InfoAdvisors

Your Master Data Is a Graph: Are You Ready?

Karen Lopez, InfoAdvisors

March 2015

InfoAdvisors

www.datamodel.com
karenlopez@Infoadvisors.com
@datachick

DATA STORY

Your Master Data Is a Graph- Are You Ready?

Your data wants to tell a story. It's not a *data problem*; it's *a data story.*

Master Data Management: Today

Your Master Data Management (MDM) program likely uses the same database technology as your transactional application: a mature, highly-tuned, relational database (RDBMS). You excel at relational databases because you have many years of experience working with them and most of your data lives there, so it makes sense to keep master data there. **But what if the data relationships, both the ones you know about and the ones you haven't yet discovered are telling you that they also need to be managed just as well?** Those *data stories* are becoming more and more important to competitive advantage as we enter into the future of data and business analytics.

Traditionally, MDM has included *Customer*, *Product*, *Accounts*, *Vendor*, *Partners* and any other highly shareable data in an enterprise. Sometimes reference data is included in master data.

As with many recent additions to data management approaches, we have a wide variety of definitions of MDM. These definitions range from a very physical one of *managing data in single data source or file* to a more conceptual definition of *managing data assets so that they can be accessed via a single service or application.* In both cases, though, the focus is on the data as a single concept about one type of thing.

We data professionals have struggled, though, with responding to business and business environment changes with our traditional implementations of master data. Given that master data, by definition is highly shared, this struggle tends to cost business agility in a way that ripples throughout the organization. Our architectures have focused on getting data to fit a single definition of the truth, something most of us come realize is not a feasible solution in the long run.

The Future of Master Data Management

MDM programs that attempted to physically persist data in one single location continue to struggle with the realities of modern information technology. Most enterprise organizations use vendor applications: customer relationship management (CRM) systems, work management systems, accounts payable, accounts receivable, point of sale systems, etc. Due to this approach it's not always feasible to move *all* master data to a single location. Even with a CRM system in place, we typically end up with customer information maintained in several systems. The same is true for product and accounting data as well.

The most successful programs will not strive to find a single physical location for *all* data, but will provide the standards, tools and services necessary to provide a consistent vision of enterprise data. There will be data we can store in one place, using the technologies that best fit its data story. Data will also likely be found in multiple physical systems due to the increasing use of packaged applications as well for performance and geographically-distributed processing needs. Once we understand our environment, we can architect solutions that build upon those needs.

The most successful programs will not strive to find a single physical location for all data of one concept, but will provide the standards, tools and services necessary to provide a consistent vision of enterprise data.

The future of master data management will derive value from data and its relationships to other data. MDM will be about supplying consistent, meaningful views of master data. In many cases, we will be able to unify data into one location, especially to optimize for query performance and data fit. Graph databases offer exactly that type of data/performance fit, as we will see below. In this paper, we discuss why your master data is a graph and how graph databases like Neo4j are the best technologies for master data.

How Data Relationships Drive Better Insights

When we think of data relationships, we often first think of foreign keys, those relational database features that allow us to link rows of data to other rows of data. But that's just one type of relationship. **In fact, those foreign keys are technically just constraints on our data**. The foreign key relationship between TRANSACTION and TRANSACTION LINE ITEMs is there to ensure (provide a constraint) that we don't have TRANSACTION LINE ITEMs hanging around without a corresponding TRANSACTION. The more valuable data stories focus on much more complex, multi-faceted questions:

- Which products tend to be purchased together?
- Which products tend to be purchased by the same customer and the members of their household on a regular basis? What about groups of customers?
- What products and services should we recommend to customers, separately and as a bundle?
- What other factors might come in to play for how people make purchase decisions?

More on these types of question later.

Graphs: Theory and Real World

Graphs are data structures that describe both data and their relationships. The most commonly recognized forms of a graph are networks and hierarchies. Graphs have nodes and relationships between those nodes. Both the nodes and the relationships can have properties. We can also apply labels to nodes.

Networks

A network comprises a set of nodes, with relationships between them.

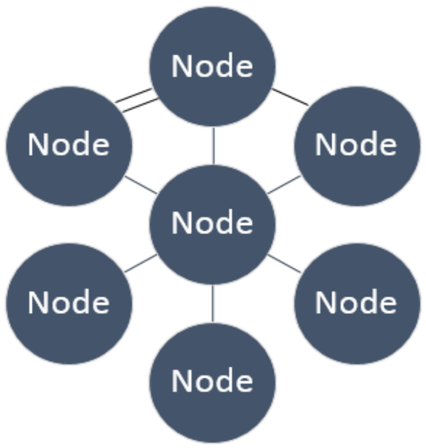


Figure 1- Network

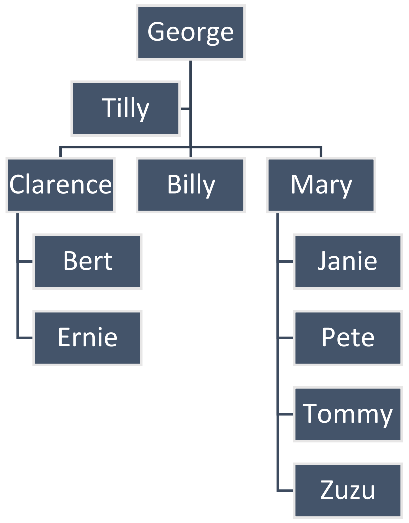
You might think of a network of friends, servers, databases or customers. In a graph database, the nodes in a graph don't have to be all the same type of thing – and neither do the relationships. Instead of defining an entity that imposes a standard on all instances, we build nodes and relationships that can each have their own set of properties. The nodes in *Figure 1- Network* might be *customers*, *orders*, *products* and *promotions*. Each of these could have different types of properties. Not just by type of node, but also by instance of each node. For the customer node, we might have given name and family name, birthdate and acquisition date as properties. But we might not have all of those properties for all customers. In a relational database, we would define common properties but set some of them to NULL if they don't have values for that instance. The relationships we have between nodes gets the same power: we can store properties about each relationship without imposing those properties on all of them. That's right, we can store properties about the relationships as well. This is a key difference from relational databases, where we would need to convert a relationship to a table to store metadata about that relationships.

We don't have to have the same structure for all instances in a graph database. As we will see later, below, this flexibility pays off for real-world data stories.

Hierarchies & Trees

A hierarchy is a structure where nodes have other nodes above them and below them. A tree is a hierarchy with branches (multiples of nodes related to a parent node.) Of course, pure hierarchies rarely exist in the real world – more on that in the next section. We think of organizational reporting structures and supervisory assignments as examples of hierarchies. In a typical supervisory hierarchy, one employee reports to exactly one other employee. And that employee may have many other persons reporting to him. We might even be told these are the business rules around this reporting structure.

Figure 2-Employee Hierarchy



We can easily implement this in a relational database, with a recursive relationship on an EMPLOYEE table. A small hierarchy such as in *Figure 2-Employee Hierarchy,* maintaining those reporting relationships is easy. As soon as we model a much larger set, though, maintenance gets more expensive. We often need to use workarounds such as special hierarchy datatypes or calculated columns that keep track of levels and pointers. Then what happens when a node gets a promotion? All those relationships must be reset and recalculated. If that node participates in many types of hierarchies, it's possible that many relationships must be reset and recalculated. **This data story is so complex there are specialized books and tutorials on how to design and maintain these workarounds.** Even with all those tricks, we still struggle to maintain the data and application performance levels.

Hierarchies in the Real World are Actually Graphs

Each employee in *Figure 2-Employee Hierarchy* is a node. The lines in the reporting structure are implied relationships: "*Reports To"*. We can traverse up and down that reporting structure to find all the employees that report to one employee and vice versa. But in the real world, strict hierarchies are rare. Even in our reporting example, employees often report to more than one person. Sometimes these relationships exist for transitional reasons (job shadowing, job sharing, job coverage) and other times because there are many different types of *"Reports To"* relationships (*manages*, *mentors*, *project manages*, etc.) Figure 3 - "Reports To" Network shows a more realistic, richer set of relationships. Not only are we tracking straight reporting to relationships, we are recording administrative, protective, and even familiar relationships between employees. We have multiple reporting relationships, some skipping levels in our previous example of constrained relationships.

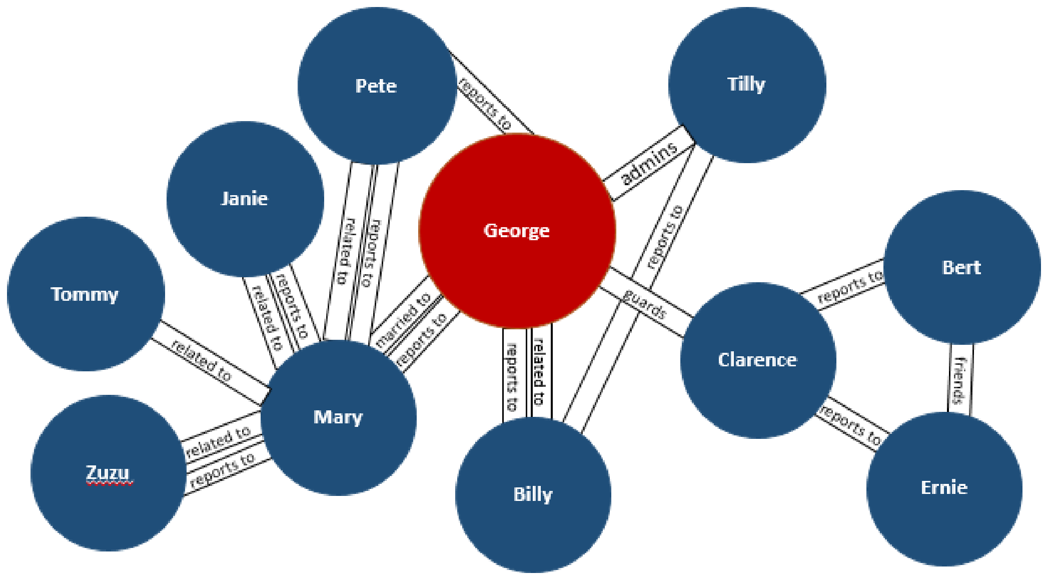


Figure 3 - "Reports To" Network

Even though we think of them as hierarchies, most business hierarchies are actually networks. Once you have one real-life complexity of multiple reporting relationships with multiple types of relationships your vision of a beautiful, perfect hierarchy is destroyed. This happens with organizational charts, product “hierarchies”, locations and document. Once you see your world as not fitting a true hierarchy, you can see all the graphs around you.

Data Modeling and Graph

In traditional data management, we prepare logical and physical data models. The logical data model describes business requirements for a data story and the physical data model specifies how data is to be persisted in a database. In a relational design, we apply a common structure to each instance of an entity. We have a CUSTOMER entity and all those entities share the same set of properties or attributes. This means we must discover and document all the properties we want to support prior to building the database and importing data.

In a graph database, the logical model ***is*** the physical model. You can even think about the graph model as a *service model*. We can do whiteboard-level modeling of nodes and relationships, then add properties and labels. At that point we have completed all the data modeling we need to do to implement in a graph database. If we have traditional logical data models we can even pull in properties we already know about into our graph model. Because the logical model is the only model, going from data to database takes significantly less time and fewer resources (modelers, architects, DBAs and developers) than building relational master data solutions.

We can also label each instance so that we can query our data based on a role they fill. For instance, we might want to query only organizational customers in certain queries.

In *Figure 4 - Employee Roles, Activities, Skills, Degrees, Teams Graph*, we can see that both nodes and relationships have varied properties. This is our data model, just as we might draw it on a whiteboard.

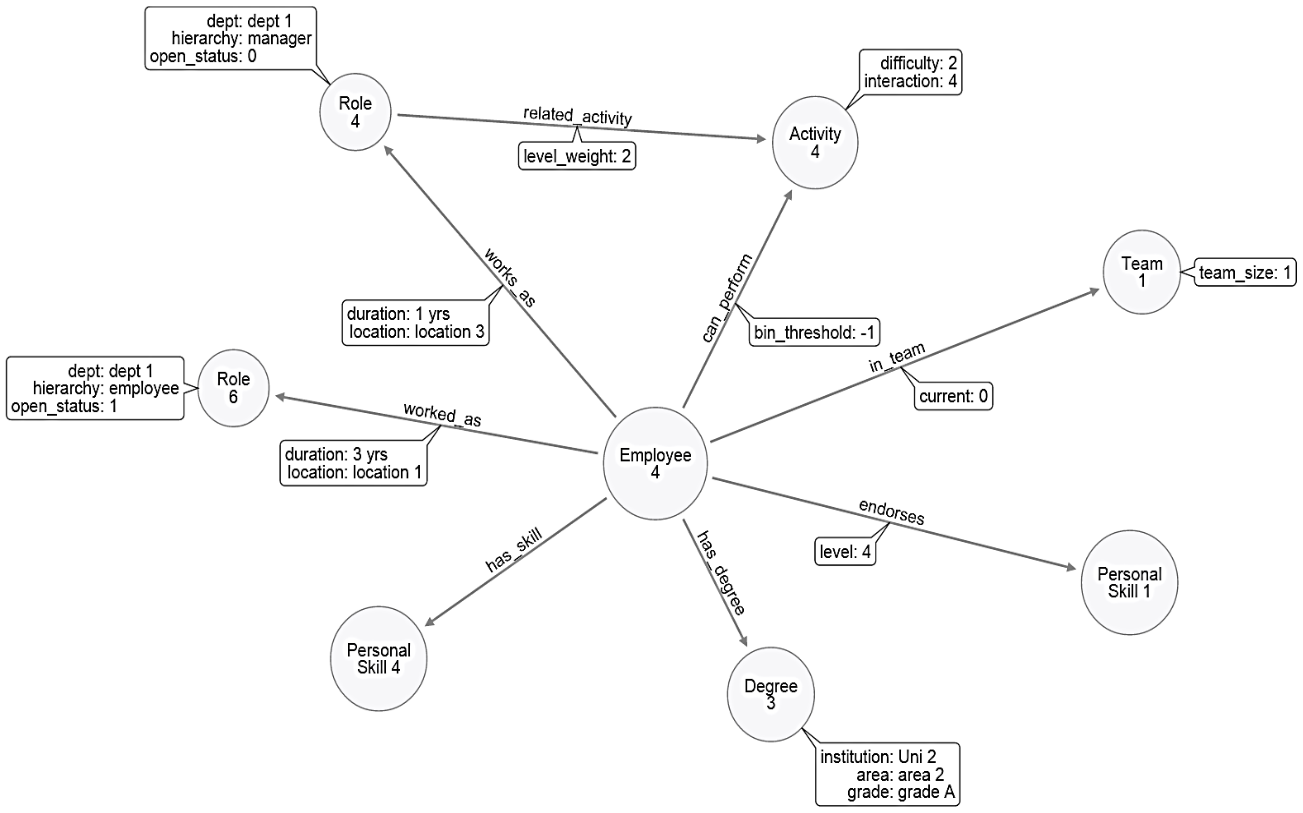


Figure 4 - Employee Roles, Activities, Skills, Degrees, Teams Graph

Graphs in the Real World

Many organizations are using graph databases to focus on the data relationships in their master data to be more competitive.

Retail Transformation at a Communications Organization

An information and communications that offers mobile, TV and digital services upgraded their customer experiences by adopting Neo4j in the production of their retail offerings. Their previous retail process required sales associates to use multiple systems to sell a mobile handset, accessories, promotions and plans to a walk-in customer. Their technologies were optimized for maintaining their product data and not for providing efficient customer service, nor for helping product management to design efficient product bundles. They attempted to build hard relationships between product and services items, but then paid the cost of having to adjust and re-engineer those hard relationships when new properties needed to be added. They realized they needed a more flexible, more responsive solution a constantly changing items and promotions world.

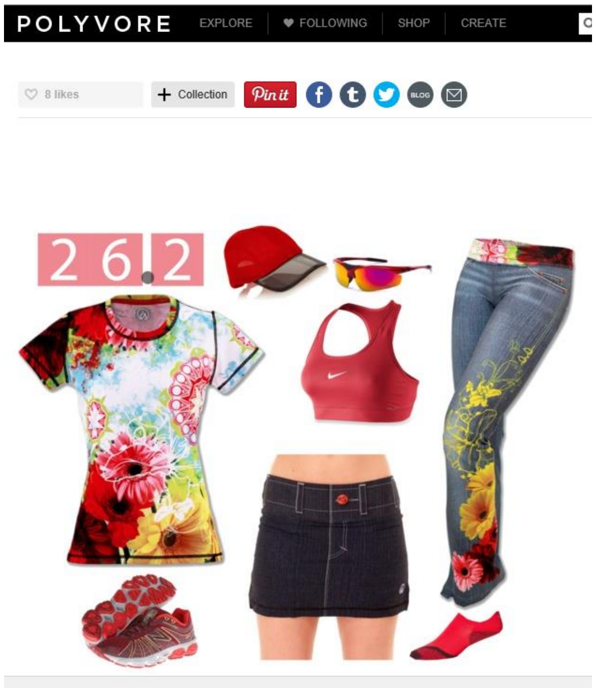
Today they use Neo4j to allow buyers to pre-package groups of items so that sales associates can quickly provide bundles for customers. Leveraging the flexibility and the focus on product and promotion relationships in Neo4j, they have been able to reduce the time required to put together product bundles from one month to 10 days. In store, they can help customers choose the items they want and sell them other services or items to bundle well with those items. Using the many different types of data relationships in their product and customer master data is key to improving customer service, completing orders and decreasing cost of sales, as well as substantially reducing queue times at their physical stores.

This retailer hosts nearly one million nodes and millions of relationships between those nodes and continues to scale up as volume and number of users increases. They are already working on adding Neo4j to cable and digital TV services and products to bring the same competitive advantages to their other lines of business.

Leveraging Multi-vendor Product Combinations at Polyvore

POLYVORE

Polyvore allows shoppers to put together sets of clothing and products in a highly visual, interactive format. A shopper (a *stylist* in Polyvore terms) can pull products, their metadata and prices from any website using a *clipper* tool, then enhance these sets with graphics, text and personal photos. The stylist can then put together sets of sets, called collections, to build even more relationships among items.



Polyvore users compose millions of these sets a year, creating a wealth of buying intentions and influence for other shoppers. Today there are millions of retail items (SKUs) in this database of user-generated content. Users build these collections of items based on their own fashion tastes and trends.

Polyvore can ask questions about what items "go together" and infer from those answers other products that may influence interests and sales of additional items. Users can also tag products with the relevant characteristics of each product. These tags also become relationships for product data, further enhancing the rich dataset being crowdsourced.

Brands can also participate in the Polyvore community by clipping their items into the item database, then looking at trends and popularity, based on community sharing and "likes" of their products in relationship to other products.

This use case of product data, metadata and stylist-assigned tags is all about the relationships between these concepts. A traditional product catalog would focus on the products and their properties. Yet in the Polyvore case, the power is in these shopper-generated data relationships. It's not just ***all about the data***; it's ***all about the relationships***. In using the power of the graph in Neo4j, Polyvore is able to identify not just trends, but the factors that are influencing those trends spanning prices, styles, colors, sizes and anything else the community tags items to be. Brands gain insights into *how* their products are coveted by the most influential purchasers in e-commerce.

Why a Graph Database?

Now that we've covered some real life uses of a graph database, let's talk about the scenarios where their use makes the most sense. In many cases the potential for graph databases is clear: social media, friend-of-a-friend networks, actual technology networks. In those cases, the relationships between users and things are clearly key data structures – we even call them networks. But there are many enterprise use cases where the value in data relationships is just as strong, even if we don't think of them as networks.

Discovering New Correlations within Data

Imagine if we mashed up in-store purchase data with data about a customer's online shopping experiences:

- Visits to store online presence
- Items viewed, clicked on, rated, wish-listed, shared via social media
- Items added to shopping cart and the clicks that preceded that addition

In that data, though, there will be valuable data relationships that a competitive retailer will want to uncover. While possible to manage and query all these in a relational database, would be very time-consuming and expensive to do so. In addition, as the volume of that data increases, the more expensive (in processing time) for a relational database to return results. The more data we collect, the more real-time results will become less and less feasible. Think about the following questions we might want to ask of this data:

- What sorts of items tend to be purchased together by a category of customer? What about over time?
- Which members of a household tend to make which types of purchases on which days? How does this vary by distance from a store?

Deduplication of Data

Duplication of master data happens due to acquisition of third party data and mergers of organizations. The use of vendor packages can also lead to duplication of master data. Graph databases can provide faster and easier methods for identifying whether customer "Karen Lopez" in Toronto, ON is the same customer as "Karena Lopez" in Scarborough, ON. We can do this by leveraging the advantages of graph queries that can analyze all the data relationships across several instances to give a confidence score each record being the same person. Understanding that two Karen Lopez's with differing addresses get the same vehicle serviced or visit a location at the same time allows us to understand the patterns and the relationships in the data. We can use those data relationships to identify master data where we might not already have common customer identifier.

Impact and What-if Analysis

In addition to managing master data, business users need to plan for changes to it. For instance, product data is often the target of re-engineering and rebranding. Marketing and product manages want to be able to answer questions such as:

- If I change the rules about use of this promotion on these products, what would the impact be?
- If I change the bundle for this set of product and services, how would that impact our average order totals?
- If we retired this bundle, how many customers would be impacted and what bundles should we offer them as a replacement?
- Given the customer demographics of a new sales territory, which products and promotions might be the best to advertise?

All of these questions focus on the data relationships across sets of master data.

360 Degree view of Master Data

It's one thing to model and build a view of CUSTOMER or PRODUCT. But businesses don't make decisions based on data silos. We, for the most part, have customers because they buy products or want to buy products. We have products because we expect customers to buy them.

As we've seen in the data stories above, having master data in a graph makes asking these questions easier, more flexible, and faster to process

Modern Hybrid Data Architectures

The modern master data architecture no longer depends on one technology. In fact, the most competitive organizations understand that finding the right tool for the right job means that answers can be produced faster and with greater insights.

View of a Modern Master Data Architecture

In *Figure 5 - Modern Data* you can see that we still have data stories that are best fit to relational databases. We also have data stories that can make best use of a variety of NoSQL technologies, including graph databases. In fact, a January 2015 Gartner report shows that CIOs and data professionals require a variety of database technologies:

> *The rise of polyglot persistence requires CIOs and IM leaders to evaluate new approaches to data consistency across multiple persistence types, optimized data access to multiple persistence types, and integration of data across multiple persistence types.*[i]

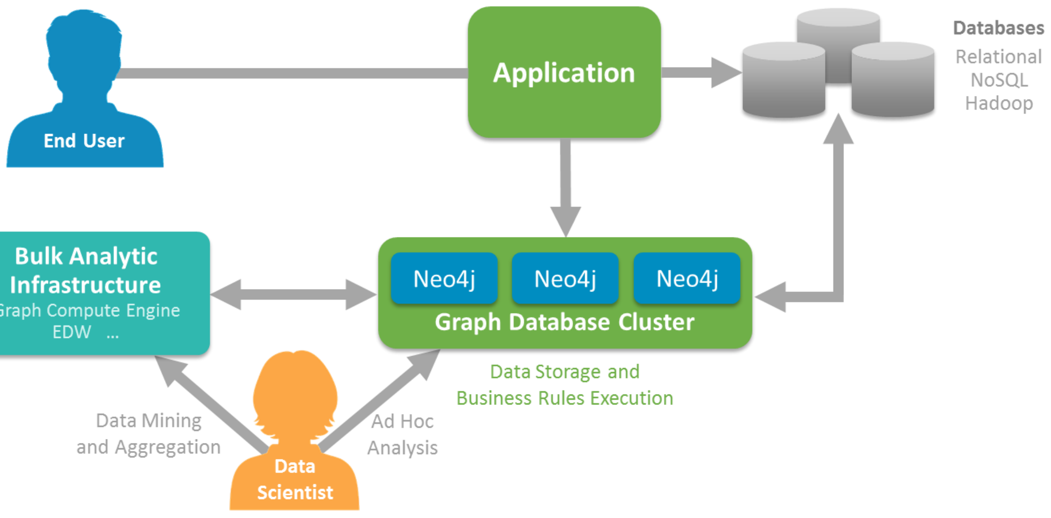


Figure 5 - Modern Data Architecture
(Neo4j. Used with permission)

Organizations that want not only to collect data, but use it to support business decisions in real time. That means leveraging technologies that provide the best value for the types of questions we ask of our data. Relational databases aren't going away; we still have strong use cases for highly structured, business-rule driven datastores. These stories demand the types of constraints and "sameness" that relational databases were designed to provide. We need those technologies to enforce integrity in our data, to ensure we are collecting all the data we need to complete a transaction and to provide a highly-predictable, stable dataset for using in our business processes. However, those strengths of a relational systems can get in the way of other use case for managing and analyzing data. That's why some data stores demand less structure, fewer constraints and less consistency between instances of data. In fact, that's one of the driving reasons for implementing a NoSQL solution.

Performance

With scalability built into the underlying design, Neo4j allows master data solutions to grow as fast as the business does. As a database built specifically for storing and processing graph queries, unlocking the value in data relationships is significantly faster than other SQL or NoSQL solutions.

Flexibility

A graph database allows us to maintain data without a prescriptive set of attributes or properties that are the same for each node – or even for a relationship between nodes. This works to our advantage when we are bringing together data from multiple sources. For example, we might have customer data from our online retail presence, our bricks and mortar stores, plus customer data we've acquired through an opt-in mailing list. Each of those data sources likely have overlapping and different attributes, with a variety of completeness. If we tried to import this data into a relational database, we'd be forced to either turn off data quality constraints or to cleanse and reject some data in order to meet integrity rules. But we don't want to do that with this data – we want to bring it all in to a database so that we can work with the data as it is.

We are also free to add new data, on the fly, because we aren't trying to force all nodes or relationships must have the same type of metadata. We can bring in the data we want, relate it to existing data (or not) and continue on asking questions. Since the logical model is the physical model, we don't have to re-architect our solution every time we want to gain new insights about our master data.

This real-world, flexible management of data and data relationships gives organizations the power to ask more questions, to get more answers, than if we determined all them at data acquisition time. The flexibility to add more data relationships, in real time, allows organizations to continue to enhance their data stories.

Summary

We've looked at the future of master data management for competitive organizations being a hybrid, polyglot architecture. We know that the value in master data management isn't just in the data about things, but also in the powerful relationships in the data. And we have seen how difficult is it to develop fast and flexible systems for traversing all those relationships. Traditional relational systems have their benefits, but traversing a network of master data is expensive, less business agile and slow.

Graph databases such as Neo4j are built from the ground up to support graph data stories, the same stories that your data wants to tell. The fact that the logical data model is the physical model means that data professionals can deliver answers to these data relationship questions faster and with more flexibility than ever before.

What's Next?

You've learned that graph databases have a place in modern enterprise data architectures; what should you do next?

1. Read Graph Databases http://graphdatabases.com and for a deeper dive into the theory and practical application of graph technologies.
2. Take the graph database online course http://neo4j.com/graphacademy/online-course
3. Talk to key data-driven business users in your organization about the types of questions they'd love to ask their data, but have not been able to do so due to technological limitations or costs.
4. Learn about RDBMS to Graph Databases concepts and tools http://neo4j.com/developer/graph-db-vs-rdbms
5. Help those business users understand that there are data relationships that may not be apparent in the existing data structures.
6. Download Neo4j http://neo4j.com or use Amazon Machine Image (AMI) http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html to build an environment in the cloud.
7. Discover new insights into your own master data that will give your organization a competitive advantage based on data relationships in your existing data.

About the Author

Karen López is Senior Project Manager and Architect at InfoAdvisors. She has more than twenty years of experience in helping organizations implement large, multi-project programs.

InfoAdvisors is a Toronto-based data management consulting firm. We specialize in the practical application of data management. Our philosophy is based on assessing the cost, benefit, and risk of any technique to meet the specific needs of our client organizations.

We want you to love your data.

Find us at datamodel.com

About Neo Technology

Neo Technology is the creator of Neo4j, the world's leading graph database that brings data relationships to the fore. From companies offering personalized product and service recommendations; to websites adding social capabilities; to telcos diagnosing network issues; to enterprises reimagining master data, identity, and access models; organizations adopt graph databases to model, store and query both data and its relationships. Large enterprises like Walmart, eBay, UBS, Nomura, Cisco, HP, and Telenor, as well as startups like CrunchBase, Medium, Polyvore, and Zephyr Health use Neo4j to unlock business value from data relationships.

Neo Technology is headquartered in San Mateo, CA, with offices in Sweden, UK, Germany, and Malaysia. For more information, please visit Neo4j.com.

[i] Randall, L., Huedecker, N., Feinberg, D. (January 2015) "The Rise of Polyglot Persistence Demands Your Consideration", *Gartner Research,* https://www.gartner.com/doc/2954719/rise-polyglot-persistence-demands-consideration