which was focused almost entirely on ease of use. We released Neo4j 2.0 at the end of last year, adding several key features to make graph databases easier to use. Prior to 2.x, we did a pretty good job of building a scalable and robust database that solves some very important technical and business problems, but we hadn't yet made it very easy to use. All of that changed with Neo4j 2.x. We believe the 2.x series is going to catapult graphs into the mainstream.

Going forward, one thing I am particularly excited about is something we informally call a SQL gateway. It's an automated way of replicating data between Neo4j and a relational database. What we have seen out in the field is that many times graph-y data is trapped in a two- or three-column table in a relational database (e.g., person ID, related person ID). It's actually fairly easy to replicate data like this into Neo4j bidirectionally, and people do this all the time. It's extra work to do it yourself though, and because it's such a common pattern, we want to make it easier. It will allow people who have an existing Oracle installation, where some aspects of their data set is a graph, to more easily take that portion of the data model and replicate it into Neo4j. The result is that you keep all of your data in Oracle but add the power of a graph by replicating relevant parts of the data set into Neo4j. This pattern of "polyglot persistence" is one I think we'll be seeing more and more of in the years to come—not just with graphs but with databases in general. As far as the rest of the roadmap, it's continuing to improve what Neo4j already does well, with a special focus on scalability and ease of use. ▲

Member IDs 6322478, 6510532, 6527185, 7082228, 9094728, and 9224454 have won a prize: *The 7 Habits of Highly Effective People* by Stephen Covey. ***Must attend the winter conference to collect your prize.***

How to Become an Expert

Conference #108 at TechMart in Santa Clara had 181 attendees, more than any board member can remember except at Conference #100 at Computer History Museum.

If you want to become an expert, you must learn from one!

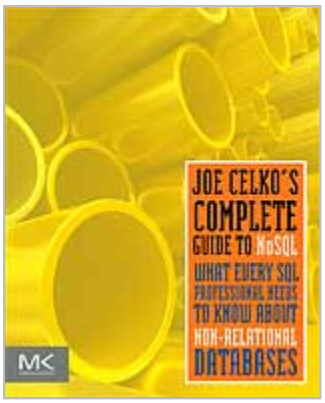
Look, Ma, I won a toaster! Kamran Rassouli photo-bombs the candid moment.

The Winner's Circle with NoCOUG president Naren Nagtode and vice-president Hanan Hit.

Only at NoCOUG will the Director of Managed Services of Database Specialists become your personal IT trainer!

From this angle, it looks like Confio's "Looney Tuner" Janis Griffin is rummaging through keynote speaker Tom Kyte's backpack while board member Randy Samberg distracts him at the wine-and-cheese reception sponsored by Fusion-io. Funny camera angles are a terrible thing to waste!

Even the photographer won a prize. High five, Ahbaid!



Graph Databases

from *Joe Celko's Complete Guide to NoSQL*

Joe Celko

Reprinted with permission from Joe Celko's Complete Guide to NoSQL. *DOI: 10.1016/B978-0-12-407192-6.00003-0.* *Journal readers can use the code PBTY14 to purchase the book at a discounted price at the Elsevier Store* (**http://goo.gl/d61hl3**).

Introduction

This chapter discusses graph databases, which are used to model relationships rather than traditional structured data. Graph databases have nothing to do with presentation graphics. Just as FORTRAN is based on algebra, and relational databases are based on sets, graph databases are based on graph theory, a branch of discrete mathematics. Here is another way we have turned a mind tool into a class of programming tools!

Graph databases are not network databases. Those were the prerelational databases that linked records with pointer chains that could be traversed record by record. IBM's Information Management System (IMS) is one such tool still in wide use; it is a hierarchical access model. Integrated Database Management System (IDMS), TOTAL, and other products use more complex pointer structures (e.g., single-link lists, double-linked lists, junctions, etc.) to allow more general graphs than just a tree. These pointer structures are "hardwired" so that they can be navigated; they are the structure in which the data sits.

In a graph database, we are not trying to do arithmetic or statistics. We want to see relationships in the data. Curt Monash the database expert and blogger (**http://www.dbms2.com/, http://www.monash.com**) coined the term for this kind of analysis: *relationship analytics*.

Programmers have had algebra in high school, and they may have had some exposure to naive set theory in high school or college. You can program FORTRAN and other procedural languages with high school–level algebra and only a math major needs advanced set theory, which deals with infinite sets (computers do not have infinite storage no matter what your boss thinks). But you cannot really understand RDBMS and SQL without naive set theory.

But only math majors seem to take a whole semester of graph theory. This is really too bad; naive graph theory has simple concepts and lots of everyday examples that anyone can understand. Oh, did I mention that it is also full of sudden surprises where simple things become nonpolynomial (NP)-complete problems? Let's try to make up that gap in your education.

In computer science, we use the "big O" notation, $O(n)$, to express how much effort it takes to run an algorithm as the size of the input, (n). For example, if we have a simple process that handles one record at a time, the $O(n)$ is linear; add one more record and it takes one more unit of execution time. But some algorithms increase in more complex ways. For example, sorting a file can be done in $O(n \log_2(n))$ time. Other algorithms can have a complexity that is a polynomial usually with squares and cubes.

Then we get to the NP algorithms. They usually involve having to try all possible combinations to find a solution, so they have a factorial in their complexity, $O(n!)$.

NP complexity shows up in many of the classic graph theory problems. Each new edge or node added to the graph can result in more and more combinations. We often find that we look for near-optimal solutions instead of practical reasons.

3.1 Graph Theory Basics

A graph has two things in it. There are edges (or arcs) and nodes (or vertices); the edges are drawn as lines that connect nodes, which are drawn as dots or circles. That is it! Two parts! Do not be fooled; binary numbers only have two parts and we build computers with them.

3.1.1 Nodes

Nodes are abstractions. They usually (not always) model what would be an entity in RDBMS. In fact, some of the power of graph theory is that a node can model a subgraph. A node may or may not have "something inside it" (electrical components) in the model; it can just sit there (bus stop) or simply be (transition state). A node is not an object. Objects have methods and local data inside them. In a complex graph query, we might be looking for an unknown or even nonexistent node. For example, a bus stop with a Bulgarian barbeque stand might not exist. But a bus stop with a barbeque in a Bulgarian neighborhood might exist, and we not do know it until we connect many factors together (e.g., riders getting off at the Bulgarian Culture Center bus stop, restaurants or Bulgarian churches within *n* blocks of the bus stop, etc.). Other examples of graphs you might have seen are:

- *Schematic maps:* the nodes are the bus stops, towns, and so forth.
- *Circuit diagrams:* the nodes are electrical components.
- *State transitions:* the nodes are the states (yes, this can be modeled in SQL).

3.1.2 Edges

Edges or arcs connect nodes. We draw them as lines that may or may not have an arrow head on them to show a direction of flow. In schematic maps, the edges are the roads. They can have a distance or time on them. In the circuit diagrams, the edges are the wires that have resistance, voltage, etc. Likewise, the abstract state transitions are connected by edges that model the legal transition paths.

In one way, edges are more fun than nodes in graph theory. In RDBMS models of data, we have an unspecified single relationship between tables in the form of a REFERENCES clause. In a graph database, we can have multiple edges of different kinds between nodes. These can be of various strengths that we know (e.g., "I am your father, Luke," if you are a *Star Wars* fan; "is a pen pal of") and ones that we establish from other sources (e.g., "subscribes to the *Wall Street Journal*"; "friend of a friend of a friend"; "son of a son of a sailor," if you are a Jimmy Buffett fan).

At the highest level of abstraction an edge can be directed or undirected. In terms of maps, these are one-way streets; for state transitions, this is prior state–current state pairs and so forth. We tend to like undirected graphs since the math is easier and there is often an inverse relationship of some sort (e.g., "father–son" in the case of Luke Skywalker and Darth Vader).

Colored edges are literally colored lines on a display of a graph database. One color is used to show the same kind of edge, the classic "friend of a friend of a friend," or a Bacon(n) relationship (I will explain this shortly) used in social networks when they send you a "you might also know . . ." message and ask you to send an invitation to that person to make a direct connection.

Weighted edges have a measurement that can accumulate. In the case of a map—distances—the accumulation rule is additive; in the case of the Bacon(n) function it diminishes over distance (you may ask, "Who? Oh, I forgot about him!").

3.1.3 Graph Structures

The bad news is that since graph theory is fairly new by mathematical standards, which means less than 500 years old, there are still lots of open problems and different authors will use different terminology. Let me introduce some basic terms that people generally agree on:

- A *null graph* is a set of nodes without any edges. A *complete graph* has an edge between every pair of nodes. Both of these extremes are pretty rare in graph databases.
- A *walk* is a sequence of edges that connect a set of nodes without repeating an edge.
- A *connected graph* is a set of nodes in which any two nodes can be reached by a walk.
- A *path* is a walk that goes through each node only once. If you have n nodes, you will have (n–1) edges in the path.
- A *cycle* or *circuit* is a path that returns to where it started. In RDBMS, we do not like circular references because referential actions and data retrieval can hang in an endless loop. A *Hamiltonian circuit* is one that contains all nodes in a graph.
- A *tree* is a connected graph that has no cycles. I have a book on how to model trees and hierarchies in SQL (Celko, 2012). Thanks to hierarchical databases, we tend to think of a directed tree in which subordination starts at a root node and flows down to leaf nodes. But in graph databases, trees are not as obvious as an organizational chart, and finding them is a complicated problem. In particular, we can start at a node as the root and look for the minimal spanning tree. This is a subset of edges that give us the shortest path from the root we picked to each node in the graph.

Very often we are missing the edges we need to find an answer. For example, we might see that two people got traffic tickets in front of a particular restaurant, but this means nothing until we look at their credit card bills and see that these two people had a meal together.

3.2 RDBMS Versus Graph Database

As a generalization, graph databases worry about relationships, while RDBMSs worry about data. RDBMSs have difficulties with complex graph theoretical analysis. It's easy to manage a graph where every path has length one; that is just a three-column table (node, edge, node). By doing self-joins, you can construct paths of length two, and so forth, called a breadth-first search. If you need a mental picture, think of an expanding search radius. You quickly get into Cartesian explosions for longer paths, and can get lost in an endless loop if there are cycles. Furthermore, it is extremely hard to write SQL for graph analysis if the path lengths are long, variable, or not known in advance.

3.3 Six Degrees of Kevin Bacon Problem

The game "Six Degrees of Kevin Bacon" was invented in 1994 by three Albright College students: Craig Fass, Brian Turtle, and Mike Ginelli. They were watching television movies when the film *Footloose* was followed by *The Air Up There*, which lead to the speculation that everyone in the movie industry was connected in some way to Kevin Bacon. Kevin Bacon himself was assigned the Bacon number 0; anyone who had been in a film with him has a Bacon number of 1; anyone who worked with that second person has a Bacon number of 2; and so forth. The goal is to look for the shortest path. As of mid-2011, the highest finite Bacon number reported by the Oracle of Bacon is 8.

This became a fad that finally resulted in the website "Oracle of Bacon" (**http://oracleofbacon.org**), which allows you to do online searches between any two actors in the Internet Movie Database (**www.imdb.com**). For example, Jack Nicholson was in *A Few Good Men* with Kevin Bacon, and Michelle Pfeiffer was in *Wolf* with Jack Nicholson. I wrote a whitepaper for Cogito, Inc. of Draper, UT, in which I wrote SQL queries to the Kevin Bacon problem as a benchmark against their graph database. I want to talk about that in more detail.

3.3.1 Adjacency List Model for General Graphs

Following is a typical adjacency list model of a general graph with one kind of edge that is understood from context. Structure goes in one table and the nodes in a separate table, because they are separate kinds of things (i.e., entities and relationships). The SAG card number refers to the Screen Actors Guild membership identifier, but I am going to pretend that they are single letters in the following examples.

```
CREATE TABLE Actors
(sag_card CHAR(9) NOT NULL PRIMARY  KEY
 actor_name VARCHAR(30) NOT NULL);
```

```
CREATE TABLE MovieCasts
(begin_sag_card CHAR(9) NOT NULL
 REFERENCES Nodes (sag_card)
 ON UPDATE  CASCADE
 ON DELETE  CASCADE,
 end_sag_card CHAR(9) NOT NULL
 REFERENCES Nodes (sag_card)
 ON UPDATE  CASCADE
 ON DELETE CASCADE,
 PRIMARY  KEY (begin_sag_card, end_sag_card),
 CHECK (begin_sag_card <> end_sag_card));
```

I am looking for a path from Kevin Bacon, who is 's' for "start" in the example data, to some other actor who has a length less than six. Actually, what I would really like is the shortest path within the set of paths between actors.

The advantage of SQL is that it is a declarative, set-oriented language. When you specify a rule for a path, you get *all* the paths in the set. That is a good thing—usually. However, it also means that you have to compute and reject or accept all possible candidate paths. This means the number of combinations you have to look at increases so fast that the time required to process them is beyond the computing capacity in the universe. It would be nice if there were some heuristics to remove dead-end searches, but there are not.

I made one decision that will be important later; I added self-traversal edges (i.e., an actor is always in a movie with himself) with zero length. I am going to use letters instead of actor names. There are a mere five actors called {'s', 'u', 'v', 'x', 'y'}:

```
INSERT INTO Movies - 15  edges
VALUES ('s', 's'), ('s', 'u'), ('s', 'x'), ('u', 'u'), ('u', 'v'), ('u', 'x'), ('v', 'v'), ('v', 'y'), ('x',
'u'), ('x', 'v'), ('x', 'x'), ('x', 'y'), ('y', 's'), ('y', 'v'), ('y', 'y');
```

I am not happy about this approach, because I have to decide the maximum number of edges in the path before I start looking for an answer. But this will work, and I know that a path will have no more than the total number of nodes in the graph. Let's create a query of the paths:

```
CREATE TABLE Movies
(in_node CHAR(1) NOT NULL,
 out_node CHAR(1) NOT NULL)

INSERT INTO Movies
VALUES ('s', 's'), ('s', 'u'), ('s', 'x'), ('u', 'u'), ('u', 'v'), ('u', 'x'), ('v', 'v'), ('v', 'y'),
('x', 'u'), ('x', 'v'), ('x', 'x'), ('x', 'y'), ('y', 's'), ('y', 'v', ('y', 'y');

CREATE TABLE Paths
(step1 CHAR(2) NOT NULL,
 step2 CHAR(2) NOT NULL,
 step3 CHAR(2) NOT NULL,
 step4 CHAR(2) NOT NULL,
 step5 CHAR(2) NOT NULL,
 path_length INTEGER NOT NULL,
PRIMARY  KEY (step1, step2, step3, step4, step5));
```

Let's go to the query and load the table with all the possible paths of length five or less:

```
DELETE  FROM Paths;

INSERT INTO Paths
SELECT DISTINCT M1.out_node AS s1, -- it is 's' in this example
 M2.out_node AS s2,
 M3.out_node AS s3,
 M4.out_node AS s4,
 M5.out_node AS s5,
 (CASE WHEN  M1.out_node NOT IN (M2.out_node, M3.out_node, M4.out_
node, M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M2.out_node NOT IN (M3.out_node, M4.out_node,
M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M3.out_node NOT IN (M2.out_node, M4.out_node,
M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M4.out_node NOT IN (M2.out_node, M3.out_node,
M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M5.out_node NOT IN (M2.out_node, M3.out_node,
M4.out_node) THEN 1 ELSE 0 END) AS path_length
FROM Movies AS M1, Movies AS M2, Movies AS M3, Movies AS M4,
Movies AS M5
WHERE M1.in_node = M2.out_node
AND M2.in_node = M3.out_node
AND M3.in_node = M4.out_node
AND M4.in_node = M5.out_node
AND 0 < (CASE WHEN  M1.out_node NOT IN (M2.out_node, M3.out_node,
M4.out_node, M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M2.out_node NOT IN (M1.out_node, M3.out_node,
M4.out_node, M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M3.out_node NOT IN (M1.out_node, M2.out_node,
M4.out_node, M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M4.out_node NOT IN (M1.out_node, M2.out_node,
M3.out_node, M5.out_node) THEN 1 ELSE 0 END
  + CASE WHEN  M5.out_node NOT IN (M1.out_node, M2.out_node,
M3.out_node, M4.out_node) THEN 1 ELSE 0 END);

SELECT * FROM Paths ORDER BY step1, step5, path_length;
```

The Paths.step1 column is where the path begins. The other columns of Paths are the second step, third step, fourth step, and so forth. The last step column is the end of the journey. The SELECT DISTINCT is a safety thing and the "greater than zero" is to clean out the zero-length start-to-start paths. This is a complex query, even by my standards.

The path length calculation is a bit harder. This sum of CASE expressions looks at each node in the path. If it is unique within the row, it is assigned a value of 1; if it is not unique within the row, it is assigned a value of 0.

There are 306 rows in the path table. But how many of these rows are actually the same path? SQL has to have a fixed number of columns in a table, but paths can be of different lengths. That is to say that $(s, y, y, y, y)=(s, s, y, y, y)=(s, s, s, y, y)=(s, s, s, s, y)$. A path is not supposed to have cycles in it, so you need to filter the answers. The only places for this are in the WHERE clause or outside of SQL in a procedural language.

Frankly, I found it was easier to do the filtering in a procedural language instead of SQL. Load each row into a linked list structure and use recursive code to find cycles. If you do it in SQL, you need a predicate for all possible cycles of size 1, 2, and so forth, up to the number of nodes in the graph.

Bacon Number	SQL	Cogito
1	00:00:24	0.172ms
2	00:02:06	00:00:13
3	00:12:52	00:00:01
4	00:14:03	00:00:13
5	00:14:55	00:00:16
6	00:14:47	00:00:43

Table 3.1 Query Times for Bacon Numbers.

Internally, graph databases will also use a simple (node, edge, node) storage model, but they will additionally add pointers to link nearby nodes or subgraphs. I did a benchmark against a "Kevin Bacon" database. One test was to find the degrees with Kevin Bacon as "the center of the universe," and then a second test was to find a relationship between any two actors. I used 2,674,732 rows of data. Ignoring the time to set up the data, the

query times for the simple Bacon numbers are given in Table 3.1. The timings are raw clock times starting with an empty cache running on the same hardware. The SQL was Microsoft SQL Server, but similar results were later obtained with DB2 and Oracle.

The figures became much worse for SQL as I generalized the search (e.g., change the focus actor, use only actress links, use one common movie, and add directors). For example, changing the focus actor could be up to 9,000 times slower, most by several hours versus less than one minute.

3.3.2 Covering Paths Model for General Graphs

What if we attempt to store all the paths in a directed graph in a single table in an RDBMS? The table for this would look like the following:

```
CREATE TABLE Paths
(path_nbr INTEGER NOT NULL,
 step_nbr INTEGER NOT NULL
 CHECK (path_nbr >= 0),
 node_id CHAR(1) NOT NULL,
 PRIMARY  KEY (path_nbr, step_nbr));
```

Each path is assigned an ID number and the steps are numbered from 0 (the start of the path) to *k*, the final step. Using the simple six-node graph, the one-edge paths are:

```
1 0 A
1 1 B
2 0 B
2 1 F
3 0 C
3 1 D
4 0 B
4 1 D
5 0 D
5 1 E
```

Now we can add the two-edge paths:

```
6 0 A
6 1 B
6 2 F
7 0 A
7 1 B
7 2 D
8 0 A
8 1 C
8 2 D
9 0 B
9 1 D
9 2 E
```

And finally the three-edge paths:

```
10 0 A
10 1 B
10 2 D
10 3 E
11 0 A
11 1 B
11 2 D
11 3 E
```

These rows can be generated from the single-edge paths using a common table expression (CTE) or with a loop in a procedural language, such as SQL/ PSM. Obviously, there are fewer longer paths, but as the number of edges increases, so does the number of paths. By the time you get to a realistic-size graph, the number of rows is huge. However, it is easy to find a path between two nodes, as follows:

```
SELECT DISTINCT :in_start_node, :in_end_node, (P2.step_nbr- P1.step_
nbr) AS distance
FROM Paths AS P1, Paths AS P2
WHERE P1.path_nbr=P2.path_nbr
AND P1.step_nbr <= P2.step_nbr
AND P1.node_id = :in_start_node
AND P2.node_id = :in_end_node;
```

Notice the use of SELECT DISTINCT because most paths will be a subpath of one or more longer paths. Without it, the search for all paths from A to D in this simple graph would return:

```
7 0 A
7 1 B
7 2 D
8 0 A
8 1 C
8 2 D
10 0 A
10 1 B
10 2 D
11 0 A
11 1 B
11 2 D
```

However, there are only two distinct paths, namely (A, B, D) and (A, C, D). In a realistic graph with lots of connections, there is a good chance that a large percentage of the table will be returned.

Can we do anything to avoid the size problems? Yes and no. In this graph, most of the paths are redundant and can be removed. Look for a set of subpaths that cover all of the paths in the original graph. This is easy enough to do by hand for this simple graph:

```
1 0 A
1 1 B
1 2 F

2 0 A
2 1 B
2 2 D
2 3 E

3 0 A
3 1 C
3 2 D
3 3 E
```

The problem of finding the longest path in a general graph is known to be NP-complete, and finding the longest path is the first step of finding a minimal covering path set. For those of you without a computer science degree, NP-complete problems are those that require drastically more resources (storage space and/ or time) as the number of elements in the problem increases. There is usually an exhaustive search or combinatory explosion in these problems.

While search queries are easier in this model, dropping, changing, or adding a single edge can alter the entire structure, forcing us to rebuild the entire table. The combinatory explosion problem shows up again, so loading and validating the table takes too long for even a medium number of nodes. In another example, MyFamily.com (owner of Ancestry.com) wanted to let visitors find relationships between famous people and themselves. This involves looking for paths 10 to 20+ edges long, on a graph with over 200 million nodes and 1 billion

edges. Query rates are on the order of 20 per second, or 2 million per day.

3.3.3 Real-World Data Has Mixed Relationships

Now consider another kind of data. You are a cop on a crime scene investigator show. All you have is a collection of odd facts that do not fall into nice, neat relational tables. These facts tie various data elements together in various ways. You now have 60 minutes to find a network of associations to connect the bad guys to the crime in some as-of-yet-unknown manner.

Ideally, you would do a join between a table of "bad guys" and a table of "suspicious activities" on a known relationship. You have to know that such a join is possible before you can write the code. You have to insert the data into those tables as you collect it. You cannot whip up another relationship on-the-fly.

Let's consider an actual example. The police collect surveillance data in the form of notes and police reports. There is no fixed structure in which to fit this data. For example, U-Haul reports that a truck has not been returned and they file a police report. That same week, a farm-supply company reports someone purchased a large amount of ammonium nitrate fertilizer. If the same person did both actions, and used his own name (or with a known alias) in both cases, then you could join them into a relationship based on the "bad guys" table. This would be fairly easy; you would have this kind of query in a view for simple weekly reports. This is basically a shortest-path problem and it means that you are trying to find the dumbest terrorist in the United States.

In the real world, conspirator A rents the truck and conspirator B buys the fertilizer. Or one guy rents a truck and cannot return it on time while another totally unrelated person buys fertilizer paying in cash rather than using an account that is known to belong to a farmer. Who knows? To find if you have a coincidence or a conspiracy, you need a relationship between the people involved. That relationship can be weak (both people live in New York state) or strong (they were cellmates in prison).

Figure 3.1 is a screenshot of this query and the subgraph that answers it. Look at the graph that was generated from the sample data when it was actually given a missing rental truck and a fertilizer purchase. The result is a network that joins the truck to the fertilizer via two ex-cons, who shared jail time and a visit to a dam. Hey, that is a red flag for anyone! This kind of graph network is called a *causal diagram* in statistics and fuzzy logic. You will also see the same approach as a fishbone diagram (also known as cause-and-effect diagram and Ishikawa diagram after their inventor) when you are looking for patterns in data. Before now, this method has been a "scratch-paper" technique. This is fine when you are working on one very dramatic case in a 60-minute police show and have a scriptwriter.

In the real world, a major police department has a few hundred cases a week. The super-genius Sherlock Holmes characters are few and far between. But even if you could find such geniuses, you simply do not have enough whiteboards to do this kind of analysis one case at a time in the real world. Intelligence must be computerized in the 21st century if it is going to work.

Most crime is committed by repeat offenders. Repeat offenders tend to follow patterns—some of which are pretty horrible, if you look at serial killers. What a police department wants to do is describe a case, then look through all the open cases to see if there are three or more cases that have the same pattern.

One major advantage is that data goes directly into the graph, while SQL requires that each new fact has to be checked against the existing data. Then the SQL data has to be encoded on some scale to fit into a column with a particular data type.

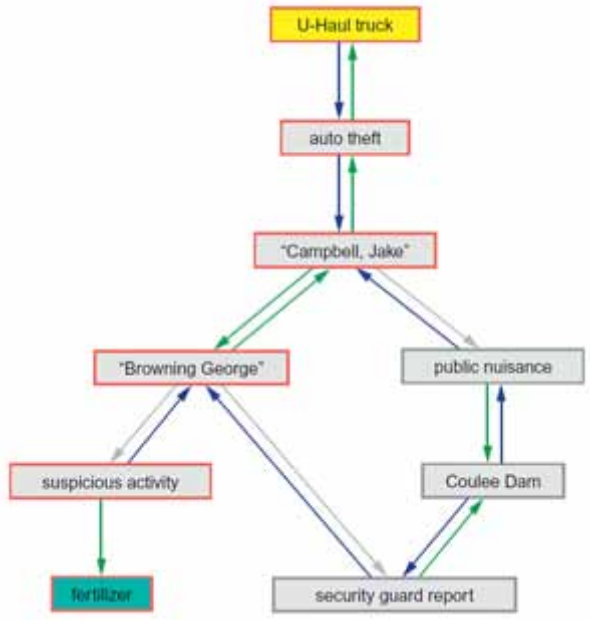


Figure 3.1 Possible Terrorist Attack Graph.

3.4 Vertex Covering

Social network marketing depends on finding "the cool kids," the trendsetters who know everybody in a community. In some cases, it might be one person. The Pope is fairly well known among Catholics, for example, and his opinions carry weight.

Formally, a vertex cover of an undirecterd graph G is a set C of vertices such that each edge of G is incident to at least one vertex in C. The set C is said to cover the edges of G. Informally, you have a weird street map and want to put up security cameras at intersections (nodes) in such a way that no street (edge) is not under surveillance. We also talk about coloring the nodes to mark them as members of the set of desired vertices. Figure 3.2 shows two vertex coverings taken from a Wikipedia article on this topic.

However, neither of these coverings is minimal. The three-node solution can be reduced to two nodes, and the four-node solution can be reduced to three nodes, as follows:

```
CREATE TABLE Grid
(v1 SMALLINT  NOT NULL CHECK (v1>0),
v2  SMALLINT  NOT NULL CHECK (v2>0),
PRIMARY  KEY (v1, v2),
CHECK (v1<v2),
color SMALLINT DEFAULT 0 NOT NULL CHECK (color>= 0));

INSERT INTO Grid (v1, v2)
VALUES (1, 2), (1, 4), (2, 3), (2, 5), (2, 6);
```

{1, 2, 6} and {2, 4} are vertex covers. The second is the minimal cover. Can you prove it? In this example, you can use brute force and try all possible coverings. Finding a vertex covering is known to be an NP-complete problem, so brute force is the only sure way. In practice, this is not a good solution because the

combinatorial explosion catches up with you sooner than you think.

One approach is to estimate the size of the cover, then pick random sets of that size. You then keep the best candidates, look for common subgraphs, and modify the candidates by adding or removing nodes. Obviously, when you have covered 100% of the edges, you have a winner; it might not be optimal, but it works.

Figure 3.2 Vertex coverings from Wikipedia: (a) three-node solution, and (b) a four-node solution.

Another consideration is that the problem might start with a known number of nodes. For example, you want to give n sample items to bloggers to publicize your product. You want to gift the bloggers with the largest readership for the gifts, with the least overlap.

3.5 Graph Programming Tools

As of 2012, there are no ANSI or ISO standard graph query languages. We have to depend on proprietary languages or open-source projects. They depend on underlying graph databases, which are also proprietary or open-source projects. Support for some of the open-source projects is available from commercial providers; Neo4j is the most popular product and it went commercial in 2009 after a decade in the open-source world. This has happened in the relational world with PostgreSQL, MySQL, and other products, so we should not be surprised.

There is an ISO standard known as resource description framework (RDF), which is a standard model for data interchange on the Web. It is based on the RDF that extends the linking structure of the Web to use URIs (uniform resource identifiers) to name the relationship between things as well as the two ends of the link (a *triple*). URIs can be classified as locators (URLs), names (URNs), or both. A uniform resource name (URN) functions like a person's name, while a uniform resource locator (URL) resembles that person's street address. In other words, the URN defines an item's identity, while the URL provides a method for finding it.

The differences are easier to explain with an example. The ISBN uniquely identifies a specific edition of a book, its identity. But to read the book, you need its location: a URL address. A typical URL would be a file path for the electronic book saved on a local hard disk.

Since the Web is a huge graph database, many graph databases build on RDF standards. This also makes it easier to have a distributed graph database that can use existing tools.

3.5.1 Graph Databases

Some graph databases were built on an existing data storage system to get started, but then were moved to custom storage engines. The reason for this is simple: performance. Assume you want to model a simple one-to-many relationship, such as the Kevin Bacon problem. In RDBMS and SQL, there will be a table for the relationship, which will contain a reference to the table with the unique entity and a reference for each row matching to that entity in the many side of the relationship. As the relational tables grow, the time to perform the join increases because you are working with entire sets.

In a graph model, you start at the Kevin Bacon node and traverse the graph looking at the edges with whatever property you want to use (e.g., "was in a movie with"). If there is a node property, you then filter on it (e.g., "this actor is French"). The edges act like very small, local relationship tables, but they give you a traversal and not a join.

A graph database can have ACID transactions. The simplest possible graph is a single node. This would be a record with named values, called properties. In theory, there is no upper limit on the number of properties in a node, but for practical purposes, you will want to distribute the data into multiple nodes, organized with explicit relationships.

3.5.2 Graph Languages

There is no equivalent to SQL in the graph database world. Graph theory was an established branch of mathematics, so the basic terminology and algorithms were well-known were the first products came along. That was an advantage. But graph database had no equivalent to IBM's System R, the research project that defined SEQUEL, which became SQL. Nor has anyone tried to make one language into the ANSI, ISO or other industry standard.

SPARQL

SPARQL (pronounced "sparkle," a recursive acronym for SPARQL Protocol and RDF Query Language) is a query language for the RDF format. It tries to look a little like SQL by using common keywords and a bit like C with special ASCII characters and lambda calculus. For example:

```
PREFIX abc: <http://example.com/exampleOntology#>

SELECT ?capital ?country
WHERE {
?x  abc:cityname ?capital;
abc:isCapitalOf ?y.
?y  abc:countryname ?country;
abc:isInContinent abc:Africa.}
```

where the ? prefix is a free variable, and : names a source.

SPASQL

SPASQL (pronounced "spackle") is an extension of the SQL standard, allowing execution of SPARQL queries within SQL statements, typically by treating them as subqueries or function clauses. This also allows SPARQL queries to be issued through "traditional" data access APIs (ODBC, JDBC, OLE DB, ADO.NET, etc.).

Gremlin

Gremlin is an open-source language that is based on traversals of a property graph with a syntax taken from OO and the C programming language family (**https://github.com/tinkerpop/gremlin/wiki**). There is syntax for directed edges and more complex queries that looks more mathematical than SQL-like. Following is a sample program. Vertexes are numbered and a traversal starts at one of them. The path is then constructed by in–out paths on the 'likes' property:

```
g = new Neo4jGraph('/tmp/neo4j')

// calculate basic collaborative filtering for vertex 1
m  = [:]
g.v(1).out('likes').in('likes').out('likes').groupCount(m) m.sort{-it.value}
```

```
// calculate the primary eigenvector (eigenvector centrality) of a graph
m  = [:]; c = 0;
g.V.out.groupCount(m).loop(2){c++<1000}
m.sort{-it.value}
```

Eigenvector centrality is a measure of the influence of a node in a network. It assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. It measures the effect of the "cool kids" in your friends list. Google's PageRank is a variant of the eigenvector centrality measure.

Cypher (NEO4j)

Cypher is a declarative graph query language that is still growing and maturing, which will make SQL programmers comfortable. It is not a weird mix of odd ASCII charterers, but human-readable keywords in the major clauses. Most of the keywords like WHERE and ORDER BY are inspired by SQL. Pattern matching borrows expression approaches from SPARQL. The query language is comprised of several distinct clauses:

- START: starting points in the graph, obtained via index lookups or by element IDs.
- MATCH: the graph pattern to match, bound to the starting points in START.
- WHERE: filtering criteria.
- RETURN: what to return.
- CREATE: creates nodes and relationships.
- DELETE: removes nodes, relationships, and properties.
- SET: sets values to properties.
- FOREACH: performs updating actions once per element in a list.
- WITH: divides a query into multiple, distinct parts.

For example, following is a query that finds a user called John in an index and then traverses the graph looking for friends of John's friends (though not his direct friends) before returning both John and any friends-of-friends who are found:

```
START john=node:node_auto_index(name = 'John')
MATCH john-[:friend]->()-[:friend]->fof
RETURN john, fof
```

We start the traversal at the john node. The MATCH clause uses arrows to show the edges that build the friend-of-friend edges into a path. The final clause tells the query what to return.

In the next example, we take a list of users (by node ID) and traverse the graph looking for those other users who have an outgoing friend relationship, returning only those followed users who have a name property starting with S:

```
START user=node(5,4,1,2,3)
MATCH user-[:friend]->follower
WHERE follower.name =~ 'S.*'
RETURN user, follower.name
```

The WHERE clause is familiar from SQL and other programming languages. It has the usual logical operators of AND, OR, and NOT; comparison operators; simple math; regular expressions; and so forth.

Trends

Go to **http://www.graph-database.org/** for PowerPoint shows on various graph language projects. It will be in flux for the next several years, but you will see several trends. The proposed languages are declarative, and are borrowing ideas from SQL and the RDBMS model. For example, GQL (Graph Query Language) has syntax for SUBGRAPH as a graph venison of a derived table. Much like SQL, the graph languages have to send data to external users, but they lack a standard way of handing off the information.

It is probably worth the effort to get an open-source download of a declarative graph query language and get ready to update your resume.

Concluding Thoughts

Graph databases require a change in the mindset from computational data to relationships. If you are going to work with one of these products, then you ought to get math books on graph theory. A short list of good introductory books are listed in the Reference section. ▲

References

- Celko, J. (2012). *Trees and hierarchies in SQL for smarties*. Burlington, MA: Morgan-Kaufmann. ISBN: 978-0123877338.
- Chartrand, G. (1984). *Introductory graph theory.* Mineola, NY: Dover Publications. ISBN: 978-0486247755.
- Chartrand, G., & Zhang, P. (2004). *A first course in graph theory.* New York: McGraw-Hill. ISBN: 978-0072948622.
- Gould, R. (2004). *Graph theory.* Mineola, NY: Dover Publications. ISBN: 978-0486498065.
- Trudeau, R. J. (1994). *Introduction to graph theory.* Mineola, NY: Dover Publications. ISBN: 978-0486678702.
- Maier, D. (1983). *Theory of relational databases.* Rockville. MD: Computer Science Press. ISBN: 978-0914894421.
- Wald, A. (1973). *Sequential analysis*. Mineola, NY: Dover Publications. ISBN: 978-0486615790.

Joe Celko served 10 years on the ANSI/ISO SQL Standards Committee and contributed to the SQL-89 and SQL-92 Standards. Mr. Celko is author of a series of books on SQL and RDBMS for Elsevier/Morgan Kaufmann. He is an independent consultant based in Austin, Texas. He has written over 1200 columns in the computer trade and academic press, mostly dealing with data and databases.

Oracle Spatial *and* Graph

by Bill Beauregard

Bill Beauregard

For the last ten years, Oracle Spatial and Graph has delivered the industry's most advanced enterprise spatial and graph data management solution. Oracle Spatial and Graph is completely integrated with Oracle Database 12*c*, thereby providing developers with the enterprise-class performance, scalability, reliability, and security necessary for today's graph-based applications. It provides support for two graph data models, along with the industry's leading spatial data management.

RDF Semantic Graph supports the World Wide Web Consortium (W3C) Resource Description Framework (RDF) graph standard. This model supports the unique data management, querying, and inferencing commonly used in a variety of social network analysis and linked open data solutions.

Network Data Model (NDM) Graph is a property graph data model used to model and analyze physical and logical networks used in industries such as transportation, logistics, and utilities. NDM persists the network connectivity metadata in the database, while a Java API provides fast in-memory graph analytics, including shortest path, nearest neighbors, within cost, and reachability.

Spatial data management supports the complex requirements found in Geographic Information Systems (GISs), enterprise applications, and location-enabled business and web mapping applications. Capabilities include native support for spatial data models and types such as georaster (for geo-referenced imagery and gridded data); topology; 3D, including triangulated irregular networks (TINs) and point clouds (supporting LIDAR data); and linear referencing. In addition, a geocoding engine, a routing engine, and spatial web services conformant with Open Geospatial Consortium (OGC) and ISO standards are also included. These geospatial features provide a complete platform to deploy a range of enterprise GIS and web mapping solutions used in the public sector, utilities, telecommunications, retail, and logistics.

The following sections provide detailed overview of Oracle's two native graph capabilities: RDF Semantic Graph and the Network Data Model.

RDF Semantic Graph Features Overview

Graphs are becoming central to a new category of social network and linked data applications common in public sector, health sciences, finance, media and intelligence communities. The graph data structure is an ideal *metadata layer* to support the integration of various data sources.

EmpNo	Ename	Job	Mgr	DeptNo
7521	Ward	Salesman	7698	10
7698	Blake	Manager	7839	10
7839	King	President		30

DeptNo	LOC
10	NYC
30	CHI

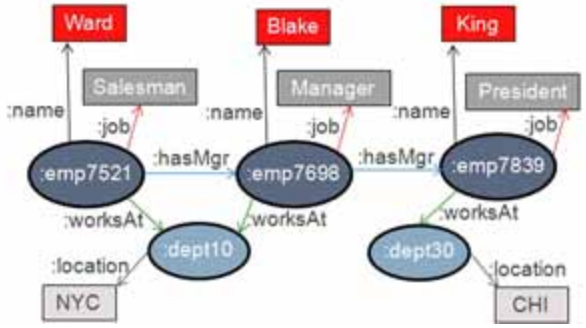


Figure 1: Representing relational table as an RDF graph.

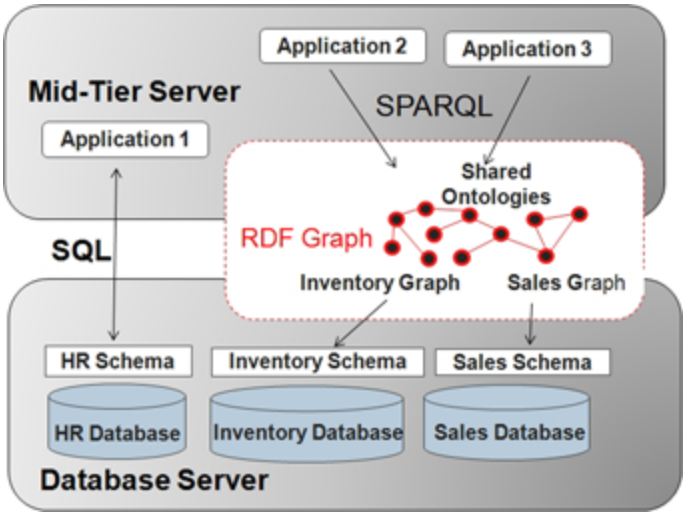


Figure 2: RDF creates a flexible common enterprise vocabulary.

The RDF graph model can represent relational database schema as a graph. It can align the entities and semantics in the graph (schema) with the semantics of an enterprise vocabulary or ontology. This ensures that application developers code to a common, semantically consistent vocabulary when performing federated queries. The RDF model can also federate diverse data types, such as relational, text, spatial, images, and spreadsheets.

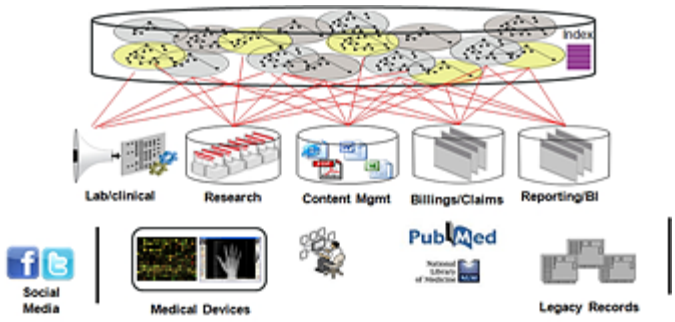


Figure 3: Modeling diverse federated data types with RDF.

Storing, Loading, and Data Manipulation

RDF is the W3C standard for semantic data. An RDF data element is a "resource." A unique benefit of RDF graphs is that resources enable data integration between different and even disparate data sets. Integration is possible because each resource has a globally unique universal resource identifier (URI), like a Social Security number. Resources form simple "subject-predicate-object" statements. The predicate is a property of the subject and the type of relationship between the subject and the object. The object is either a value for the property (a literal) or the URI of another subject that connects triples to form a graph. Ultimately, RDF graphs comprise nodes (subjects and objects) connected by directed, labeled edges (predicates) as illustrated by the labeled arrow in the figure below.

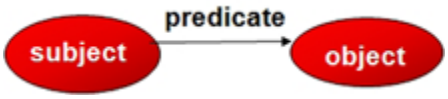


Figure 4: An RDF triple has a directed, labeled edge.

RDF graphs are flexible. It is easy to expand and combine RDF graphs because business information about the data (metadata) is stored in the graph as relationships rather than as column headers in a table.

Relational data can easily be converted or mapped to an RDF graph.

Person	FriendOf	Age
John	Mary	32
Mary	John	null

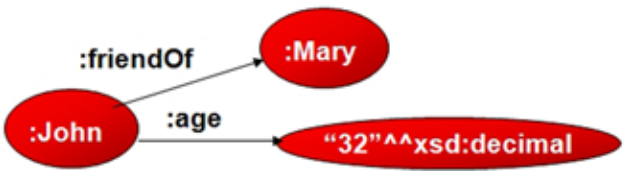


Figure 5: Mapping relational data to RDF triples.

The example above illustrates another benefit of RDF graphs; sparse data is stored flexibly and efficiently. Only existing relationships for a given subject are stored. Add new relationships by simply inserting a new triple for a particular subject.

The unit of storage in RDF Semantic Graph is a model. Models are user-defined and application-specific. In addition, *virtual model* capability provides a view-like feature to combine models for querying. RDF Semantic Graph leverages the standard Oracle Database 12*c* loading, storing, and data manipulation operations on RDF/OWL models. Space-efficient storage saves up to 60% disk space for scalable and performant loading, querying, and inferencing. It has proven scalability beyond tens of billions of triples (LUBM 200K benchmark) and can readily scale into the tens of petabytes of triples.

Querying RDF Graphs in Oracle Database 12*c*

RDF has its own graph query language, SPARQL. It is a simpler way to write query patterns that are joined together. For example, the following SPARQL query finds pairs of siblings, people with the same parents.

```
SELECT ?x ?y
 FROM <rdf_graph>
WHERE
{
  ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  FILTER( ?x != ?y)
}
```

The same query in SQL is more complex:

```
SELECT g1.subject x, g3.subject y
FROM rdf_graph g1, rdf_graph g2, rdf_graph g3, rdf_graph g4
WHERE g1.predicate = 'hasFather'
    AND g2.predicate = 'hasMother'
    AND g3.predicate = 'hasFather'
    AND g4.predicate = 'hasMother'
    AND g1.subject = g2.subject
    AND g3.subject = g4.subject
    AND g1.object = g3.object
    AND g2.object = g4.object
    AND g1.subject != g3.subject
```

RDF Semantic Graph supports SPARQL 1.1, the latest W3C standard and a range of deployment approaches, including a web service endpoint and Java APIs. Unique to Oracle, SQL can query RDF graphs by embedding SPARQL queries in SQL using the SEM_MATCH table function.

Here is an example of a SPARQL query:

```
PREFIX foaf: <http://...>
SELECT ?n1 ?n2
FROM <http://g1>
WHERE
{
  ?p foaf:name ?n1
  OPTIONAL
  {
    ?p foaf:knows ?f .
    ?f foaf:name ?n2
  }
  FILTER (REGEX(?n1, "^A"))
}
```

Here is an example of a SQL query with embedded SPARQL:

```
SELECT n1 n2
FROM TABLE(SEM_MATCH (
  '{
    ?p foaf:name ?n1
    OPTIONAL
    {
      ?p foaf:knows ?f .
      ?f  foaf:name ?n2
    }
    FILTER (REGEX (?n1, "^A"))
  } ',
  SEM_MODELS('g1') ,…,
  SEM_ALIASES (SEM_ALIAS ('foaf','http://…')), …
))
```

The SEM_MATCH table function in Oracle Spatial and Graph supports SPARQL 1.1. Additionally, RDF Semantic Graph supports the OGC GeoSPARQL standard that allows spatial querying and filtering on location data stored in the graph.

Viewing Relational Data as RDF Triples

It is possible to apply RDF views on relational tables using Oracle Spatial and Graph. RDF views present relational data in RDF triple format. SPARQL queries on these views enable powerful data integration across federated data sources, a key requirement for Linked Data and enterprise integration applications. Development is simplified; graph pattern-matching relationship queries on relational tables do not require converting the tables to RDF triples.

RDF Semantic Graph supports three types of view definitions: a) *automatic mapping* (also referred to as Direct Mapping); b) *custom mapping*, using the W3C R2RML language; and c) *materialized views*. The simplest way to create a mapping of relational data to RDF data is by calling the SEM_APIS.CREATE_RDFVIEW_MODEL procedure to create an RDF view. It takes as input the list of tables or views you would like to view as RDF. This provides a direct mapping of those relational tables or views.

The following example illustrates Direct Mapping:

```
-- Use the following relational tables.

CREATE TABLE dept (
  deptno NUMBER CONSTRAINT pk_DeptTab_deptno PRIMARY KEY,
  dname VARCHAR2(30),
  loc VARCHAR2(30)
);

CREATE TABLE emp (
  empno NUMBER PRIMARY KEY,
  ename VARCHAR2(30),
  job VARCHAR2(20),
  deptno NUMBER REFERENCES dept (deptno)
);

-- Insert some data.

INSERT INTO dept (deptno, dname, loc)
  VALUES (1, 'Sales', 'Boston');
INSERT INTO dept (deptno, dname, loc)
  VALUES (2, 'Manufacturing', 'Chicago');
INSERT INTO dept (deptno, dname, loc)
  VALUES (3, 'Marketing', 'Boston');

INSERT INTO emp (empno, ename, job, deptno)
  VALUES (1, 'Alvarez', 'SalesRep', 1);
INSERT INTO emp (empno, ename, job, deptno)
  VALUES (2, 'Baxter', 'Supervisor', 2);
INSERT INTO emp (empno, ename, job, deptno)
  VALUES (3, 'Chen', 'Writer', 3);
INSERT INTO emp (empno, ename, job, deptno)
  VALUES (4, 'Davis', 'Technician', 2);

-- Create an RDF view model using direct mapping of two tables, EMP and
DEPT,
-- with a base prefix of http://empdb/.
-- Specify KEY_BASED_REF_PROPERTY=T for the options parameter.

BEGIN
  sem_apis.create_rdfview_model(
    model_name => 'empdb_model',
    tables => SYS.ODCIVarchar2List('EMP', 'DEPT'),
    prefix => 'http://empdb/',
    options => 'KEY_BASED_REF_PROPERTY=T'
  );
END;
/

-- Query an RDF view using SPARQL in a SEM_MATCH-based SQL query.
-- The next statament is a query against an RDF view named empdb_model
-- to find the employees who work for any department located in Boston.

SELECT emp
  FROM TABLE(SEM_MATCH(
    '{?emp emp:ref-DEPTNO ?dept . ?dept dept:LOC "Boston"}',
    SEM_Models('empdb_model'),
    NULL,
    SEM_ALIASES(
      SEM_ALIAS('dept','http://empdb/TESTUSER.DEPT#'),
      SEM_ALIAS('emp','http://empdb/TESTUSER.EMP#')
    ),
    null));
```

This produces the following output:

```
EMP
--------------------------------------------------------------------------------
http://empdb/TESTUSER.EMP/EMPNO=1
http://empdb/TESTUSER.EMP/EMPNO=3
```

The preceding query is functionally comparable to this:

```
SELECT e.empno FROM emp e, dept d
WHERE e.deptno = d.deptno AND d.loc = 'Boston';
```

Alternatively, an R2RML mapping document could customize the mapping between the relational tables and the RDF graph in the example above.

Native Inferencing in Oracle Database 12*c*

Inferencing allows machine-driven discovery of implicit facts from explicit RDF data. It gives application developers the advantage of deriving more application knowledge with no coding by using standards-defined rules and semantics. Inferencing adds new facts to the graph.

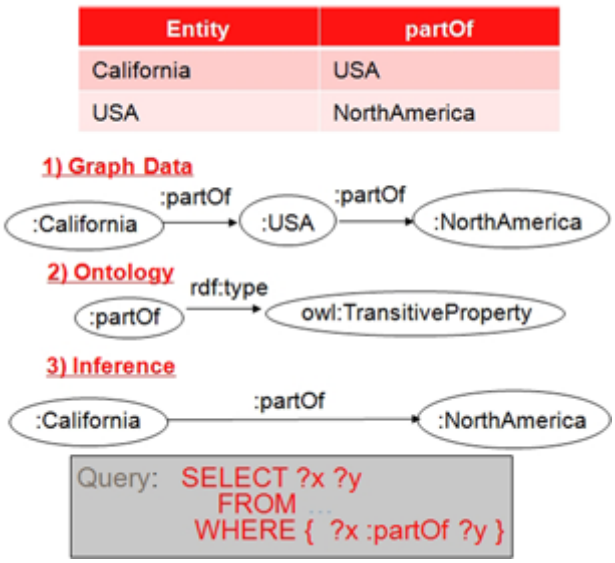


Figure 6: Inferencing adds the fact that California is part of North America.

RDF Semantic Graph provides native, forward-chaining, persistent inferencing using any combination of the built-in RDF, RDFS, and OWL 2 RL and EL profiles, as well as user-defined rules for specialized inference capabilities. Because inferencing is persistent and performed in advance of queries, it enhances query performance. Recent performance enhancements include optimized large owl:sameAs sets, incremental inference to update entailments after triple inserts, and parallel inference on multi-core or multi-CPU architectures.

For high-security applications, Oracle Spatial and Graph supports ladder-based inferencing. This feature applies appropriate security labels to newly inferred triples. RDF Semantic Graph supports additional reasoning requirements with a plug-in framework to integrate third-party special-purpose reasoners.

XML, JSON and Relational Interoperability

Oracle uniquely allows a SQL query to consult an ontology to provide more complete SQL results. Ontology-assisted SQL querying allows SQL queries to extract more semantically complete results from relational tables. By aligning relational data with a pre-defined ontology (or enterprise vocabulary) developers can better organize their instance data with a rich domain context.

For example, the following SQL query to find patients with upper extremity fractures on the table below returns no results:

```
SELECT p_id, diagnosis
FROM Patients
WHERE diagnosis ='Upper_Extremity_Fracture';
```

P_ID	DIAGNOSIS
1	Hand_Fracture
2	Rheumatoid_Arthritis
3	Finger_Fracture

Figure 7: SQL query on patient diagnosis table.

However, the National Cancer Institute (NCI) medical ontology indicates that a hand fracture is a type of upper extremity fracture.

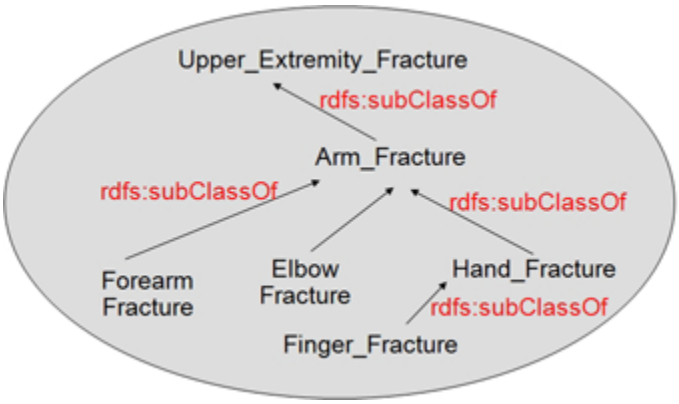


Figure 8: Portion of the NCI medical ontology.

When the Oracle SEM_RELATED operator is added to the SQL query, it consults the NCI ontology and finds patient 1 has an upper extremity fracture.

```
-- Ontology assisted SQL query
SELECT p_id, diagnosis
 FROM Patients
 WHERE SEM_RELATED (
              diagnosis,
              'rdfs:subClassOf',
              'Upper_Extremity_Fracture';
              'Medical_ontology' = 1)
AND SEM_DISTANCE() <= 2;
```

To support integration with external business intelligence (BI) and mid-tier tools, Oracle Spatial and Graph also provides a SPARQL Gateway feature that presents SPARQL query results in XML format. This is essential for supporting enterprise software tools that ingest XML data sources, such as Oracle Business Intelligence (OBIEE). RDF Semantic Graph can return RDF query results in JSON interoperability format as well.

Fine-Grained Security

As mentioned earlier, certain applications require fine-grained security. Model-level access control is the default for RDF graph data. Oracle Label Security provides triple-level security for the most stringent security levels. It defines sensitivity labels on individual triples and users that conditionally restrict a user's access to individual triples stored in an RDF model. Oracle Spatial and Graph is unique in providing such a fine-grained security model.

Graph Analytics

To support social network analysis (SNA), Oracle Spatial and Graph supports SPARQL 1.1 property path expressions that find relationships within a large social graph. Oracle Advanced Analytics data mining and R statistical analysis can operate on graph query results. The Network Data Model provides in-memory graph analytics for shortest path, reachability, within cost, and nearest neighbor analysis on RDF data.

SPARQL Property Path Expressions

Support for SPARQL property path expressions is included. An expression determines whether there is any connectivity in the graph between two nodes in the graph, such as between Tim and Sam in the example below.

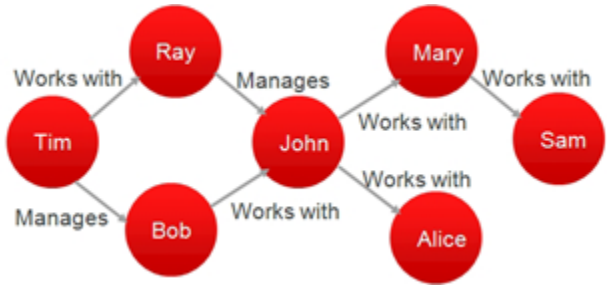


Figure 9: Modeling social connectivity with RDF.

Oracle Advanced Analytics provides data mining and R statistical analysis capabilities for Oracle Database 12*c*. These features can also analyze the results of SPARQL pattern-matching queries on graphs in Oracle Database 12*c*.

Problem Classification	Sample Problem
Classification	Given demographic data about a set of customers, predict customer response to an affinity card program
Regression	Given demographic and purchasing data about a set of customers, predict customers' age
Attribute Importance	Given customer response to an affinity card program, find the most significant predictors

Figure 10: Oracle Data Mining.

Oracle R Enterprise

- Open source language and environment

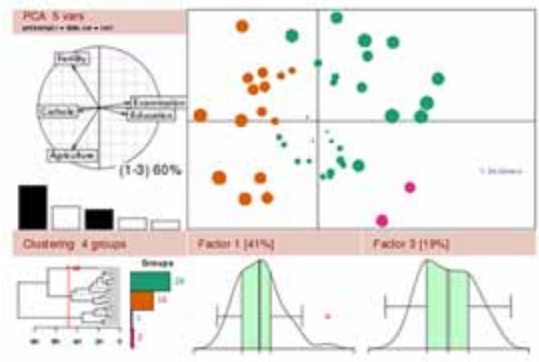


Figure 11: Oracle R Enterprise.

- Statistical computing and graphics
- Easily produces publication-quality plots
- Highly extensible with open-source R packages

Semantic Indexing for Documents

Semantic indexing for documents goes beyond key word searching to find documents that have semantically related content. SPARQL graph pattern queries interrogate the semantic index to find relevant documents. For instance, referring to our earlier SPARQL example, one could find all documents that mention pairs of siblings. The results could include the names of the siblings and the document(s) that mention them, as well as other attributes.

Indexing occurs automatically by inserting documents, text, and URLs into a table that has a semantic index defined on it. An associated third-party natural language processing (NLP) engine or annotator analyzes the text and generates RDF triples or XML from the concepts and terms found in the text for the semantic index. Inferencing on the semantic index can discover new relationships and expand search results.

Enterprise Performance and Scalability

RDF Semantic Graph supports parallelism, compression, partitioning, Oracle Real Applications Clusters, and the Oracle Exadata Database Machine for enterprise-level performance, scalability, and reliability.

Partitioning Support for Spatial Indexes

Models subdivide a graph in Oracle Database 12*c* and reside in separate partitions. Partitioning offers significant performance, scalability, and manageability benefits, including the following:

- Reduce response times for long-running queries; partitioning can reduce disk I/O operations.
- Reduce response times for concurrent queries; I/O operations run concurrently on each partition.
- Maintain indexes more easily with partition-level operations to create and rebuild an index.
- Rebuild indexes on partitions without affecting the queries on other partitions.
- Change storage parameters for each local index independent of other partitions.
- Split, merge, and exchange partitions.

Parallel Support

Support for parallelism is an important differentiating feature of Oracle Spatial and Graph. Parallelism enhances the performance of graph data loading, queries, and inferencing operations. Parallelism also speeds index creation; it subdivides graph B-tree index creation into multiple tasks performed in parallel.

Unique Enterprise Manageability

RDF Semantic Graph supports Oracle Database 12*c* utilities and tools, including:

- **Bulk loading**, including the Jena Adapter bulk loader, Oracle external tables, and SQL*Loader Direct Path loading
- **Replication and recovery**, including Data Guard: physical standby, Data Pump staging tables, and Recovery Manager
- **Tuning** with parallelism, Btree indexing of triples and quads, typed literals indexing, SPARQL query hints, statistics gathering, and Dynamic Sampling

Figure 12: Oracle Enterprise Manager supports RDF Semantic Graph.

- **Query management and execution controls** in Jena Adapter, including timeout, query abort framework, query hints, and parallel execution
- **Performance Analysis** with Enterprise Manager

Open Standards

Oracle is a World Wide Web Consortium (W3C) member and active contributor and/or editor in various technical working groups.

RDF Semantic Graph supports the latest W3C specifications, including RDF, RDF Schema (RDFS), SPARQL 1.1 query language, OWL 2 (RL and EL profiles) knowledge representation languages for authoring ontologies, the Simple Knowledge Organization System (SKOS), and the RDB2RDF specifications for creating RDF views on relational tables—Direct Mapping (DM) and Mapping Language (R2RML). In addition, Oracle was instrumental in defining and supporting the Open Geospatial Consortium's (OGC's) GeoSPARQL 1.0 query specification.

Rich Ecosystem of Tools

Leading third-party commercial graph tools and applications

support RDF Semantic Graph. Products include ontology modeling and engineering, vocabulary alignment, and graph visualization tools. In addition, Oracle Business Intelligence and Oracle Advanced Analytics products can analyze RDF data. Finally, Oracle's RDF graph database supports the leading open-source application development frameworks, including Apache Jena, Sesame, Protégé, and associated open-source tools as well as linked open data ontologies.

Figure 13: Business Intelligence (OBIEE) integration with RDF.

Big Data and NoSQL Interoperability

As graph data management becomes increasingly important to big data and NoSQL platforms, Oracle now supports RDF graphs in Oracle NoSQL Database. By incorporating W3C standards–based RDF support and open-source Apache Jena support (a Jena Adapter) into Oracle NoSQL Database, users benefit from high performance, low latency, and a no schema design ideal for low-latency retrieval and high-volume capture on simple data. This key-value store is ideally suited for exploiting the cost/performance benefits of horizontally scaled cloud-based and Linked Open Data architectures.

Network Data Model (NDM) Graph Features Overview

The NDM graph is a feature of Oracle Spatial and Graph that supports the network modeling and analysis requirements of leading telecommunications, electrical utilities, transportation, and logistics companies worldwide.

NDM stores general network (or property graph) data structures persistently in Oracle Database 12*c*. It explicitly stores and maintains network connectivity and provides network analysis capability, including shortest path, nearest neighbors, within cost, and reachability.

NDM has a PL/SQL API for managing network data in the database and a Java API for performing network analysis in mid-tier memory. The Java API is also instrumental in creating and applying network constraints.

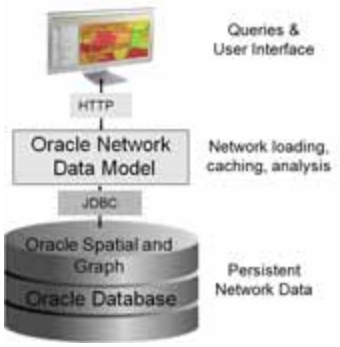


Figure 14: NDM Architecture.

To support analysis of very large graph models that exceed the platform's memory limits, the NDM supports partitioning of large networks into manageable sub-networks that automatically load graph partitions into memory as needed for efficient in-memory analysis. This enables application developers to benefit from the speed of in-memory analytics on networks that are larger than available memory.

Geocoding and Route Analysis

NDM is central to support a range of web mapping, point-to-point routing, and geocoding services common in most geospatial applications. It uses the features of geocoding and the routing engine in Oracle Spatial and Graph. NDM supports commercial street network data from data providers Here (formerly Navteq) and Tom Tom in their respective Oracle formats.

Modeling Capabilities

- Model and represent any point along a link for all analysis functions, such as specific addresses in street networks with any number of properties on the nodes and links.

Figure 15: San Francisco Park: Finding Parking Availability Analytical Capabilities.

- Model partial-link paths (sub-paths).
- Customize link and node properties (e.g., costs).
- Perform path analysis with multiple link and node properties (e.g., distance/time/hops costs).
- Perform partitioning of logical networks (e.g., social and biochemical pathway networks) based on metrics appropriate to the application.
- Compute the shortest route connecting a given set of nodes (the traveling salesperson problem).
- Generate a polygon representing the region that is reachable from a given node with a specific cost. A typical application is the generation of drive time and drive distance polygons.

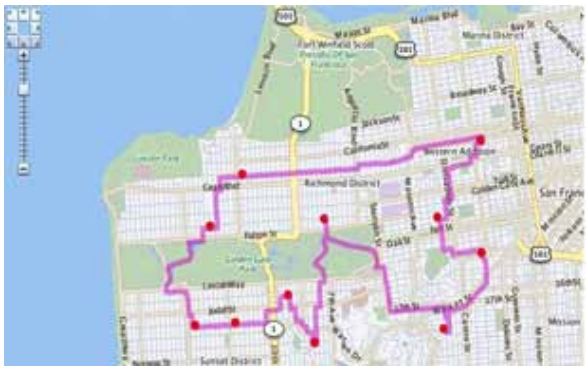


Figure 16: Calculating Optimal Paths on Roadways.

- Generate the shortest path on a hierarchical network, where links are prioritized by property (e.g., highways, local roads), to support queries such as finding the route between two addresses that favors highways over local roads as much as possible.
- Compute a buffer based on network cost; the buffer representation contains coverage and cost information.
- Compute K shortest paths between two nodes.

Real-World Feature Modeling and Analysis

Oracle Spatial and Graph NDM simplifies feature editing and analysis by providing a feature analysis function that associates

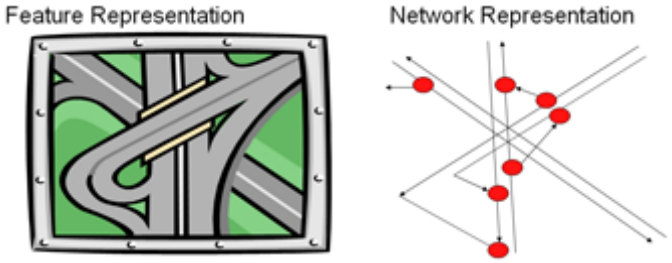


Figure 17: Mapping Features to a Network.

feature representations with network elements.

Feature modeling simplifies application development by associating real-world objects with network elements. For example, if a utility network application needs to find affected households when a substation experiences a power failure, it is necessary to associate the application features (substations, power lines, and transformers) with network elements (links and nodes). Feature modeling maintains these relationships through feature metadata, simplifying application development and maintenance.

Network Modeling with Time; Multimodal Transportation Routing

Figure 18: Developing Routing, Geocoding, Mapping Solutions.

Most real-world networks have a time element. Travel times on road segments vary with the time of day. Utility networks experience different demand loads based on seasonal demand and the time of day. To support temporal analysis of such networks, Oracle Spatial and Graph supports the modeling of networks containing a time dimension. NDM supports queries such as finding the fastest travel route for a specified time of day. NDM supports modeling and analysis of multimodal transportation networks, and computing the fastest paths on multimodal transportation networks.

Summary

Graphs are central to a new category of social media and linked data applications emerging in the life sciences, finance, media, and public sectors. In addition, graphs have been a fundamental data model to support a range of enterprise applications that require the modeling and analysis of network models common in telecommunications, utilities, and transportation. The availability of two graph models in Oracle Spatial and Graph ensures that applications developers can choose the appropriate model for their application. In doing so, they can leverage the industry's most scalable and secure platform for enterprise-scale graph applications.

More information about Oracle Spatial and Graph is available on oracle.com: **http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html**. ▲

Bill Beauregard is a senior principal product manager with Oracle Corporation.

Disruptive Innovations?

by Iggy Fernandez

Iggy Fernandez

Excerpted from The Twelve Days of NoSQL (**http://iggyfernandez.wordpress.com/**)

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

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

Established players usually ignore disruptive innovations because they do not see them as a threat to their bottom lines. In fact, they are more than happy to abandon the low-margin segments of the market, and their profitability actually increases when they do so. The disruptive technologies eventually take over most of the market.

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

So where lies the dilemma? Christensen explains:

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

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

Schemaless Design

One of the innovations of the NoSQL camp is "schemaless design." Data is stored in "blobs" and documents and the NoSQL database management system does not police their structure.

Let's do a thought experiment. Suppose that we don't have a schema and let's suppose that the following facts are known.

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

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

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

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

The above data could be modeled with the following three relations. It is certainly disruptive to suggest that this be done on

the fly by the database management system but not outside the realm of possibility.

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

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

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

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

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

Thanks to the hard work of many people at Hadapt from several different groups, including the science team who developed an initial design of the feature, the engineering team who continued to refine the design and integrate it into Hadapt's SQL-on-Hadoop solution, and the customer solutions team who worked with early customers to test and collect feedback on the functionality of this feature, this feature is now available in Hadapt." (**http://hadapt.com/blog/2013/10/28/all-sql-on-hadoop-solutions-are-missing-the-point-of-hadoop/**)

This is not really new ground. Oracle Database provides the ability to convert XML documents into relational tables (**http://docs.oracle.com/cd/E11882_01/appdev.112/e23094/xdb01int.htm#ADXDB0120**), though it ought to be possible to view XML data as tables while physically storing it in XML format in order to benefit certain use cases. It should also be possible to redundantly store data in both XML and relational formats in order to benefit other use cases.

In *Extending the Database Relational Model to Capture More Meaning*, Dr. Codd explains how a "formatted database" arises from an unorganized collection of facts:

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

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

Big Data in a Nutshell

In 1998, Sergey Brin and Larry Page invented the PageRank algorithm for ranking web pages (*The Anatomy of a Large-Scale Hypertextual Web Search Engine* by Brin and Page) and founded Google.

The PageRank algorithm required very large matrix-vector multiplications (*Mining of Massive Datasets* Ch. 5 by Rajaraman and Ullman) so the MapReduce technique was invented to handle such large computations (*MapReduce: Simplified Data Processing on Large Clusters*).

Smart people then realized that the MapReduce technique could be used for other classes of problems, and an open-source project called Hadoop was created to popularize the MapReduce technique (**http://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/**).

Other smart people realized that MapReduce could handle the operations of relational algebra such as join, anti-join, semi-join, union, difference, and intersection (*Mining of Massive Datasets* Ch. 2) and began looking at the possibility of processing large volumes of business data (a.k.a. "Big Data") better and more cheaply than mainstream database management systems.

Initially programmers had to write Java code for the "mappers" and "reducers" used by MapReduce. However, smart people soon realized that SQL queries could be automatically translated into the necessary Java code, and "SQL-on-Hadoop" was born. Big Data thus became about processing large volumes of business data with SQL but better and more cheaply than mainstream database management systems.

However, the smart people have now realized that MapReduce is not the best solution for low-latency queries (**http://gigaom.com/2013/11/06/facebook-open-sources-its-sql-on-hadoop-engine-and-the-web-rejoices/**). Big Data has finally become about processing large volumes of business data with SQL but better and more cheaply than mainstream database management systems and with or without MapReduce.

That's the fast-moving story of Big Data in a nutshell. ▲

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

"And so the choice that you have to make is: Should we make better products that we can sell for better profits to our best customers. Or maybe we ought to make worse products that none of our customers would buy that would ruin our margins. What should we do? And that really is the dilemma."

NoCOUG Winter Conference

Session Descriptions

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

–Keynote–

Oracle Database In-Memory Option—The Next Big Thing
—Juan Loaiza, Oracle Corporation. 9:30–10:30

The Oracle Database In-Memory option dramatically accelerates the performance of analytic queries by storing data in a highly optimized columnar in-memory format. Analytic operations run in real-time and return completely current and consistent data. A unique dual-format approach ensures outstanding performance and complete data consistency for all workloads. Oracle Database In-Memory automatically maintains data in both the existing Oracle row format for OLTP operations, and a new purely in-memory column format optimized for analytical processing. Both formats are simultaneously active and transactionally consistent. Unlike other in-memory approaches that represent data exclusively in column format thus delivering poor OLTP performance, Oracle Database In-Memory eliminates the need for expensive overhead to maintain analytic indexes, and therefore greatly accelerates OLTP operations. "*Virtually every existing application that runs on top of the Oracle database will run dramatically faster by simply turning on the new In-Memory feature. Customers don't have to make any changes to their applications whatsoever; they simply flip on the in-memory switch, and the Oracle database immediately starts scanning data at a rate of billions or tens of billions of rows per second.*" New applications that were previously impractical due to performance limitations can be developed with existing tools in use today. All of Oracle's industry-leading availability, security, and management features continue to work unchanged.

As Senior Vice President of Systems Technology at Oracle, Juan Loaiza is in charge of developing the mission-critical capabilities of Oracle Database, including data and transaction management, high availability, performance, backup and recovery, enterprise replication, and Oracle Exadata. Juan joined the Oracle Database development organization in 1988 and has contributed to every Oracle Database release since Oracle Version 6.

–Auditorium–

Oracle Graph: Graph Features in Oracle Database 12*c*—*Zhe Wu, Oracle Corporation* . . . . . 11:00–12:00 **Editor's Pick**

With every release since its introduction over ten years ago, Oracle Spatial and Graph has delivered the most advanced spatial and graph data management capabilities to database management systems. Formerly known as Oracle Spatial option, Oracle Spatial and Graph underlines its existing graph capabilities, which comprise the most robust, mature database graph technologies available in the industry. Oracle Spatial and Graph provides two graph data models: Network Data Model graph (NDM), and RDF Semantic Graph. NDM is a property graph model used to model and analyze physical and logical networks used in industries such as transportation, logistics, and utilities. NDM persistently manages the network connectivity in the database, while a Java API provides fast in-memory graph analytics, including: shortest path, nearest neighbors, within cost, and reachability. RDF Semantic Graph supports the World Wide Web Consortium (W3C) Resource Description Framework (RDF) standards. It provides RDF data management, querying and inferencing that are commonly used in a variety of applications ranging from semantic data integration to social network analysis and linked open data applications. It includes RDF views on relational data, more extensive inferencing capabilities, the latest SPARQL support, spatial RDF data support, and graph analytics and statistics support. Oracle Spatial and Graph RDF support has become the industry's leading open, scalable, and secure RDF database.

Oracle Spatial: Spatial Features in Oracle Database 12*c*
—Jean Ihm, Oracle Corporation. 1:00–2:00

With every release since its introduction over ten years ago, Oracle Spatial and Graph has delivered the most advanced spatial and graph data management capabilities for database management systems. The geospatial data features are designed to support the most complex requirements found in geographic information systems (GISs), enterprise applications, and location-enabled business and web applications. It extends the Oracle Locator spatial query and analysis features in Oracle Database with more advanced spatial analysis and processing capabilities. These geospatial data features include native support for advanced models and types such as GeoRaster (for geo-referenced imagery and gridded data); topology; 3D, including triangulated irregular networks (TINs) and point clouds (supporting LIDAR data); and linear referencing. Geocoding, a routing engine, and spatial web services conformant with Open Geospatial Consortium (OGC) and ISO standards are also included. These advanced features provide a complete platform for geospatial applications in many domains, including defense, land management, retail, insurance, and finance.

SQL: The Best Development Language for Big Data
—Hermann Baer, Oracle Corporation. 2:30–3:30

Large-scale data processing is undergoing tremendous transformations: new data sources are more readily available, and businesses are focusing more heavily on analytic solutions. Hadoop and other non-relational data sources are becoming more common, but working with and analyzing this data is hard. SQL, on the other hand, is the most commonly used language for data analysis. So how can we combine these two? This session discusses Oracle's analytical SQL capabilities and how complex analytical queries running on large to extremely large data sets are becoming a reality, even on data sources outside the relational world. You'll also learn how Oracle envisions the future of a unified analytical world.

(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:

- Aspect
- 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

Dharmendra Rai, *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)

Real-World Performance: Why Is My SQL Slow?
—Bob Carlin, Oracle Corporation 4:00–5:00

This session focuses on identifying problems with SQL statements and shows how to deliver true real-world performance gains without resorting to random hacking and guesses. Hidden within the Oracle Database Development team, the Real World Performance team focuses on your database performance. With its unique insight into how the database is designed to be used and having studied how it is actually used, the team can bring a new perspective on how to get the best performance from your database. The Real World Performance team uses healthy amounts of humor, irreverence, pragmatism, and passion for database performance to identify what Oracle users fail to appreciate and where the big gains are.

–Room 102–

DBA Boot Camp—*Kyle Hailey, Delphix* 11:00–5:00

Spend the whole day with ACE Director Kyle Hailey and dive deep into DBA topics such as I/O performance, SQL tuning, wait events, and database cloning.

Kyle Hailey was a principal designer for the Oracle Enterprise Manager performance pages. He is a member of Oracle Oak Table and the co-author of Oracle Insights: Tales of the Oak Table*, and he was a technical editor of Oracle Wait Interface. He holds a patent in the area of database performance diagnosis and has been a speaker at Hotsos, NOCOUG, RMOUG, NYCOUG, and Oracle World; he also organizes Oaktable World. Kyle teaches classes around the world on Oracle performance tuning. Currently Kyle works as a performance architect at Delphix, along with industry-leading software, kernel, and filesystem designers, to take corporate data management to a new level of agility.*

Lies, Damned Lies, and I/O Statistics 11:00–12:00

Given a description of gas dynamics and the atmosphere, you would be hard pressed to forecast tornadoes. The term "emergence" refers to the phenomena of surprising behaviors arising in complex systems. Modern storage systems are complex and full of emergent behavior that makes forecasting application I/O performance fiendishly difficult. In collaboration with Matt Hayward, Adam Leventhal, and others, Kyle Hailey has learned some rules of thumb for how to make accurate I/O performance forecasts. You'll learn about forecasting, benchmarking, and analyzing I/O performance in this talk.

Visual SQL Tuning 1:00–2:00

The load on the database is caused by SQL; thus it makes sense that performance bottlenecks are caused by poorly performing SQL statements. We will follow a solid step-by-step method for analyzing, understanding, and tuning these problem SQL statements though Visual SQL Tuning (VST) diagrams. VST is a method of laying out the tables and joins of a query graphically to indicate key features of the query in the graphics. Through the VST, you'll learn how to quickly visualize any coding errors in the query; discover flaws in the underlying database schema; and most important, find the best execution path through the query.

ASH Masters 2:30–3:30

Learn about ASH Masters, a GitHub repository of queries against Average Active History (ASH). Learn about ASH math, average active sessions, and the power and pitfalls of ASH queries. ASH queries will be shown that display over database performance from on high to diving deep down into the nitty-gritty of a performance bottleneck's internal workings. ASH is the most powerful source of performance analytic data in Oracle. Querying ASH can be difficult, with a number of pitfalls, but the rewards of correct data analysis of ASH make it the most powerful tool a DBA has for performance analysis. Learn how to query ASH correctly, and learn where to get powerful prewritten ASH queries.

Agile Data: Revolutionizing Database Cloning 4:00–5:00

Database virtualization allows the same datafiles to be shared by multiple clones, allowing almost instantaneous creation of new copies of databases with almost no disk footprint. Along with storage efficiency, database virtualization allows agile management of database copies. The data agility eliminates bottlenecks in development by removing wait time for creating database environments, allows developers to have their own full copy of the database, and provides QA and UAT with immediate copies of the development environments for testing. This presentation will compare and contrast different types of database virtualization from Oracle 11 CloneDB, Oracle 12*c* Snap Clones, 12*c* Snapshot Manager Utility, Oracle ZFS Appliance, Delphix Appliance, VMware Data Director, NetApp Snap Manager for Oracle, and EMC. We'll explain how database virtualization works and discuss the advantages and disadvantages of different approaches.

–Room 103–

ADF in a Nutshell—*Peter Koletzke, Quovera* 11:00–5:00

Learn how to build an application using the same tools Oracle uses to build Fusion Applications: Oracle Application Development Framework (ADF). This course focuses on introducing and demonstrating the core technologies needed to build an ADF application: ADF Business Components, ADF Faces, and ADF Controller. Half of this all-day session consists of hands-on labs where you can experience and practice development the ADF way.

Peter Koletzke is a technical director and principal instructor for Quovera in Palo Alto, California. He has over 30 years of industry experience and has presented at various Oracle users group conferences more than 310 times. Additionally, he has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools User Group Editor's Choice (three times), Oracle Development Tools User Group (ODTUG) Best Speaker, NY Oracle Users Group Editor's Choice (three times), East Coast Oracle/Southeastern Oracle Users Conference Oracle Designer Award, and the ODTUG Volunteer of the Year. Peter is an Oracle ACE Director, an Oracle Certified Master, and coauthor—variously with Duncan Mills, Avrom Roy-Faderman, and Dr. Paul Dorsey—of the Oracle Press (McGraw-Hill Professional) books Oracle JDeveloper 11*g* Handbook, Oracle JDeveloper 10*g* for Forms & PL/SQL Developers, Oracle JDeveloper 10*g* Handbook, Oracle9*i* JDeveloper Handbook, Oracle JDeveloper 3 Handbook, Oracle Developer Advanced Forms and Reports, Oracle Designer Handbook, 2nd Edition, *and* The Oracle Designer/2000 Handbook.

ADF in a Nutshell: Part I 11:00–12:00
ADF in a Nutshell: Part II 1:00–2:00
ADF in a Nutshell: Part III 2:30–3:30
ADF in a Nutshell: Part IV 4:00–5:00

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 Winter Conference Schedule

Thursday, February 20, 2014—Oracle Conference Center, Redwood City, 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:** *Oracle Database In-Memory Option—The Next Big Thing*—Juan Loaiza, Oracle Corporation
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Auditorium: *Oracle Graph: Graph Features in Oracle Database 12c*—Zhe Wu, Oracle Corp. **Journal Editor's Pick**
	Room 102: *Lies, Damned Lies, and I/O Statistics*—Kyle Hailey, Delphix
	Room 103: *ADF in a Nutshell: Part I*—Peter Koletzke, Quovera
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Auditorium: *Oracle Spatial: Spatial Features in Oracle Database 12c*—Jean Ihm, Oracle Corp.
	Room 102: *Visual SQL Tuning*—Kyle Hailey, Delphix
	Room 103: *ADF in a Nutshell: Part II*—Peter Koletzke, Quovera
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Auditorium: *SQL: The Best Development Language for Big Data*—Hermann Baer, Oracle Corp.
	Room 102: *ASH Masters*—Kyle Hailey, Delphix
	Room 103: *ADF in a Nutshell: Part III*—Peter Koletzke, Quovera
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Auditorium: *Real-World Performance: Why Is My SQL Slow?*—Bob Carlin, Oracle Corp.
	Room 102: *Agile Data: Revolutionizing Database Cloning*—Kyle Hailey, Delphix
	Room 103: *ADF in a Nutshell: Part IV*—Peter Koletzke, Quovera
5:00–	**NoCOUG Networking and No-Host Happy Hour at BJ's Restaurant & Brewhouse, San Mateo**

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

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo: Little Kid Using Laptop By Night by djedzura, Photos.com